Lifted Model Construction without Normalisation: A Vectorised Approach to Exploit Symmetries in Factor Graphs

Malte Luttermann[1], Ralf Möller[2] and **Marcel Gehrke[2]**
[1]German Research Center for Artificial Intelligence (DFKI), Lübeck
[2]Institute for Humanities-Centered Artificial Intelligence, University of Hamburg
malte.luttermann@dfki.de,{ralf.moeller,marcel.gehrke}@uni-hamburg.de

Abstract

Lifted probabilistic inference exploits symmetries in a probabilistic model to allow for tractable probabilistic inference with respect to domain sizes of logical variables. We found that the current state-of-the-art algorithm to construct a lifted representation in form of a parametric factor graph misses symmetries between factors that are exchangeable but scaled differently, thereby leading to a less compact representation. In this paper, we propose a generalisation of the advanced colour passing (ACP) algorithm, which is the state of the art to construct a parametric factor graph. Our proposed algorithm allows for potentials of factors to be scaled arbitrarily and efficiently detects more symmetries than the original ACP algorithm. By detecting strictly more symmetries than ACP, our algorithm significantly reduces online query times for probabilistic inference when the resulting model is applied, which we also confirm in our experiments.

1 Introduction

Parametric factor graphs (PFGs) are probabilistic relational models, i.e., they combine probabilistic models and relational logic (which can be seen as first-order logic with known universes) to efficiently reason about objects and their relationships under uncertainty. To allow for tractable probabilistic inference (e.g., inference requiring polynomial time) with respect to domain sizes of logical variables, PFGs use representatives of indistinguishable objects to represent groups of random variables (randvars), thereby yielding a more compact model that can be exploited by lifted inference algorithms for faster inference. Here, probabilistic inference (or just inference for short) refers to the task of computing marginal distributions of randvars given observations for other randvars (see Appendix A for more details). Clearly, to run a lifted inference algorithm on a PFG, the PFG has to be constructed first. The current state-of-the-art algorithm to construct a PFG is the advanced colour passing (ACP) algorithm. The ACP algorithm begins with a propositional model in form of a factor graph (FG) and exploits symmetries therein to obtain a PFG entailing equivalent semantics as the initial FG. During the course of ACP, potentials of factors are compared to decide whether factors are equivalent and thus might be grouped. However, all potentials of the factors must be scaled equally for ACP to be able to detect symmetries between factors. In other words, ACP fails to detect symmetries between factors that are exchangeable but whose potentials differ only by a scalar, thereby leading to a less compact lifted representation if potentials are not normalised before running ACP. In this paper, we solve the problem of constructing a PFG from a given FG such that the resulting PFG entails equivalent semantics as the initial FG and exchangeable factors are detected independent of the scale of their potentials. We therefore allow potentials to be learned from different data sources without having to perform a normalisation step while at the same time obtaining a more compact representation for lifted inference than the output of ACP.

In previous work, Poole (2003) introduces PFGs and lifted variable elimination as an inference algorithm to carry out lifted probabilistic inference in PFGs. Lifted inference exploits symmetries in a probabilistic model by using a representative of indistinguishable objects for computations

Malte Luttermann, Ralf Möller, Marcel Gehrke, Lifted Model Construction without Normalisation: A Vectorised Approach to Exploit Symmetries in Factor Graphs. *Proceedings of the Third Learning on Graphs Conference (LoG 2024)*, PMLR 269, Virtual Event, November 26–29, 2024.

while maintaining exact answers (Niepert and Van den Broeck, 2014). By using logical variables in parameterised randvars (PRVs) to represent groups of indistinguishable randvars, lifted variable elimination operating on PFGs is able to allow for tractable probabilistic inference with respect to domain sizes of logical variables (Taghipour et al., 2013a). After its first introduction, lifted variable elimination has been steadily refined by many researchers to reach its current form (De Salvo Braz et al., 2005, 2006; Milch et al., 2008; Kisyński and Poole, 2009; Taghipour et al., 2013b; Braun and Möller, 2018). Recently, Luttermann et al. (2024a,b) extend PFGs to incorporate causal knowledge, thereby allowing for lifted causal inference in addition to lifted probabilistic inference. In any case (purely probabilistic or causal), the construction of a PFG (or its causal extension, respectively) is necessary to apply lifted inference algorithms afterwards. The "CompressFactorGraph" algorithm (Kersting et al., 2009; Ahmadi et al., 2013) builds on work by Singla and Domingos (2008) and detects symmetries in an FG to obtain possible groups of randvars and factors by deploying a colour passing procedure similar to the Weisfeiler-Leman algorithm (Weisfeiler and Leman, 1968), which is commonly used to test for graph isomorphism. To obtain a valid PFG, the resulting groups must be represented by introducing logical variables in PRVs, and the current state-of-the-art algorithm to construct a valid PFG entailing equivalent semantics as an initially given FG is the ACP algorithm (Luttermann et al., 2024c,d,e). While ACP successfully constructs a valid PFG from a given FG, it requires all potentials of factors to be scaled by the same scalar, which imposes a serious limitation for practical applications, e.g., when potentials are learned from various data sources and normalisation is undesirable due to floating point arithmetic issues.

To circumvent the requirement of equally scaled potential values in all factors, we propose a modification of the ACP algorithm that encodes potential values of factors as vectors. In an earlier work, Gehrke et al. (2020) show that potentials of factors can be conceived as vectors such that the cosine similarity provides a useful measure to check whether factors "behave similarly", thereby allowing to keep symmetries over time and to avoid groundings during temporal probabilistic inference. By using vector representations of factors' potentials, the potentials do not have to be scaled by the same scalar and thus, we make use of this property already during the construction procedure of the PFG in this paper. Detecting symmetries independent of scalars already during the construction of the lifted representation (i.e., before lifted inference takes place) yields a more compact representation right from the beginning and thereby significantly speeds up online inference afterwards. We formally show that using such a vector representation maintains equivalent semantics. Further, we demonstrate that the vector representation can easily be incorporated into the ACP algorithm to obtain a more compact representation, which we also confirm in our empirical evaluation.

The remaining part of this paper is structured as follows. First, we provide the necessary background information and introduce notations. We begin to recap FGs and afterwards formalise the problem of detecting exchangeable factors. Thereafter, we take a closer look at the problem of detecting exchangeable factors independent of the scale of their potentials and present our approach that makes use of vector representations of potentials to solve this problem. We then embed the vectorised approach into the framework of the ACP algorithm to obtain a generalisation of ACP, which we evaluate empirically to demonstrate its practical effectiveness before we conclude.

2 Background

We begin by defining FGs as propositional probabilistic graphical models. An FG compactly encodes a full joint probability distribution over a set of randvars by factorising the distribution into a product of factors (Frey et al., 1997; Kschischang et al., 2001).

Definition 1 (Factor Graph)**.** *An* FG $G = (\boldsymbol{V}, \boldsymbol{E})$ *is an undirected bipartite graph consisting of a 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), as well as a set of edges* $\boldsymbol{E} \subseteq \boldsymbol{R} \times \boldsymbol{\Phi}$*. The term* $\mathrm{range}(R_i)$ *denotes the possible values of a randvar* R_i*. There is an edge between a variable node* R_i *and a factor node* ϕ_j *in* $\boldsymbol{E}$ *if* R_i *appears in the argument list of* ϕ_j*. The argument list* $\mathcal{A}_j$ *of a factor* $\phi_j(\mathcal{A}_j)$ *is a sequence of randvars from* $\boldsymbol{R}$*. A factor is a function that maps its arguments to a positive real number, called potential. The semantics of* G *is given by*

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

where Z *is the normalisation constant and* $\mathcal{A}_j$ *denotes the randvars occurring in* ϕ_j*'s argument list.*

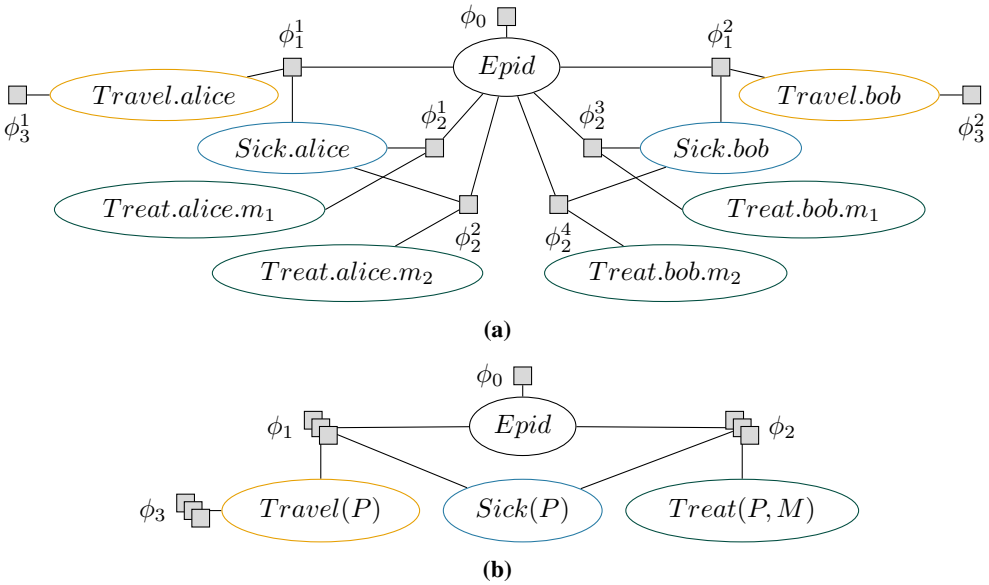


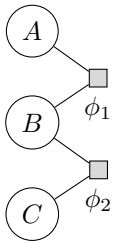
Figure 1: (a) An FG encoding a full joint probability distribution for an epidemic example (Hoffmann et al., 2022), (b) a PFG corresponding to the lifted representation of the FG shown in Fig. 1a. The mappings of argument values to potentials of the factors are omitted for brevity.

Example 1. *Figure 1a shows an FG for an epidemic example. The FG consists of two people ($alice$ and bob) as well as two possible medications (m_1 and m_2) for treatment. For each person, there are two Boolean randvars (that is, randvars having a Boolean range) $Sick$ and $Travel$, indicating whether the person is sick and travels, respectively. Moreover, there is another Boolean randvar $Treat$ for each combination of person and medication, specifying whether the person is treated with the medication. The Boolean randvar $Epid$ states whether an epidemic is present.*

Lifted inference algorithms exploit symmetries in FGs to allow for tractable probabilistic inference with respect to domain sizes of logical variables. In a lifted representation such as a PFG, parameterised randvars and parametric factors represent sets of randvars and factors, respectively (Poole, 2003). Symmetries in FGs frequently occur in relational models and are highly relevant in many real world domains. For example, in the epidemic domain, each person influences the probability of an epidemic in the same way—that is, the probability of having an epidemic depends on the number of sick people and not on individual people being sick. In other words, the probability for an epidemic is the same if there is a single sick person and the remaining people in the universe are not sick, independent of whether $alice$ or bob is sick. Analogously, there are symmetries in many other domains, e.g., for movies the popularity of an actor influences the success of a movie in the same way for each actor being part of the movie, and so on.

Example 2. *A PFG corresponding to the lifted representation of the FG illustrated in Fig. 1a is shown in Fig. 1b. Here, two logical variables P and M with domains $\mathrm{dom}(P) = \{alice, bob\}$ and $\mathrm{dom}(M) = \{m_1, m_2\}$ are introduced to represent groups of indistinguishable people and medications, respectively. Further, there are parametric factors that represent groups of factors, e.g., ϕ_1 represents ϕ_1^1 and ϕ_1^2. The underlying assumption is that ϕ_1^1 and ϕ_1^2 are exchangeable and hence encode equivalent semantics. By using logical variables in parameterised randvars, the number of parameterised randvars and parametric factors in the graph remains constant even if the number of people and medications increases.*

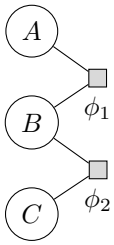
To detect symmetries in an FG and obtain a PFG for lifted inference, the ACP algorithm (Luttermann et al., 2024c) is the current state of the art. The ACP algorithm employs a colour passing routine to identify symmetric subgraphs and transforms a given FG into a PFG entailing equivalent semantics as the initial FG. A formal description of the ACP algorithm is given in Appendix B. For now, it is important to understand that ACP has to detect exchangeable factors during the course of the algorithm. Exchangeable factors are factors that encode equivalent semantics and play a crucial role when detecting and exploiting symmetries in an FG. In the next section, we investigate the problem



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

(a)



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	$\alpha \cdot \varphi_1$
true	false	$\alpha \cdot \varphi_2$
false	true	$\alpha \cdot \varphi_3$
false	false	$\alpha \cdot \varphi_4$

(b)

Figure 2: (a) An exemplary FG, (b) another FG encoding equivalent semantics as the FG shown in (a) but containing a factor ϕ_2 whose potentials are scaled by factor $\alpha \in \mathbb{R}^+$.

of detecting exchangeable factors in FGs independent of the scale of their potentials in detail and provide an efficient solution to this problem.

3 Avoiding Normalisation During Lifted Model Construction

Before we formally define the notion of exchangeable factors, let us take a look at the upcoming example, which illustrates the idea of having differently scaled potentials in exchangeable factors.

Example 3. *Consider again the FG depicted in Fig. 1a and let us assume we want to learn the potentials of the factors from observed data (e.g., by counting the occurrences of combinations of range values). For example, to obtain the potentials for the factor $\phi_1^1(Travel.alice, Sick.alice, Epid)$, occurrences of $alice$ becoming sick when travelling are counted. Analogously, occurrences of bob becoming sick when travelling are counted to obtain the potentials of $\phi_1^2(Travel.bob, Sick.bob, Epid)$. If both factors encode equivalent potentials, they can be grouped (as in Fig. 1b). Now, assume $alice$ travels twice as much as bob and both become sick on every second trip on average. In consequence, $alice$ and bob "behave identically" with respect to becoming sick when travelling but the potentials of ϕ_1^1 and ϕ_1^2 lie on a different scale—in this particular example, the potentials of ϕ_1^1 are equal to the potentials of ϕ_1^2 times two (because $alice$ travels twice as much as bob).*

A straightforward solution to deal with different scales of potentials is to normalise potentials. However, we cannot always assume that a given FG contains normalised potentials by default and in practical applications, the normalisation of potentials is often undesirable as it results in additional floating point arithmetics causing numerical issues. We thus develop a solution that does not require potentials to be normalised but still detects exchangeable factors, which we formally define next.

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 scalar $\alpha \in \mathbb{R}^+$ and a permutation π of $\{1, \ldots, n\}$ such that for all $r_1, \ldots, r_n \in \times_{i=1}^{n} \mathrm{range}(R_i)$ it holds that $\phi_1(r_1, \ldots, r_n) = \alpha \cdot \phi_2(r_{\pi(1)}, \ldots, r_{\pi(n)})$. Factors that represent equivalent potentials are called* exchangeable factors.

Note that as a necessary condition, exchangeable factors must be defined over the same function domain and hence must have the same number of arguments.

Example 4. *Take a look at the FG depicted in Fig. 2a, which features two factors ϕ_1 and ϕ_2 that map to the exact same potential values $\varphi_i \in \mathbb{R}^+$, $i \in \{1, \ldots, 4\}$. In this scenario, both tables of mappings from assignments of arguments to potential values are identical (i.e., $\alpha = 1$ and π is the identity function) and hence, it is easy to tell that ϕ_1 and ϕ_2 are exchangeable. If we now consider the FG shown in Fig. 2b, we can observe that the potential values of ϕ_2 are scaled by a factor $\alpha \in \mathbb{R}^+$. Despite the scaling, ϕ_1 and ϕ_2 encode equivalent semantics and thus are exchangeable.*

We remark that in general, π does not have to be the identity function, i.e., there might be situations where, for example, the argument positions of C and B in ϕ_2 are swapped and the potential values in the table read $\varphi_1, \varphi_3, \varphi_2, \varphi_4$ from top to bottom instead of $\varphi_1, \varphi_2, \varphi_3, \varphi_4$. Note that the potential mappings are still the same but their order is a different one. For now, we focus on the scalar

α and assume that π is the identity function, that is, the arguments are already ordered such that exchangeable arguments are located at the same argument positions if there are any exchangeable arguments. Later on, in Sec. 3.2, we also show how to deal with arbitrary permutations of arguments.

A fundamental insight is that factors, whose potential mappings are equivalent up to a scalar α, are semantically equivalent. The intuition here is that the *ratio* of the potentials within a factor is the relevant part for the semantics of the factor whereas the absolute values do not matter. For example, think of a factor ϕ that has two mappings in total, one for the assignment true and one for the assignment false. Semantically, it does not matter whether ϕ maps true to 1 and false to 2 or true to 2 and false to 4 because in both cases, ϕ weights the assignment false twice as much as the assignment true. The normalisation constant Z in Eq. (1) ensures that in both cases, the probability for true is $1/3$ and the probability for false is $2/3$. We next formalise this insight.

Theorem 1. *Let $G = (\boldsymbol{V}, \boldsymbol{E})$ denote an FG with $\boldsymbol{V} = \boldsymbol{R} \cup \boldsymbol{\Phi}$, where $\boldsymbol{R} = \{R_1, \ldots, R_n\}$ is a set of randvars and $\boldsymbol{\Phi} = \{\phi_1, \ldots, \phi_m\}$ is a set of factors. Then, scaling any factor $\phi_k \in \boldsymbol{\Phi}$ by a scalar $\alpha \in \mathbb{R}^+$ leaves the semantics of G unchanged.*

Proof. Recall that the semantics of G (before scaling) is given by $P_G = \frac{1}{Z} \prod_{j=1}^{m} \phi_j(\mathcal{A}_j)$, where $\mathcal{A}_j$ denotes the randvars occurring in ϕ_j's argument list and Z is the normalisation constant, defined as

$$Z = \sum_{\boldsymbol{a} \in \times_{i=1}^{n} \mathrm{range}(R_i)} \prod_{j=1}^{m} \phi_j(\mathcal{A}_j = \boldsymbol{a}_j), \tag{2}$$

where $\boldsymbol{a}_j$ denotes the assigned values to arguments $\mathcal{A}_j$ according to the assignment $\boldsymbol{a}$. Now, assume that $\phi_k \in \Phi$ is scaled by $\alpha \in \mathbb{R}^+$. Then, P_G changes to $P_G = \frac{1}{Z} \cdot \alpha \cdot \prod_{j=1}^{m} \phi_j(\mathcal{A}_j)$ and Z changes to $Z = \alpha \cdot \sum_{\boldsymbol{a} \in \times_{i=1}^{n} \mathrm{range}(R_i)} \prod_{j=1}^{m} \phi_j(\mathcal{A}_j = \boldsymbol{a}_j)$. In consequence, it holds that $P_G = \frac{1}{\alpha \cdot Z} \cdot \alpha \cdot \prod_{j=1}^{m} \phi_j(\mathcal{A}_j)$, which is equivalent to the original definition of P_G as α cancels out. □

Theorem 1 implies that it is also possible to scale various factors by different scalars without changing the semantics of the underlying model. Using this insight, it becomes clear that FGs can be further compressed by taking into account factors that are exchangeable up to a scalar α. We next show how exchangeable factors can efficiently be detected independent of the scaling factor α.

3.1 Dealing with Scaled Potentials

Previous work by Gehrke et al. (2020) shows that potentials of factors can be conceived as vectors such that the cosine similarity of the vectors can be used to check whether factors "behave identically", thereby avoiding groundings in temporal probabilistic inference. We apply the idea of representing potentials as vectors to detect exchangeable factors independent of a scaling factor already during the construction of a PFG to obtain a more compact model even before online inference takes place.

Definition 3 (Vector Representation of Factors)**.** *Let $\phi(R_1, \ldots, R_n)$ denote a factor. The* vector representation *of ϕ is defined as the vector $\vec{\phi} = (\phi(\boldsymbol{a}))_{\boldsymbol{a} \in \times_{i=1}^{n} \mathrm{range}(R_i)}$.*

Example 5. *Consider the factors ϕ_1 and ϕ_2 depicted in Fig. 2b. The vector representations of ϕ_1 and ϕ_2 are given by $\vec{\phi}_1 = (\varphi_1, \varphi_2, \varphi_3, \varphi_4)$ and $\vec{\phi}_2 = (\alpha\varphi_1, \alpha\varphi_2, \alpha\varphi_3, \alpha\varphi_4)$, respectively.*

Given a vector representation of a factor, the idea is that vectors of exchangeable factors point to the same direction in the vector space. Thus, the angle between those vectors can be computed to determine whether the factors are exchangeable because exchangeable factors have vector representations whose angle is equal to zero (i.e., they are collinear). The upcoming example illustrates this idea.

Example 6. *Take a look at Fig. 3, which shows the vector representations $\vec{\phi}_1 = (8, 2)$, $\vec{\phi}_2 = (4, 1)$, $\vec{\phi}_3 = (2, 2)$, and $\vec{\phi}_4 = (4.4, 3.6)$ for exemplary factors $\phi_1, \ldots, \phi_4$. To allow for a two-dimensional visualisation, every factor has two possible assignments (e.g., due to having a single Boolean argument). The angle between $\vec{\phi}_1$ and $\vec{\phi}_2$ is exactly zero, indicating that ϕ_1 and ϕ_2 are collinear and hence exchangeable, which can be verified as $\phi_1(\boldsymbol{a}) = 2 \cdot \phi_2(\boldsymbol{a})$ holds for all assignments $\boldsymbol{a}$. At the same time, the angle between, e.g., $\vec{\phi}_1$ and $\vec{\phi}_3$ is much larger than zero, indicating that ϕ_1 and ϕ_3 are not exchangeable. Further, observe that the angle between $\vec{\phi}_3$ and $\vec{\phi}_4$ is not exactly zero but close to zero, indicating that ϕ_3 and ϕ_4 are not equivalent but approximately equivalent.*

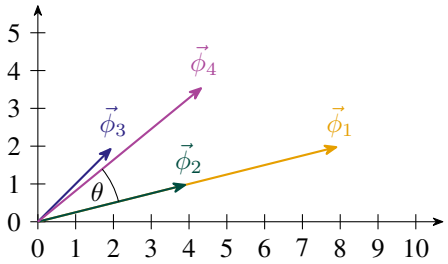


Figure 3: Vector representations for exemplary factors $\phi_1, \ldots, \phi_4$. For the sake of this example, every factor maps two possible assignments to a potential value each. The mappings of the factors are encoded as vectors and are given by $\vec{\phi}_1 = (8, 2)$ (i.e., ϕ_1 maps its first assignment to potential value 8 and the second assignment to potential value 2), $\vec{\phi}_2 = (4, 1)$, $\vec{\phi}_3 = (2, 2)$, and $\vec{\phi}_4 = (4.4, 3.6)$.

By using vector representations and the cosine similarity between them, exchangeable factors can efficiently be detected in practice. The cosine similarity between two vector representations of factors lies within the interval $[0, 1]$ and reaches its maximum value of one if the angle between the vectors is zero. To obtain a distance measure, we define the cosine distance as one minus the cosine similarity.

Definition 4 (Cosine Distance, Gehrke et al., 2020)**.** *Let $\phi_1(R_1, \ldots, R_n)$ and $\phi_2(R'_1, \ldots, R'_n)$ denote two factors. The* cosine distance *between ϕ_1 and ϕ_2 is defined as*

$$D_{\cos}(\phi_1, \phi_2) = 1 - \frac{\sum_{\boldsymbol{a} \in \times_{i=1}^{n} \mathrm{range}(R_i)} \phi_1(\boldsymbol{a}) \cdot \phi_2(\boldsymbol{a})}{\sqrt{\sum_{\boldsymbol{a} \in \times_{i=1}^{n} \mathrm{range}(R_i)} \phi_1(\boldsymbol{a})^2} \cdot \sqrt{\sum_{\boldsymbol{a} \in \times_{i=1}^{n} \mathrm{range}(R'_i)} \phi_2(\boldsymbol{a})^2}}. \tag{3}$$

If ϕ_1 and ϕ_2 are defined over different function domains, we define $D_{\cos}(\phi_1, \phi_2) = \infty$.

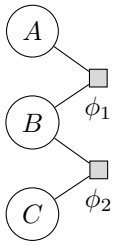
A fundamental advantage of using vector representations in combination with the cosine distance to search for exchangeable factors is that it is also possible to allow for a small deviation of $D_{\cos}$ from zero (e.g., dependent on a hyperparameter ε). While it is also conceivable to directly compare the tables of potential mappings and allowing for a deviation controlled by ε, a direct comparison of tables becomes sophisticated in settings where both a deviation and a scaling factor α have to be considered at the same time. Vector representations circumvent such issues and allow for a straightforward comparison of factors at any time. In this paper, however, we focus on the problem of exact lifted model construction, i.e., we aim to transform a given FG into a PFG entailing equivalent semantics as the initial FG. Allowing for a deviation between potentials results in the problem setup of approximate lifted model construction, which is a different problem not considered in detail here.

Before we continue to deal with permutations of arguments in addition to scaled potentials, we formally show that the cosine distance is a suitable measure to check for exchangeability.

Theorem 2. *Let $\phi_1(R_1, \ldots, R_n)$ and $\phi_2(R'_1, \ldots, R'_n)$ denote two factors. If ϕ_1 and ϕ_2 are exchangeable, then it holds that $D_{\cos}(\phi_1, \phi_2) = 0$.*

Proof Sketch. If ϕ_1 and ϕ_2 are exchangeable, there exists a scalar $\alpha \in \mathbb{R}^+$ and a permutation π of $\{1, \ldots, n\}$ such that for all $r_1, \ldots, r_n \in \times_{i=1}^{n} \mathrm{range}(R_i)$ it holds that $\phi_1(r_1, \ldots, r_n) = \alpha \cdot \phi_2(r_{\pi(1)}, \ldots, r_{\pi(n)})$. Without loss of generality, assume that the arguments of ϕ_2 are rearranged such that for all $r_1, \ldots, r_n \in \times_{i=1}^{n} \mathrm{range}(R_i)$ it holds that $\phi_1(r_1, \ldots, r_n) = \alpha \cdot \phi_2(r_1, \ldots, r_n)$. Then, entering $\phi_1(r_1, \ldots, r_n) = \alpha \cdot \phi_2(r_1, \ldots, r_n)$ into Eq. (3) yields $D_{\cos}(\phi_1, \phi_2) = 0$. □

Note that the cosine distance is a measure to check for collinearity of the vectors $\vec{\phi}_1$ and $\vec{\phi}_2$, that is, to check whether there exists a scalar α such that $\vec{\phi}_1 = \alpha \cdot \vec{\phi}_2$. As we only have to check for collinearity, we can avoid computing the cosine distance (and hence avoid floating point arithmetics during exchangeability checks) by checking the equality of products of potential values. The technical details for collinearity checks using only multiplication operations are given in Appendix D. However, also keep in mind that even when using the cosine distance to determine collinearity of vectors (which involves floating point arithmetics), we are able to avoid floating point numbers in the tables of



A	B	$\phi_1(A,B)$	b
true	true	φ_1	$[2,0]$
true	false	φ_2	$[1,1]$
false	true	φ_3	$[1,1]$
false	false	φ_4	$[0,2]$

B	C	$\phi_2(B,C)$	b
true	true	$\alpha \cdot \varphi_1$	$[2,0]$
true	false	$\alpha \cdot \varphi_3$	$[1,1]$
false	true	$\alpha \cdot \varphi_2$	$[1,1]$
false	false	$\alpha \cdot \varphi_4$	$[0,2]$

b	$\phi_1^{\succ}(b)$	$\phi_2^{\succ}(b)$
$[2,0]$	$\langle \varphi_1 \rangle$	$\langle \alpha \cdot \varphi_1 \rangle$
$[1,1]$	$\langle \varphi_2, \varphi_3 \rangle$	$\langle \alpha\varphi_3, \alpha\varphi_2 \rangle$
$[0,2]$	$\langle \varphi_4 \rangle$	$\langle \alpha \cdot \varphi_4 \rangle$

Figure 4: An FG entailing equivalent semantics as the FGs shown in Figs. 2a and 2b with corresponding buckets. Note that the arguments of the factor ϕ_2 are arranged in a different order than in Figs. 2a and 2b as B appears at position one and C at position two, whereas in the previous examples, C was at position one and B at position two.

potential mappings of the factors, which is the more important place to avoid floating point arithmetics (because during probabilistic inference, these numbers are multiplied).

So far, we did not pay attention to permutations of arguments when looking for exchangeable factors. In practice, however, we cannot assume that exchangeable arguments are always located at the same argument position in their respective factors. Therefore, in the next section, we investigate the problem of detecting exchangeable factors independent of the scale of their potentials while at the same time taking arbitrary permutations of their arguments into account.

3.2 Dealing with Permutations of Arguments

A straightforward approach to handle permutations of arguments when searching for exchangeable factors is to iterate over all possible argument permutations of one of the factors, rearrange its arguments and its table of potential mappings accordingly, and then compute the cosine distance as described in Sec. 3.1. If there exists a permutation such that the cosine distance is zero, the factors are exchangeable, otherwise they are not. Such an approach, however, is computationally expensive as it iterates over $O(n!)$ argument permutations for a factor with n arguments in the worst case. Luttermann et al. (2024d) introduce the *detection of exchangeable factors (DEFT)* algorithm, which avoids iterating over all permutations of arguments and thereby allows to efficiently detect exchangeable factors according to Def. 2 where $\alpha = 1$. In other words, DEFT is able to efficiently handle permutations of arguments but does not consider differently scaled potentials. We now combine the ideas of DEFT and the vector representation in combination with the cosine distance to handle both scalars different from one and permutations of arguments simultaneously.

The idea behind the DEFT algorithm is that a factor maps its arguments to potential values that can be distributed across so-called *buckets*. Buckets count the occurrences of specific range values in an assignment for a subset of a factor's arguments and within these buckets, possible permutations of arguments are heavily restricted such that not all permutations have to be considered. Before we illustrate the idea at an example, we give a formal definition of a bucket.

Definition 5 (Bucket, Luttermann et al., 2024d)**.** *Let $\phi(R_1, \ldots, R_n)$ denote a factor and let $\boldsymbol{S} \subseteq \{R_1, \ldots, R_n\}$ denote a subset of ϕ's arguments such that $\mathrm{range}(R_i) = \mathrm{range}(R_j)$ holds 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 potential 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 $\phi^{\succ}(b)$ the ordered multiset of potentials a bucket b is mapped to by ϕ (in order of their appearance in ϕ's table of potential mappings). The set of all buckets entailed by ϕ is denoted as $\mathcal{B}(\phi)$.*

Example 7. *Take a look at Fig. 4, which displays an FG entailing equivalent semantics as the FGs shown in Figs. 2a and 2b with corresponding buckets. In this example, B appears at position one and C at position two in ϕ_2 whereas in Figs. 2a and 2b, C was at position one and B at position two. Both ϕ_1 and ϕ_2 entail three buckets $\{(\mathrm{true}, 2), (\mathrm{false}, 0)\}$, $\{(\mathrm{true}, 1), (\mathrm{false}, 1)\}$, $\{(\mathrm{true}, 0), (\mathrm{false}, 2)\}$—or $[2,0]$, $[1,1]$, $[0,2]$ in shorthand notation. Every bucket corresponds to at least one assignment, e.g.,*

Algorithm 1 Detection of Exchangeable Factors without Normalisation

Input: Two 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.

if $n \neq m \vee \mathcal{B}(\phi_1) \neq \mathcal{B}(\phi_2)$ **then**
  return false
for each $b \in \mathcal{B}(\phi_1)$ **do** ▷ It holds that $\mathcal{B}(\phi_1) = \mathcal{B}(\phi_2)$
  $\alpha \leftarrow \max(\phi_1^{\succ}(b)) \,/\, \max(\phi_2^{\succ}(b))$
  if α differs from α for a previous bucket **then**
    return false
  $C_b \leftarrow$ Possible swaps to obtain $\phi_1^{\succ}(b) = \alpha \cdot \phi_2^{\succ}(b)$
if there exists a swap of ϕ_2's arguments in $\bigcap_{b \in \mathcal{B}(\phi_1)} C_b$ such that $D_{\cos}(\phi_1, \phi_2) = 0$ **then**
  return true
else
  return false

the bucket $[1, 1]$ *corresponds to all assignments that contain one* true *and one* false *value.* ϕ_1 *maps* $[1, 1]$ *to* $\langle \varphi_2, \varphi_3 \rangle$ *and* ϕ_2 *maps* $[1, 1]$ *to* $\langle \alpha\varphi_3, \alpha\varphi_2 \rangle$.

Luttermann et al. (2024d) show that two factors ϕ_1 and ϕ_2 are exchangeable (for the setting of $\alpha = 1$) 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. The DEFT algorithm exploits this property by checking for each bucket b whether arguments can be rearranged such that $\phi_1^{\succ}(b) = \phi_2^{\succ}(b)$ holds. The idea is that the potential values in the ordered multisets determine possible permutations of arguments and thus, DEFT avoids iterating over all permutations of arguments. For example, assuming that $\alpha = 1$ in Fig. 4, we know that $\alpha\varphi_3$ must be located at position one and $\alpha\varphi_2$ at position two in $\phi_2^{\succ}$ to match the order of $\phi_1^{\succ}$. The corresponding assignments of $\alpha\varphi_2$, i.e., $(\text{false}, \text{true})$, and $\alpha\varphi_3$, i.e., $(\text{true}, \text{false})$, are then used to determine possible positions of arguments to achieve that $\alpha\varphi_3$ is located at position one and $\alpha\varphi_2$ at position two in $\phi_2^{\succ}$. For now, it is sufficient to understand that identical potential values in $\phi_1^{\succ}$ and $\phi_2^{\succ}$ must be found. Further technical details about the DEFT algorithm are given in (Luttermann et al., 2024d). We next generalise the DEFT algorithm such that it is able to handle the setting of $\alpha \neq 1$ as well. A crucial observation is that the orders of the potential values in the ordered multisets must be identical up to the scaling factor α.

Theorem 3. *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* $\phi_1^{\succ}(b) = \alpha \cdot \phi_2^{\succ}(b)$ *for all buckets* b *entailed by the arguments of* ϕ_1 *and* ϕ_2*, where, by abuse of notation,* $\alpha \cdot \phi_2^{\succ}(b)$ *denotes the ordered multiset resulting from multiplying each potential value in* $\phi_2^{\succ}(b)$ *by* α.

Proof. For the first direction, it holds that ϕ_1 and ϕ_2 are exchangeable. According to Def. 2, there exists a scalar $\alpha \in \mathbb{R}^+$ and a permutation of ϕ_2's arguments such that ϕ_1 and ϕ_2 have identical tables of potential mappings up to the scaling factor α. In consequence, for every bucket b it holds that $\phi_1^{\succ}(b) = \alpha \cdot \phi_2^{\succ}(b)$ since both tables read identical potential values up to α from top to bottom.

For the second direction, it holds that $\phi_1^{\succ}(b) = \alpha \cdot \phi_2^{\succ}(b)$ for all buckets b. Converting the buckets back to tables of potential mappings then results in identical tables of potential mappings up to the scalar α, which implies that ϕ_1 and ϕ_2 are exchangeable. □

Using the insight from Thm. 3, the DEFT algorithm can be adapted to search for identical potential values up to scalar α in the ordered multisets of potential values, as shown in Alg. 1. To do so, α is determined first, which is done by computing $\alpha = \max(\phi_1^{\succ}(b)) \,/\, \max(\phi_2^{\succ}(b))$ in bucket b. Note that α must be identical for every bucket b, otherwise the two factors cannot be exchangeable. Having determined α, possible permutations of arguments are obtained by looking for identical potential values up to scalar α. Possible permutations of arguments are then verified (or rejected) using the cosine distance between ϕ_1 and ϕ_2 after rearranging ϕ_2's arguments.

Example 8. *Consider again the factors* ϕ_1 *and* ϕ_2 *depicted in Fig. 4. Rearranging* ϕ_2*'s arguments such that* B *is placed at position two and* C *at position one in* ϕ_2*'s argument list yields the table of potential mappings for* ϕ_2 *depicted in Fig. 2b and thus results in* $D_{\cos}(\phi_1, \phi_2) = 0$.

We next demonstrate the practical effectiveness of Alg. 1 in our empirical evaluation.

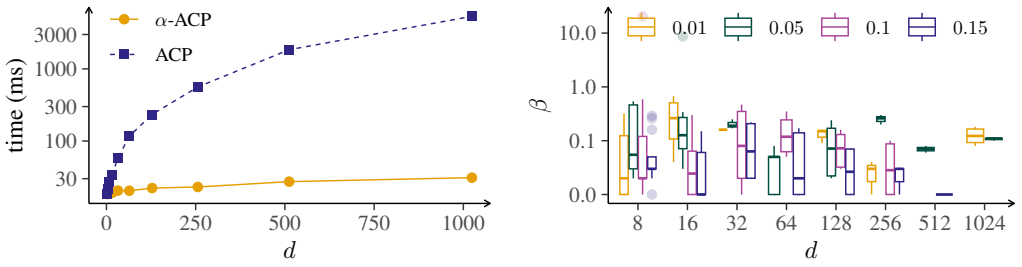


Figure 5: Average query times of lifted variable elimination on the output of ACP and α-ACP (left) and the average number β of queries after which the offline overhead of α-ACP amortises (right).

4 Experiments

To assess the effectiveness of Alg. 1 in practice, we compare the run times of running lifted variable elimination on the output of ACP in its original form and of running lifted variable elimination on the output of ACP extended by running Alg. 1 to detect exchangeable factors (α-ACP).[1] For our experiments, we generate FGs containing between $2d + 1$ and $d \cdot \lfloor \log_2(d) \rfloor + 2d + 1$ randvars as well as between $2d$ and $d \cdot \lfloor \log_2(d) \rfloor + d + 1$ factors, where the parameter $d \in \{2, 4, 8, 16, 32, 64, 128, 256, 512, 1024\}$ controls the size of the FG. In every FG, a proportion of $p \in \{0.01, 0.05, 0.1, 0.15\}$ of the factors is scaled by a scalar $\alpha \in \{1, \ldots, 10\}$ (chosen uniformly at random). For each choice of d, we pose three to four queries to each FG and report the average run time over all queries. Figure 5 displays the results. The left plot shows the average run times of lifted variable elimination on the output of ACP and α-ACP, respectively. As expected, lifted variable elimination runs significantly faster and is able to handle larger values of d if α-ACP instead of ACP is applied. Since α-ACP is able to detect exchangeable factors that are scaled differently, it detects strictly more symmetries than ACP resulting in a more compact model and hence in faster inference times. Clearly, the speedup depends on the proportion p of scaled factors and thus, we provide additional experimental results for each individual choice of p in Appendix E.

The boxplot on the right in Fig. 5 displays the average number β of queries after which the additional offline overhead of α-ACP compared to ACP amortises.[2] In particular, it holds that $\beta = \Delta_o \,/\, \Delta_g$, where Δ_o denotes the offline overhead of α-ACP (i.e., the difference of the offline run times required by α-ACP and ACP) and Δ_g denotes the online gain (i.e., the difference of the times required by lifted variable elimination run on the output of ACP and α-ACP). In other words, after β queries, the additional time needed by α-ACP to construct the PFG is saved due to faster inference times. Negative values for β are not displayed (hence the missing boxes for some p) as there is no overhead in these cases. The boxplot shows a box for each choice of p for every $d \geq 8$ and it becomes clear that the median value for β is always smaller than one. Apart from a single outlier, all values of β are smaller than ten. Thus, after a maximum of ten queries, the additional offline overhead of α-ACP amortises, showing that α-ACP works efficiently as it introduces almost no overhead.

5 Conclusion

In this paper, we generalise the ACP algorithm to detect exchangeable factors independent of the scale of their potentials without the requirement of normalising the potentials. Our proposed approach allows for arbitrary scalars and makes use of vector representations in combination with collinearity checks to efficiently detect exchangeable factors independent of their scale. We show that our approach maintains equivalent semantics and at the same time yields a more compact representation by detecting strictly more symmetries than the original ACP algorithm, thereby speeding up inference.

[1] Note that the run time required to perform probabilistic inference on a model directly depends on the graph size of the model, i.e., the presented run times also give information about the compactness of the models.

[2] The dots in the boxplot on the right in Fig. 5 represent outliers.

Acknowledgements

This work is funded by the BMBF project AnoMed 16KISA057.

References

David Poole. First-Order Probabilistic Inference. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 985–991. Morgan Kaufmann Publishers Inc., 2003. 1, 3

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. 2

Nima Taghipour, Daan Fierens, Guy Van den Broeck, Jesse Davis, and Hendrik Blockeel. Completeness Results for Lifted Variable Elimination. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics (AISTATS-2013)*, pages 572–580. PMLR, 2013a. 2

Rodrigo De Salvo Braz, Eyal Amir, and Dan Roth. Lifted First-Order Probabilistic Inference. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-2005)*, pages 1319–1325. Morgan Kaufmann Publishers Inc., 2005. 2

Rodrigo De Salvo Braz, Eyal Amir, and Dan Roth. MPE and Partial Inversion in Lifted Probabilistic Variable Elimination. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-2006)*, pages 1123–1130. AAAI Press, 2006. 2

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. 2

Jacek Kisyński and David Poole. Constraint Processing in Lifted Probabilistic Inference. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI-2009)*, pages 293–302. AUAI Press, 2009. 2

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, 2013b. 2

Tanya Braun and Ralf Möller. Parameterised Queries and Lifted Query Answering. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-2018)*, pages 4980–4986. IJCAI Organization, 2018. 2

Malte Luttermann, Mattis Hartwig, Tanya Braun, Ralf Möller, and Marcel Gehrke. Lifted Causal Inference in Relational Domains. In *Proceedings of the Third Conference on Causal Learning and Reasoning (CLeaR-2024)*, pages 827–842. PMLR, 2024a. 2

Malte Luttermann, Tanya Braun, Ralf Möller, and Marcel Gehrke. Estimating Causal Effects in Partially Directed Parametric Causal Factor Graphs. In *Proceedings of the Sixteenth International Conference on Scalable Uncertainty Management (SUM-2024)*, pages 265–280. Springer, 2024b. 2

Kristian Kersting, Babak Ahmadi, and Sriraam Natarajan. Counting Belief Propagation. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI-2009)*, pages 277–284. AUAI Press, 2009. 2, 12

Babak Ahmadi, Kristian Kersting, Martin Mladenov, and Sriraam Natarajan. Exploiting Symmetries for Scaling Loopy Belief Propagation and Relational Training. *Machine Learning*, 92:91–132, 2013. 2, 12

Parag Singla and Pedro Domingos. Lifted First-Order Belief Propagation. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-2008)*, pages 1094–1099. AAAI Press, 2008. 2

Boris Weisfeiler and Andrei A. Leman. The Reduction of a Graph to Canonical Form and the Algebra which Appears Therein. *NTI, Series*, 2:12–16, 1968. English translation by Grigory Ryabov available at `https://www.iti.zcu.cz/wl2018/pdf/wl_paper_translation.pdf`. 2

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, 2024c. 2, 3, 12, 13, 14

Malte Luttermann, Johann Machemer, and Marcel Gehrke. Efficient Detection of Exchangeable Factors in Factor Graphs. In *Proceedings of the Thirty-Seventh International Florida Artificial Intelligence Research Society Conference (FLAIRS-2024)*. Florida Online Journals, 2024d. 2, 7, 8, 12

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)*. PMLR, 2024e. 2, 13

Marcel Gehrke, Ralf Möller, and Tanya Braun. Taming Reasoning in Temporal Probabilistic Relational Models. In *Proceedings of the Twenty-Fourth European Conference on Artificial Intelligence (ECAI-2020)*, pages 2592–2599. IOS Press, 2020. 2, 5, 6

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. 2

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. 2

Moritz Hoffmann, Tanya Braun, and Möller. Lifted Division for Lifted Hugin Belief Propagation. In *Proceedings of the Twenty-Fifth International Conference on Artificial Intelligence and Statistics (AISTATS-2022)*, pages 6501–6510. PMLR, 2022. 3

A Probabilistic Inference in More Detail

The task of probabilistic inference describes the computation of marginal distributions of randvars given observations for other randvars. In other words, probabilistic inference refers to query answering, where a query is defined as follows.

Definition 6 (Query)**.** *A query $P(Q \mid E_1 = e_1, \ldots, E_k = e_k)$ consists of a query term Q and a set of events $\{E_j = e_j\}_{j=1}^k$ (called evidence), where Q and $E_1, \ldots, E_k$ are randvars. To query a specific probability instead of a probability distribution, the query term is an event $Q = q$.*

Example 9 (Probabilistic Inference)**.** *Take a look again at the FG depicted in Fig. 2a and assume we want to answer the query $P(B = \text{true})$.*

$$P(B = \text{true}) = \sum_{a \in \text{range}(A)} \sum_{c \in \text{range}(C)} P(A = a, B = \text{true}, C = c) \tag{4}$$

$$= \frac{1}{Z} \sum_{a \in \text{range}(A)} \sum_{c \in \text{range}(C)} \phi_1(a, \text{true}) \cdot \phi_2(c, \text{true}) \tag{5}$$

$$= \frac{1}{Z} \Big(\varphi_1 \varphi_1 + \varphi_1 \varphi_3 + \varphi_3 \varphi_1 + \varphi_3 \varphi_3 \Big). \tag{6}$$

Since $\phi_1(A, B)$ and $\phi_2(C, B)$ are exchangeable (i.e., it holds that $\phi_1(a, \text{true}) = \phi_2(c, \text{true})$ for all assignments where $a = c$), we can exploit this symmetry to simplify the computation and obtain

$$P(B = \text{true}) = \frac{1}{Z} \sum_{a \in \text{range}(A)} \sum_{c \in \text{range}(C)} \phi_1(a, \text{true}) \cdot \phi_2(c, \text{true}) \tag{7}$$

$$= \frac{1}{Z} \sum_{a \in \text{range}(A)} \phi_1(a, \text{true}) \sum_{c \in \text{range}(C)} \phi_2(c, \text{true}) \tag{8}$$

$$= \frac{1}{Z} \left(\sum_{a \in \text{range}(A)} \phi_1(a, \text{true}) \right)^2 \tag{9}$$

$$= \frac{1}{Z} \left(\sum_{c \in \text{range}(C)} \phi_2(c, \text{true}) \right)^2 \tag{10}$$

$$= \frac{1}{Z} \Big(\varphi_1 + \varphi_3 \Big)^2. \tag{11}$$

This example illustrates the idea of using a representative of indistinguishable objects for computations (here, either A or C can be chosen as a representative for the group consisting of A and C). This idea can be generalised to groups of k indistinguishable objects to significantly reduce the computational effort when answering queries.

B Formal Description of the Advanced Colour Passing Algorithm

The ACP algorithm (Luttermann et al., 2024c) extends the colour passing algorithm (Kersting et al., 2009; Ahmadi et al., 2013), thereby solving the problem of constructing a lifted representation in form of a PFG from a given FG. The idea of ACP is to first find symmetric subgraphs in a propositional FG and then group together these symmetric subgraphs. ACP searches for symmetries based on potentials of factors, on ranges and evidence of randvars, as well as on the graph structure by employing a colour passing routine. A formal description of the ACP algorithm is depicted in Alg. 2. We next explain the steps undertaken by ACP in more detail.

ACP begins with the colour assignment to variable nodes, meaning that all randvars having the same range and observed event are assigned the same colour. randvar with different ranges or different observed events are assigned distinct colours since those randvar do not "behave in the same way". Then, in Line 2, ACP assigns colours to factor nodes such that exchangeable factors encoding equivalent semantics according to Def. 2 are assigned the same colour. In its original form, ACP uses the DEFT algorithm (Luttermann et al., 2024d) to efficiently detect exchangeable factors and to assign colours accordingly. The DEFT algorithm deployed within ACP, however, is not able to

Algorithm 2 Advanced Colour Passing (reprinted from Luttermann et al., 2024c)

Input: An FG G with randvars $\boldsymbol{R} = \{R_1, \ldots, R_n\}$, factors $\boldsymbol{\Phi} = \{\phi_1, \ldots, \phi_m\}$, and evidence $\boldsymbol{E} = \{R_1 = r_1, \ldots, R_k = r_k\}$.
Output: A lifted representation G' in form of a PFG entailing equivalent semantics as G.

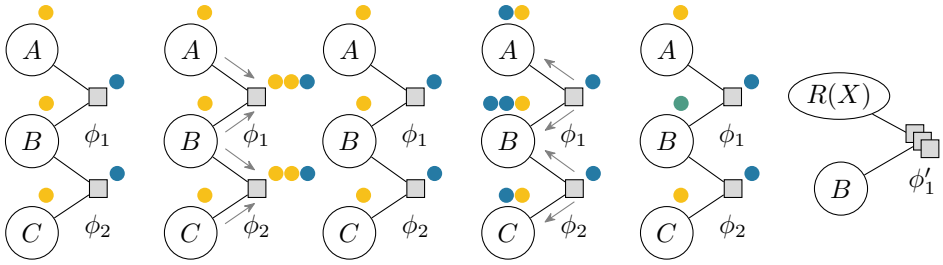
1: Assign each R_i a colour according to $\mathcal{R}(R_i)$ and $\boldsymbol{E}$
2: Assign each ϕ_i a colour according to order-independent potentials and rearrange arguments accordingly
3: **repeat**
4: **for** each factor $\phi \in \boldsymbol{\Phi}$ **do**
5: $signature_\phi \leftarrow [\,]$
6: **for** each randvar $R \in neighbours(G, \phi)$ **do** ▷ In order of appearance in ϕ
7: $append(signature_\phi, R.colour)$
8: $append(signature_\phi, \phi.colour)$
9: Group together all ϕs with the same signature
10: Assign each such cluster a unique colour
11: Set $\phi.colour$ correspondingly for all ϕs
12: **for** each randvar $R \in \boldsymbol{R}$ **do**
13: $signature_R \leftarrow [\,]$
14: **for** each factor $\phi \in neighbours(G, R)$ **do**
15: **if** ϕ is commutative w.r.t. $\boldsymbol{S}$ and $R \in \boldsymbol{S}$ **then**
16: $append(signature_R, (\phi.colour, 0))$
17: **else**
18: $append(signature_R, (\phi.colour, p(R, \phi)))$
19: Sort $signature_R$ according to colour
20: $append(signature_R, R.colour)$
21: Group together all Rs with the same signature
22: Assign each such cluster a unique colour
23: Set $R.colour$ correspondingly for all Rs
24: **until** grouping does not change
25: $G' \leftarrow$ construct PFG from groupings

handle factors with differently scaled potentials (i.e., it detects exchangeable factors according to Def. 2 only for $\alpha = 1$) and thus, our extension replaces the DEFT algorithm by Alg. 1 to detect exchangeable factors independent of the scale of their potentials in Line 2 of Alg. 2.

After the initial colour assignments, ACP runs a colour passing routine. ACP first passes the colours from each variable node to its neighbouring factor nodes and after a recolouring step to reduce communication overhead, each factor node ϕ sends its colour as well as the position $p(R, \phi)$ of R in ϕ's argument list to all of its neighbouring variable nodes R, again followed by a recolouring step. The procedure is then iterated until the identified groups do not change anymore. In the end, the determined groups are then used to construct a PFG entailing equivalent semantics as the input FG G.

Figure 6 illustrates ACP on an example input FG. In this example, all randvars are Boolean and there is no evidence available (i.e., $\boldsymbol{E} = \emptyset$). Initially, ACP assigns all randvars the same colour (e.g., yellow) because they have the same range (Boolean) and evidence (no evidence at all). As ϕ_1 and ϕ_2 encode equivalent semantics (they represent identical potentials), they are assigned the same colour (e.g., blue). The colours are then passed from variable nodes to factor nodes and as each factor has two neighbouring randvars, all factors receive the same messages. After recolouring the factors, their colour assignments remain identical to their initial assignments as they all received the same message. The purpose of the recolouring is mainly to reduce communication overhead. Afterwards, the factor nodes send their colours to their neighbouring variable nodes. Each message from a factor to a randvar contains the position of the randvar in the factor if the factor is not commutative[3], else the position is replaced by zero. For simplicity, there is no commutative factor in the example shown in Fig. 6. Thus, A receives a message $(\text{blue}, 1)$ from ϕ_1, B receives a message $(\text{blue}, 2)$ from ϕ_1 as well as a message $(\text{blue}, 2)$ from ϕ_2, and C receives a message $(\text{blue}, 1)$ from ϕ_2. Consequently, A

[3]A commutative factor is a factor which maps its arguments to the same potential value independent of the order of a (sub)set of its assigned values (Luttermann et al., 2024e).



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

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

Figure 6: A visualisation of the steps undertaken by Alg. 2 on an input FG with only Boolean randvars and no evidence (left). Colours are first passed from variable nodes to factor nodes, followed by a recolouring, and then passed back from factor nodes to variable nodes, again followed by a recolouring. The colour passing procedure is iterated until convergence and the resulting PFG is depicted on the right. This figure is reprinted from (Luttermann et al., 2024c).

and C receive identical messages (positions are not shown in Fig. 6) and after the recolouring step, A and C share the same colour while B is assigned a different colour. The groupings do not change in further iterations and the resulting PFG is shown on the right (where X has domain $\{A, C\}$).

For more details about the colour passing routine and the grouping of nodes, we refer the reader to (Luttermann et al., 2024c). The authors also demonstrate the benefits of a lifted representation in terms of speedup for probabilistic inference.

C Missing Proofs

Theorem 2. *Let $\phi_1(R_1, \ldots, R_n)$ and $\phi_2(R'_1, \ldots, R'_n)$ denote two factors. If ϕ_1 and ϕ_2 are exchangeable, then it holds that $D_{\cos}(\phi_1, \phi_2) = 0$.*

Proof. If ϕ_1 and ϕ_2 are exchangeable, there exists a scalar $\alpha \in \mathbb{R}^+$ and a permutation π of $\{1, \ldots, n\}$ such that for all $r_1, \ldots, r_n \in \times_{i=1}^n \text{range}(R_i)$ it holds that $\phi_1(r_1, \ldots, r_n) = \alpha \cdot \phi_2(r_{\pi(1)}, \ldots, r_{\pi(n)})$. Without loss of generality, assume that the arguments of ϕ_2 are rearranged such that for all $r_1, \ldots, r_n \in \times_{i=1}^n \text{range}(R_i)$ it holds that $\phi_1(r_1, \ldots, r_n) = \alpha \cdot \phi_2(r_1, \ldots, r_n)$. Then, entering $\phi_1(r_1, \ldots, r_n) = \alpha \cdot \phi_2(r_1, \ldots, r_n)$ into Eq. (3) yields

$$D_{\cos}(\phi_1, \phi_2) = 1 - \frac{\alpha \cdot \sum_{\boldsymbol{a} \in \times_{i=1}^n \text{range}(R_i)} \phi_2(\boldsymbol{a})^2}{\sqrt{\alpha^2 \cdot \sum_{\boldsymbol{a} \in \times_{i=1}^n \text{range}(R_i)} \phi_2(\boldsymbol{a})^2} \cdot \sqrt{\sum_{\boldsymbol{a} \in \times_{i=1}^n \text{range}(R_i)} \phi_2(\boldsymbol{a})^2}} \tag{12}$$

$$= 1 - \frac{\alpha \cdot \sum_{\boldsymbol{a} \in \times_{i=1}^n \text{range}(R_i)} \phi_2(\boldsymbol{a})^2}{\sqrt{\alpha^2} \cdot \sqrt{\sum_{\boldsymbol{a} \in \times_{i=1}^n \text{range}(R_i)} \phi_2(\boldsymbol{a})^2} \cdot \sqrt{\sum_{\boldsymbol{a} \in \times_{i=1}^n \text{range}(R_i)} \phi_2(\boldsymbol{a})^2}} \tag{13}$$

$$= 1 - \frac{\alpha \cdot \sum_{\boldsymbol{a} \in \times_{i=1}^n \text{range}(R_i)} \phi_2(\boldsymbol{a})^2}{\alpha \cdot \sum_{\boldsymbol{a} \in \times_{i=1}^n \text{range}(R_i)} \phi_2(\boldsymbol{a})^2} \tag{14}$$

$$= 0. \tag{15}$$

□

D Checking Collinearity of Vectors without Division Operations

Let $\vec{\phi}_1 = (\varphi_1, \ldots, \varphi_n)$ and $\vec{\phi}_2 = (\psi_1, \ldots, \psi_n)$ denote two vector representations of factors ϕ_1 and ϕ_2. Our goal is to check whether $\vec{\phi}_1$ and $\vec{\phi}_2$ are collinear, that is, whether there exists a scalar α such that $\vec{\phi}_1 = \alpha \cdot \vec{\phi}_2$, without using division or square root operations. By doing so, we avoid floating point arithmetic issues. In particular, if the potential values $\varphi_1, \ldots, \varphi_n$ and $\psi_1, \ldots, \psi_n$ are integers (which is the case if the initial FG is learned from data by counting occurrences of specific assignments), then no floating point numbers are involved in our calculations during exchangeability checks. Note that, even if we apply the cosine distance to check for collinearity, the model itself can still consist only of integer potential values and only the check for exchangeability involves floating point arithmetics in this case (as opposed to a model where potential values are normalised at the beginning, which results in floating point numbers for its potential values).

We next provide the technical details to check whether $\vec{\phi}_1 = \alpha \cdot \vec{\phi}_2$ holds by using only multiplication operations. If $\vec{\phi}_1$ and $\vec{\phi}_2$ are collinear, we have $\varphi_1 = \alpha \cdot \psi_1, \ldots, \varphi_n = \alpha \cdot \psi_n$ and hence $\alpha = \varphi_i \,/\, \psi_i$ for all $i \in \{1, \ldots, n\}$. Since $\alpha = \varphi_i \,/\, \psi_i$ holds for all $i \in \{1, \ldots, n\}$, we can enter $\alpha = \varphi_1 \,/\, \psi_1$ into the equation $\varphi_i = \alpha \cdot \psi_i$ and obtain

$$\varphi_i = \frac{\varphi_1}{\psi_1} \cdot \psi_i \tag{16}$$

$$\Leftrightarrow \varphi_i \cdot \psi_1 = \varphi_1 \cdot \psi_i. \tag{17}$$

In consequence, we can check whether $\vec{\phi}_1$ and $\vec{\phi}_2$ are collinear by verifying whether $\varphi_i \cdot \psi_1 = \varphi_1 \cdot \psi_i$ holds for all $i \in \{1, \ldots, n\}$. Verifying this equation involves only multiplication operations and hence avoids floating point arithmetics as much as possible.

To supplement Sec. 3.2, we next show that the aforementioned approach can also be extended to handle permutations of arguments. In case there exists a permutation π of $\{1, \ldots, n\}$ such that for all $r_1, \ldots, r_n \in \times_{i=1}^{n} \mathrm{range}(R_i)$ it holds that $\phi_1(r_1, \ldots, r_n) = \alpha \cdot \phi_2(r_{\pi(1)}, \ldots, r_{\pi(n)})$, it does not necessarily hold that $\alpha = \varphi_i \,/\, \psi_i$ for $i \in \{1, \ldots, n\}$. However, we know that $\alpha = \max_{i \in \{1,\ldots,n\}} \varphi_i \,/\, \max_{i \in \{1,\ldots,n\}} \psi_i$ (analogously for $\min$ instead of $\max$). At the same time, we know that in the first bucket, there is only a single potential value, which remains the same for all possible permutations π of $\{1, \ldots, n\}$ because the corresponding assignment assigns all arguments the same range value (e.g., true). We thus have $\alpha = \varphi_1 \,/\, \psi_1$ again, regardless of the order of arguments. Consequently, when searching for possible swaps to obtain $\phi_1^{\succ}(b) = \alpha \cdot \phi_2^{\succ}(b)$ in Line 7 of Alg. 1, we can find identical potential values up to the scalar α for each $\varphi_i \in \phi_1^{\succ}(b)$ in $\phi_2^{\succ}(b)$ by iterating over all potential values $\psi_i \in \phi_2^{\succ}(b)$ and checking whether $\varphi_i \cdot \psi_1 = \varphi_1 \cdot \psi_i$ holds. Having found identical potential values up to α, possible swaps to obtain $\phi_1^{\succ}(b) = \alpha \cdot \phi_2^{\succ}(b)$ are again determined by the positions of the identical potential values (up to α) in the ordered multisets. In particular, if $\varphi_i = \alpha \cdot \psi_i$ holds for $\varphi_i \in \phi_1^{\succ}(b)$ and $\psi_i \in \phi_2^{\succ}(b)$, then φ_i and ψ_i must be located at the same position in $\phi_1^{\succ}(b)$ and $\phi_2^{\succ}(b)$, respectively, to achieve that $\phi_1^{\succ}(b) = \alpha \cdot \phi_2^{\succ}(b)$.

Finally, we remark that this approach can be further extended to allow for a small deviation between potential values that are considered identical. More specifically, instead of requiring that the equality $\varphi_i = \alpha \cdot \psi_i$ holds for all $i \in \{1, \ldots, n\}$, we can allow for a small deviation of factor $(1 + \varepsilon)$ between potential values and require that $\varphi_i \in [\alpha \cdot \psi_i \cdot (1 - \varepsilon), \alpha \cdot \psi_i \cdot (1 + \varepsilon)]$ holds for all $i \in \{1, \ldots, n\}$. To check whether φ_i lies in the specified interval, we need to ensure that the following inequalities hold:

$$\varphi_i \geq \alpha \cdot \psi_i \cdot (1 - \varepsilon), \text{ and} \tag{18}$$

$$\varphi_i \leq \alpha \cdot \psi_i \cdot (1 + \varepsilon). \tag{19}$$

By entering $\alpha = \varphi_1 \,/\, \psi_1$ into Eqs. (18) and (19), we obtain

$$\varphi_i \cdot \psi_1 \geq \varphi_1 \cdot \psi_i \cdot (1 - \varepsilon), \text{ and} \tag{20}$$

$$\varphi_i \cdot \psi_1 \leq \varphi_1 \cdot \psi_i \cdot (1 + \varepsilon). \tag{21}$$

Note that in general, it holds that $\varepsilon \in [0, 1]$ is a small floating point number. To avoid floating point arithmetics, we can restrict ε to be a rational number, which does not limit the practical applicability of this approach, as ε can still be chosen arbitrarily small (e.g., $1 \,/\, q$ for some arbitrary $q \in \mathbb{Z}$). In particular, if ε is a rational number, it can be represented by a fraction $\varepsilon = p \,/\, q$ where $p \in \mathbb{Z}$ and

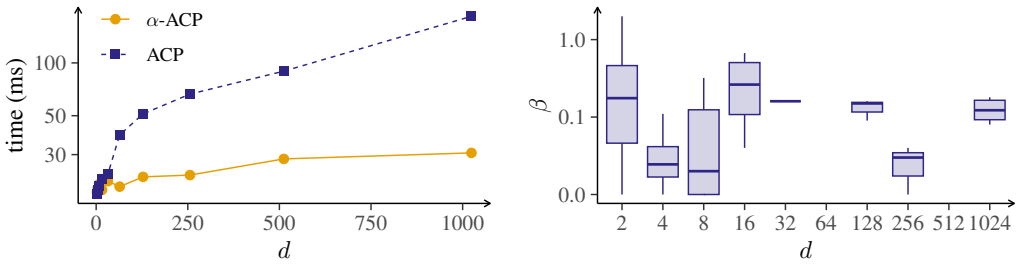


Figure 7: Average query times of lifted variable elimination run on the output of ACP and α-ACP where the input FGs contain a proportion of $p = 0.01$ scaled factors (left), and the distribution of the number β of queries after which the offline overhead of α-ACP amortises on input FGs containing a proportion of $p = 0.01$ scaled factors (right).

$q \in \mathbb{Z}$ are integers. Making use of this property, entering $\varepsilon = p \,/\, q$ into Eq. (20) yields

$$\varphi_i \cdot \psi_1 \geq \varphi_1 \cdot \psi_i \cdot (1 - \varepsilon) \tag{22}$$
$$\Leftrightarrow \varphi_i \cdot \psi_1 \geq \varphi_1 \cdot (\psi_i - \psi_i \cdot \varepsilon) \tag{23}$$
$$\Leftrightarrow \varphi_i \cdot \psi_1 \geq \varphi_1 \cdot \psi_i - \varphi_1 \cdot \psi_i \cdot \varepsilon \tag{24}$$
$$\Leftrightarrow \varphi_i \cdot \psi_1 \geq \varphi_1 \cdot \psi_i - \varphi_1 \cdot \psi_i \cdot \frac{p}{q} \tag{25}$$
$$\Leftrightarrow \varphi_i \cdot \psi_1 \cdot q \geq \varphi_1 \cdot \psi_i \cdot q - \varphi_1 \cdot \psi_i \cdot p \tag{26}$$

and analogously for Eq. (21), we get

$$\varphi_i \cdot \psi_1 \cdot q \leq \varphi_1 \cdot \psi_i \cdot q + \varphi_1 \cdot \psi_i \cdot p. \tag{27}$$

Again, these inequalities can be checked by using only multiplication operations, thereby allowing us to check for exchangeable factors independent of the scale of their potentials while at the same time allowing for arbitrary permutations of arguments and even a small deviation between potential values without using any floating point arithmetics (if the potential values themselves are integers).

E Additional Experimental Results

In addition to the experimental results provided in Sec. 4, we give further experimental results in this section. We again evaluate the run times of running lifted variable elimination on the output of ACP as well as of running lifted variable elimination on the output of α-ACP and also investigate the average number β of queries after which the additional offline overhead of α-ACP compared to ACP amortises. The instances used in this section are identical to those used in Sec. 4 but we do not average the results over the proportion $p \in \{0.01, 0.05, 0.1, 0.15\}$ of scaled factors. Instead, we present separate results for each individual choice of p to highlight the effect of p on both the run times for online query answering as well as on the offline overhead for constructing the PFG.

The results are illustrated in Figs. 7 to 10. Unsurprisingly, the run times of lifted variable elimination on the output of α-ACP remain constant for all choices of p because α-ACP is able to detect arbitrarily many scaled factors without forfeiting compression. At the same time, ACP is not able to detect exchangeable factors on different scales and thus, the run times of lifted variable elimination on the output of ACP increase as the proportion p of scaled factors increases. Regarding the amortisation of the offline overhead (depicted in the plots on the right), we can observe the same behaviour as in Fig. 5 (negative values for β are again omitted). The median value for β is always below one and there are no notable differences between the different choices of p. Even though the values of β slightly deviate between different choices for p, the deviation can be considered negligible and it seems as the deviation stems from noisy measurements. In conclusion, the additional offline overhead of α-ACP amortises after a single query most of the time, highlighting the efficiency of α-ACP.

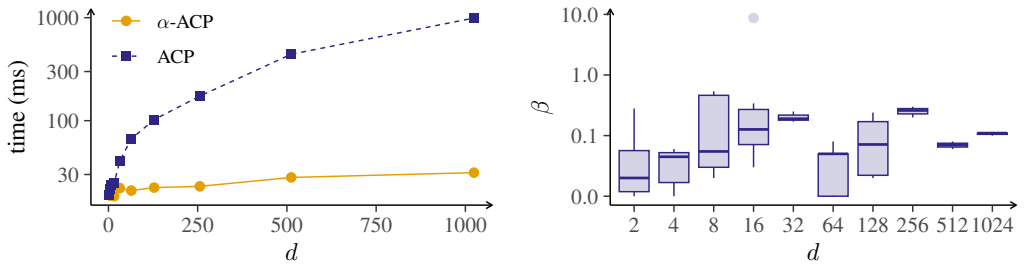


Figure 8: Average query times of lifted variable elimination run on the output of ACP and α-ACP where the input FGs contain a proportion of $p = 0.05$ scaled factors (left), and the distribution of the number β of queries after which the offline overhead of α-ACP amortises on input FGs containing a proportion of $p = 0.05$ scaled factors (right).

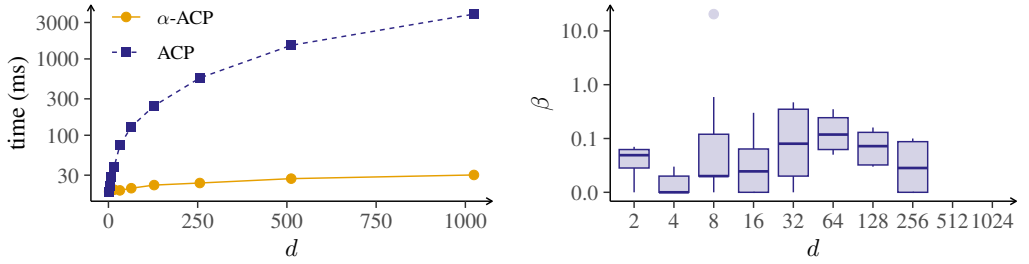


Figure 9: Average query times of lifted variable elimination run on the output of ACP and α-ACP where the input FGs contain a proportion of $p = 0.1$ scaled factors (left), and the distribution of the number β of queries after which the offline overhead of α-ACP amortises on input FGs containing a proportion of $p = 0.1$ scaled factors (right).

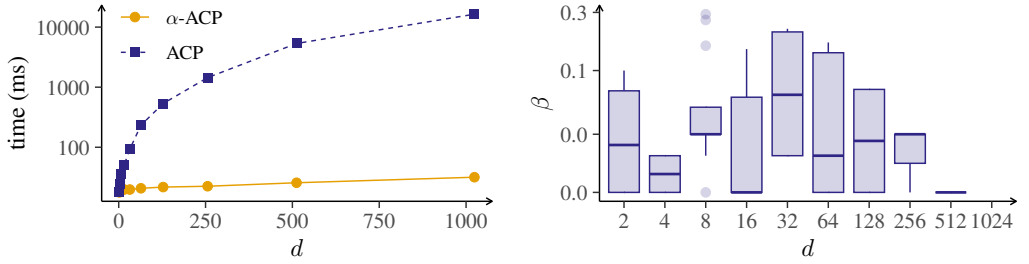


Figure 10: Average query times of lifted variable elimination run on the output of ACP and α-ACP where the input FGs contain a proportion of $p = 0.15$ scaled factors (left), and the distribution of the number β of queries after which the offline overhead of α-ACP amortises on input FGs containing a proportion of $p = 0.15$ scaled factors (right).