Colour Passing Revisited: Lifted Model Construction with Commutative Factors*

Malte Luttermann[1,2], **Tanya Braun**[3], **Ralf Möller**[1,2], **Marcel Gehrke**[2]

[1]German Research Center for Artificial Intelligence (DFKI), Lübeck, Germany
[2]Institute of Information Systems, University of Lübeck, Germany
[3]Data Science Group, University of Münster, Germany
malte.luttermann@dfki.de, tanya.braun@uni-muenster.de, {moeller, gehrke}@ifis.uni-luebeck.de

Abstract

Lifted probabilistic inference exploits symmetries in a probabilistic model to allow for tractable probabilistic inference with respect to domain sizes. To apply lifted inference, a lifted representation has to be obtained, and to do so, the so-called colour passing algorithm is the state of the art. The colour passing algorithm, however, is bound to a specific inference algorithm and we found that it ignores commutativity of factors while constructing a lifted representation. We contribute a modified version of the colour passing algorithm that uses logical variables to construct a lifted representation independent of a specific inference algorithm while at the same time exploiting commutativity of factors during an offline-step. Our proposed algorithm efficiently detects more symmetries than the state of the art and thereby drastically increases compression, yielding significantly faster online query times for probabilistic inference when the resulting model is applied.

1 Introduction

Parametric factor graphs (PFGs) combine probabilistic modeling with first-order logic by introducing logical variables (logvars), allowing for reasoning under uncertainty about individuals and their relationships. A fundamental task using PFGs is to perform probabilistic inference, i.e., to compute marginal distributions of random variables (randvars) given observations for other randvars. To allow for tractable probabilistic inference (e.g., inference requiring polynomial time) with respect to domain sizes of logvars, PFGs with corresponding inference algorithms have been developed, where the main idea is to use a representative for indistinguishable individuals for computations. To perform lifted inference, a lifted representation has to be constructed first. In this paper, we study the problem of lifted model construction—that is, we aim to obtain a lifted representation, equivalent to a given propositional (ground) model, which can then be used for lifted probabilistic inference. In particular, we consider the problem of lifted model construction independent of a specific inference algorithm and under consideration of commutative factors, i.e., factors that map to identical values regardless of the order of their arguments.

Over the last two decades, there has been a considerable amount of research dealing with lifted probabilistic inference using PFGs. Lifted inference exploits symmetries in a relational model to compute marginal distributions more efficiently while maintaining exact answers (Niepert and Van den Broeck 2014). Poole (2003) first introduces PFGs and lifted variable elimination (LVE) as an algorithm to perform lifted inference in PFGs. LVE has then been refined and developed further by many researchers to reach its current form (De Salvo Braz, Amir, and Roth 2005, 2006; Milch et al. 2008; Kisyński and Poole 2009; Taghipour et al. 2013a; Braun and Möller 2018). Other inference algorithms operating on PFGs include the lifted junction tree (LJT) algorithm, which is designed to handle sets of queries (Braun and Möller 2016). The well-known colour passing (CP) algorithm (originally named "CompressFactorGraph") introduced by Kersting, Ahmadi, and Natarajan (2009) builds on the work by Singla and Domingos (2008) and is commonly used to construct a lifted representation from a given factor graph (FG). The CP algorithm incorporates a colour passing procedure to detect symmetries in a graph similar to the Weisfeiler-Leman algorithm (Weisfeiler and Leman 1968), which is commonly used to test for graph isomorphism. Even though the CP algorithm is technically able to construct a PFG, CP in its current form is used as an intermediate step for lifted belief propagation and hence, CP is bound to this specific algorithm and does not introduce logvars to output a valid PFG. Furthermore, the CP algorithm does not handle commutative factors (i.e., factors that map to a unique output value regardless of the order of some input values) and is dependent on the order of the factors' argument lists (i.e., identical factors where the arguments of one factor are permuted are not recognised). Both impose significant limitations for practical applications as taking commutative factors into account and finding identical factors independent of the order of their argument lists result in more compressed models and thus allow for an additional speedup during inference. Other works establish a connection between automorphism groups and coloured graphs (Niepert 2012; Bui, Huynh, and Riedel 2013; Holtzen, Millstein, and Van den Broeck 2020) by searching for symmetries in a full joint probability distribution without exploiting a factorisation of the distribution. However, these works do not introduce logvars and hence, their lifted representation is dependent on a specific inference algorithm as well.

To overcome the limitations of neglecting commutative

*Extended version of paper accepted to the Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI-24).

factors and relying on fixed argument orders to detect identical factors in CP, we contribute the advanced colour passing (ACP) algorithm which is a modification of CP that also handles commutative factors and finds identical factors independent of argument orders, resulting in more compact models while maintaining equivalent model semantics. In addition, ACP is independent of a specific inference algorithm. ACP uses so-called counting randvars (CRVs) to compactly encode commutative factors and we transfer the idea of using histograms from CRVs to allow for order-independent identification of identical factors. More specifically, we (i) exploit symmetries *within* a factor, and (ii) make use of symmetries *between* factors, where potentials are identical although they have not been recognised as such before. An additional offline step allows us to tackle both (i) and (ii), which contribute to a more compact model to drastically reduce the time needed to perform online inference. CRVs are already used in lifted inference algorithms but are not yet used during learning a PFG and thus, using CRVs to encode a PFG vastly compresses the model. We also show how logvars are introduced to obtain a fully-fledged pipeline from propositional FG to PFG allowing for tractable probabilistic inference with respect to domain sizes in a PFG independent of a specific inference algorithm.

The remaining part of this paper is structured as follows. Section 2 introduces background information and notations. Afterwards, in Section 3, we present solutions to efficiently handle both symmetries within factors and symmetries between factors. Following these solutions, in Section 4, we introduce the ACP algorithm, which builds on the CP algorithm, to transform an input FG to a valid PFG under consideration of commutative factors and independent of factors' argument orders. We show how logvars are introduced by ACP to obtain the PFG as a lifted representation independent of a specific inference algorithm. In Section 5, we provide experiments confirming that ACP yields significantly faster inference times compared to the state of the art.

2 Background

We begin by recapitulating FGs as propositional probabilistic models and then move on to define PFGs. An FG is an undirected graphical model to represent a full joint probability distribution (Kschischang, Frey, and Loeliger 2001).

Definition 1 (Factor Graph)**.** *An* FG $G = (\boldsymbol{V}, \boldsymbol{E})$ *is a bipartite graph with* $\boldsymbol{V} = \boldsymbol{R} \cup \boldsymbol{F}$ *where* $\boldsymbol{R} = \{R_1, \dots, R_n\}$ *is a set of variable nodes and* $\boldsymbol{F} = \{f_1, \dots, f_m\}$ *is a set of factor nodes, and there is an edge between a variable node* R *and a factor node* f *in* $\boldsymbol{E} \subseteq \boldsymbol{R} \times \boldsymbol{F}$ *if* R *appears in the argument list of* f*. A factor is a function that maps its arguments to a positive real number (called potential). The semantics of an FG can be expressed by* $P(R_1, \dots, R_n) = \frac{1}{Z} \prod_{f \in \boldsymbol{F}} f$ *with* Z *being the normalisation constant.*

Figure 1a shows a toy example of an FG with five variable nodes $ComA$, $ComB$, Rev, $SalA$, and $SalB$ and five factor nodes $f_1, \dots, f_5$. The FG describes the relationships between a company's revenue (Rev) and its employee's competences and salaries: There are two employees $Alice$ (A) and Bob (B), their competences are denoted as $ComA$, $ComB$, respectively, and their salaries are given by $SalA$, $SalB$, respectively. The input-output pairs of the factors are omitted for brevity. We next define PFGs, first introduced by Poole (2003), based on the definitions given by Gehrke, Möller, and Braun (2020). PFGs combine first-order logic with probabilistic models, using logvars as parameters in randvars to represent sets of indistinguishable randvars, forming parameterised randvars (PRVs).

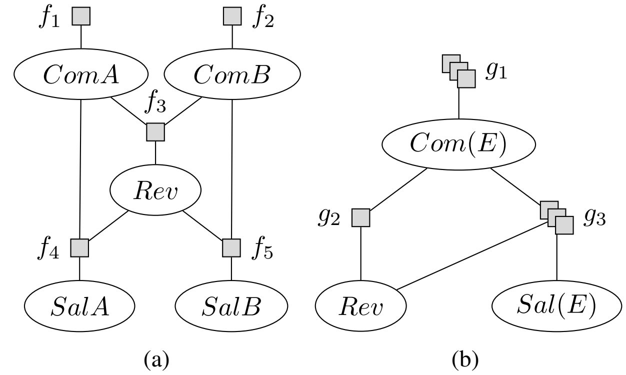


Figure 1: (a) An FG describing the relationships between a company's revenue and its employee's competences and salaries, (b) a PFG corresponding to the lifted representation of the FG shown in (a). The mappings of argument values to potentials of the (par)factors are omitted for brevity.

Definition 2 (Logvar, PRV, Event)**.** *Let* $\boldsymbol{R}$ *be a set of randvar names,* $\boldsymbol{L}$ *a set of logvar names,* Φ *a set of factor names, and* $\boldsymbol{D}$ *a set of constants. All sets are finite. Each logvar* L *has a domain* $\mathcal{D}(L) \subseteq \boldsymbol{D}$*. A* constraint *is a tuple* $(\mathcal{X}, C_{\mathcal{X}})$ *of a sequence of logvars* $\mathcal{X} = (X_1, \dots, X_n)$ *and a set* $C_{\mathcal{X}} \subseteq \times_{i=1}^{n} \mathcal{D}(X_i)$*. The symbol* $\top$ *for* C *marks that no restrictions apply, i.e.,* $C_{\mathcal{X}} = \times_{i=1}^{n} \mathcal{D}(X_i)$*. A* PRV $R(L_1, \dots, L_n)$, $n \geq 0$*, is a syntactical construct of a randvar* $R \in \boldsymbol{R}$ *possibly combined with logvars* $L_1, \dots, L_n \in \boldsymbol{L}$ *to represent a set of randvars. If* $n = 0$*, the PRV is parameterless and forms a propositional randvar. A PRV* A *(or logvar* L*) under constraint* C *is given by* $A_{|C}$ *(*$L_{|C}$*), respectively. We may omit* $|\top$ *in* $A_{|\top}$ *or* $L_{|\top}$*. The term* $\mathcal{R}(A)$ *denotes the possible values (range) of a PRV* A*. An* event $A = a$ *denotes the occurrence of PRV* A *with range value* $a \in \mathcal{R}(A)$ *and we call a set of events* $\boldsymbol{E} = \{A_1 = a_1, \dots, A_k = a_k\}$ evidence*.*

As an example, consider $\boldsymbol{R} = \{Com, Rev, Sal\}$ for competence, revenue, and salary, respectively, and $\boldsymbol{L} = \{E\}$ with $\mathcal{D}(E) = \{Alice, Bob\}$ (employees), combined into Boolean PRVs $Com(E)$, Rev, and $Sal(E)$.

A parametric factor (parfactor) describes a function, mapping argument values to positive real numbers (called potentials), of which at least one is non-zero.

Definition 3 (Parfactor, Model, Semantics)**.** *We denote a* parfactor g *by* $\phi(\mathcal{A})_{|C}$ *with* $\mathcal{A} = (A_1, \dots, A_n)$ *a sequence of PRVs,* $\phi\colon \times_{i=1}^{n} \mathcal{R}(A_i) \mapsto \mathbb{R}^+$ *a function with name* $\phi \in \Phi$ *mapping argument values to a positive real number called* potential*, and* C *a constraint on the logvars of* $\mathcal{A}$*. We may* *omit $|\top$ in $\phi(\mathcal{A})_{|\top}$. The term $lv(Y)$ refers to the logvars in some element Y, a PRV, a parfactor, or sets thereof. The term $gr(Y_{|C})$ denotes the set of all instances (groundings) of Y w.r.t. constraint C. A set of parfactors $\{g_i\}_{i=1}^{n}$ forms a* PFG *G. The semantics of G is given by grounding and building a full joint distribution. With Z as the normalisation constant, G represents $P_G = \frac{1}{Z}\prod_{f \in gr(G)} f$.*

Before we take a look at an example, we need the concept of a CRV, introduced by Milch et al. (2008).

Definition 4 (CRV). *$\#_X[R(\mathcal{X})]$ denotes a* CRV*, where $lv(\mathcal{X}) = \{X\}$ (other inputs constant). Its range is the space of possible histograms. A histogram h is a set of tuples $\{(v_i, n_i)\}_{i=1}^{m}$, $v_i \in \mathcal{R}(R(\mathcal{X}))$, $n_i \in \mathbb{N}$, $m = |\mathcal{R}(R(\mathcal{X}))|$, and $\sum_i n_i = |gr(X_{|C})|$ for some constraint C over $\mathcal{X}$. A shorthand notation is $[n_1, \dots, n_m]$. $h(v_i)$ returns n_i. Since counting binds logvar X, $lv(\#_X[R(\mathcal{X})]) = \mathcal{X} \setminus \{X\}$.*

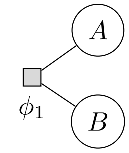
For example, the two groundings of a Boolean PRV $R(X)$ with $|\mathcal{D}(X)| = 2$ can be assigned true values ($[2,0]$), one true and one false value ($[1,1]$), or false values ($[0,2]$). Figure 1b shows a PFG $G = \{g_i\}_{i=1}^{3}$ with $g_1 = \phi_1(Com(E))_{|\top}$, $g_2 = \phi_2(\#_E[Com(E)], Rev)_{|\top}$, and $g_3 = \phi_3(Com(E), Rev, Sal(E))_{|\top}$ where ϕ_2 contains a CRV $\#_E[Com(E)]$. G is a lifted representation of the FG shown in Fig. 1a. The definition of a PFG also implies that every FG is a PFG containing only parameterless randvars.

The state of the art algorithm to transform a (propositional) FG into a lifted representation is the CP algorithm (Kersting, Ahmadi, and Natarajan 2009; Ahmadi et al. 2013), which we briefly recap in the following. CP tries to find symmetries in an FG based on potentials of factors, on ranges and evidence of randvars, as well as on the graph structure. Each randvar is assigned a colour such that randvars with identical ranges and identical evidence get the same colour, and each factor is assigned a colour such that factors with the same potentials get the same colour. The colours are then passed from every randvar to its neighbouring factors and vice versa. After each colour passing step, colours are reassigned depending on the received colours and a node's own colour. Factor nodes also include the position of a randvar in their argument list into their message. In the end, all randvars and factors, respectively, are grouped together based on their colours and the procedure is iterated until groupings do not change anymore. A formal description of CP can be found in Appendix A. In its current form, CP does not handle commutative factors and is dependent on the order of the factors' argument lists. Therefore, we next explore the problem of lifting an FG to obtain a PFG taking into account commutative factors and afterwards tackle the problem of finding identical factors in an FG independent of the order of their argument lists.

3 Colour Passing Revisited

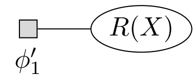
Consider again the example shown in Fig. 1. Intuitively, one might expect the CP algorithm to output the groupings corresponding to the PFG shown in Fig. 1b if f_1 and f_2 as well as f_4 and f_5 share the same potentials. The CP algorithm, however, ends up without grouping anything if it is provided with the FG from Fig. 1a as input because f_3 sends different messages to $ComA$ and $ComB$ due to different positions of $ComA$ and $ComB$ in f_3's argument list.

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



(a)

$\#_X[R(X)]$	$\phi'_1(\#_X[R(X)])$
$[2,0]$	φ_1
$[1,1]$	φ_2
$[0,2]$	φ_3



(b)

Figure 2: (a) An FG containing a commutative factor ϕ_1, (b) a PFG entailing equivalent semantics as the FG shown in (a). To obtain the PFG in (b), ϕ_1 is mapped to ϕ'_1 using a CRV $\#_X[R(X)]$ counting over a logvar X with $|\mathcal{D}(X)| = 2$.

In the upcoming subsection, we show that in cases where a factor f is commutative, the position of a randvar in f's argument list is not relevant for the colour passing procedure and hence can be omitted. Afterwards, we transfer the idea of using histograms from CRVs to efficiently detect identical factors independent of the order of their arguments. We demonstrate that instead of scanning two tables from top to bottom and comparing their values one by one, we can build a set of potential values for each possible histogram and compare the sets pairwise to guarantee order-independence without introducing additional computational overhead.

3.1 Symmetries within Factors

A simplified version of the situation regarding f_3 in Fig. 1a is depicted in Fig. 2a. Here, again, CP does not group anything because A and B have different positions in ϕ_1. However, in this example, ϕ_1 encodes a symmetric function (that is, a function returning the same value independent of the order of its arguments) because $\phi_1(\mathrm{true}, \mathrm{false}) = \phi_1(\mathrm{false}, \mathrm{true}) = \varphi_2$ and we could use the symmetries within ϕ_1 to group A and B using a CRV, as shown in Fig. 2b. Although lifted inference algorithms such as LVE use CRVs by count converting PRVs during the inference task (Taghipour et al. 2013a), CRVs have, to the best of our knowledge, not been used to learn a valid PFG so far. In consequence, we gain a significant speedup for inference algorithms when using CRVs to model a PFG.

We now explain how to make use of the symmetries within ϕ_1 in Fig. 2a to group together A and B. Even though we choose Boolean randvars to keep the example small, the idea also applies to randvars with arbitrary ranges. For two Boolean randvars (here A and B), there are three possible histograms $[2,0]$, $[1,1]$, and $[0,2]$. As ϕ_1 outputs a unique value for all of the three possible histograms, ϕ_1 is commutative and thus, ϕ_1 can be represented by a factor ϕ'_1 taking as input a CRV that counts over a logvar X with

$|\mathcal{D}(X)| = 2$. Figure 2b visualises the resulting PFG using a CRV to obtain a lifted representation equivalent to the FG shown in Fig. 2a, where the size of the table has been reduced from exponential to polynomial in the number of arguments of the factor. ϕ'_1 now maps each possible histogram to a potential, i.e., ϕ'_1 outputs φ_1 for the histogram $[2, 0]$, φ_2 for $[1, 1]$, and φ_3 for $[0, 2]$. These mappings of the histograms capture exactly the semantics of ϕ_1 from Fig. 2a. Consequently, we use CRVs to compactly encode symmetric functions and thus allow for additional groupings of randvars and factors. We incorporate this idea into the ACP algorithm by omitting the position of a randvar in a factor's argument list in a message if the factor is commutative. To profit from the usage of CRVs, it is even sufficient for a factor to be *partially commutative*, defined as follows.

Definition 5 (Partially Commutative Factor)**.** *A factor ϕ with argument list $R_1, \dots, R_n$ is called* partially commutative *if there exists a non-empty subset $\boldsymbol{S} \subseteq \{R_1, \dots, R_n\}$ with $|\boldsymbol{S}| > 1$ such that ϕ is commutative with respect to the subset $\boldsymbol{S}$, i.e., for all events $r_1, \dots, r_n \in \times_{i=1}^{n} \mathcal{R}(R_i)$ it holds that $\phi(r_1, \dots, r_n) = \phi(r_{\pi(1)}, \dots, r_{\pi(n)})$ for all permutations π of $\{1, \dots, n\}$ with $\pi(i) = i$ for all $r_i \notin \boldsymbol{S}$.*

In many practical settings, factors are partially commutative, for example when individuals are indistinguishable and only the number of individuals having a certain property is of interest (e.g., in the employee example the number of competent employees determines the revenue of the company while it does not matter which specific employees are competent). To check whether a factor $\phi(R_1, \dots, R_n)$ is commutative, let w.l.o.g. $\{R_1, \dots, R_m\} \subseteq \{R_1, \dots, R_n\}$ be a maximal subset of randvars with $\mathcal{R}(R_1) = \dots = \mathcal{R}(R_m)$ that satisfies the condition given in Definition 5. If no such subset exists, ϕ is not commutative, else ϕ can be mapped to a new (par)factor ϕ' by replacing the randvars $R_1, \dots, R_m$ by a CRV counting over a logvar X with $|\mathcal{D}(X)| = m$. The remaining randvars occurring in the argument list of ϕ but not in $R_1, \dots, R_m$ are transferred unchanged to the argument list of ϕ'. Each combination of histogram and possible values for the remaining randvars is then mapped to a unique value by ϕ'.

Example 1. *Consider again the factor f_3 from Fig. 1a, which takes the three arguments $ComA$, $ComB$, and Rev as input. Since the structure of Rev differs from the structure of $ComA$ and $ComB$, we do not intend to count over the complete set of input randvars but only over a subset of the set of input randvars. The subset should be as large as possible to obtain the most compression for our lifted representation. Moreover, to group $ComA$ and $ComB$ into $Com(E)$, they have to behave identically with respect to the potentials of f_3, that is, counting over them has to result in identical potentials for each histogram. In particular, assuming that the order of f_3's arguments is $ComA$, $ComB$, Rev, it must hold that* $f_3(\mathrm{true}, \mathrm{false}, \mathrm{true}) = f_3(\mathrm{false}, \mathrm{true}, \mathrm{true})$ *and* $f_3(\mathrm{true}, \mathrm{false}, \mathrm{false}) = f_3(\mathrm{false}, \mathrm{true}, \mathrm{false})$.

For a factor ϕ with n arguments $R_1, \dots, R_n$, the number of possible subsets we could possibly count over is in $O(2^n)$. The number of candidates can be reduced by only considering subsets $\{R_1, \dots, R_m\} \subseteq \{R_1, \dots, R_n\}$ where $R_1, \dots, R_m$ have the same number of neighbours in the graph because randvars with different numbers of neighbouring factors receive different messages during colour passing and hence cannot be grouped together. Even though the number of subsets to check in the worst case remains in $O(2^n)$, the computational effort is often reduced by the optimisation of only considering subsets consisting of randvars which have the same number of neighbours. Moreover, as we aim to compress an FG by exploiting symmetries, we inherently assume that there are at least some symmetries in the FG (otherwise, we would not intend to run CP). Thus, symmetries within factors are likely to be found fast in practical applications, which we also confirm in our experiments.

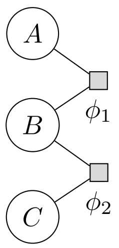
In the worst case, checking $O(2^n)$ subsets for commutativity is infeasible for large n but we argue that for practical applications, we can assume that n is reasonably small: A factor $\phi(R_1, \dots, R_n)$ defines 2^n mappings (rows in its table) if all randvars $R_1, \dots, R_n$ are Boolean and hence, storing the table of input-output pairs requires $O(2^n)$ space (for larger ranges of the R_i, there are even more mappings). Consequently, the table cannot even be stored for large n, implying that the number of arguments of each factor is limited to small values in practical applications. As there are at least as many rows as there are subsets of $\{R_1, \dots, R_n\}$, the current version of CP needs exponential time in n even without checking for commutativity of factors. Therefore, handling symmetries within factors requires no additional costs while at the same time allowing us to drastically increase compression and thus speed up online inference. The number of rows is reduced from exponential in n to polynomial in n (Milch et al. 2008) and randvars as well as factors are grouped together that could not be grouped together before.

So far, we applied the idea of using histograms to find symmetries *within* a factor. Next, we aim to find symmetries *between* factors independent of the order of their arguments before gathering both ideas into the ACP algorithm. When checking for symmetries *between* multiple factors, the same histograms that are used to check for symmetries *within* factors are computed, allowing us to reuse these histograms without additional computational effort.

3.2 Symmetries between Factors

We now deploy histograms to detect symmetries between different factors. To illustrate this point, have a look at Fig. 3. Considering the factor ϕ_2 in Fig. 3, C has position two and B position one in the argument list of ϕ_2. Swapping the positions of C and B in ϕ_2 results in a table where the positions of φ_2 and φ_3 are swapped because $\phi_2(C = \mathrm{true}, B = \mathrm{false}) = \varphi_2$ and $\phi_2(C = \mathrm{false}, B = \mathrm{true}) = \varphi_3$. In summary, ϕ_2 still entails the same semantics after rearranging its arguments if its mappings (rows in the table of ϕ_2) are swapped accordingly at the same time.

Thus, the potentials of ϕ_1 and ϕ_2 are actually identical. Running CP in its current form on the FG from Fig. 3, however, results in no groups at all for two reasons. First, depending on the way CP checks for identical potentials, it might assign different colours to the factors ϕ_1 and ϕ_2 because comparing their tables row by row would lead to dif-



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 3: An exemplary FG where the input order of the arguments of ϕ_2 (or ϕ_1) can be rearranged such that the potentials of ϕ_1 and ϕ_2 are identical when comparing their tables.

ferent colours for ϕ_1 and ϕ_2[1]. Second, A and C are located at different positions in their respective factor and hence receive different messages during message passing. However, requiring indistinguishable individuals to appear at the same position in each factor is a massive restriction for practical applications. On the other hand, naively comparing all $O(n!)$ permutations of an argument list of length n requires a lot of computational effort. Using histograms to ensure necessary conditions when searching for identical potentials avoids having to try all permutations in advance.

Theorem 1 (Necessary Conditions for Identical Potentials)**.** *For two factors $\phi_1(R_1, \ldots, R_n)$ and $\phi_2(R'_1, \ldots, R'_n)$ to be able to represent equivalent potentials, the following two conditions are required to hold:*

1. *A bijection τ: $\{R_1, \ldots, R_n\} \to \{R'_1, \ldots, R'_n\}$ exists that maps each R_i to a R'_j such that $\mathcal{R}(R_i) = \mathcal{R}(R'_j)$,*
2. *for each histogram $\{(v_i, n_i)\}_{i=1}^m$ with $v_i \in \bigcup_{i=1}^n \mathcal{R}(R_i)$, $n_i \in \mathbb{N}$, $m = |\bigcup_{i=1}^n \mathcal{R}(R_i)|$, and $\sum_i n_i = n$, the multiset of all potentials to which the histogram is mapped is identical for ϕ_1 and ϕ_2.*

Proof. If Item 1 is not satisfied, there must be at least one pair of arguments R_i and R'_j such that $\mathcal{R}(R_i) \neq \mathcal{R}(R'_j)$, implying that ϕ_1 and ϕ_2 are defined over different function domains and hence cannot have identical potentials. Regarding Item 2, we know that the set of possible histograms specifies all possible inputs for a factor if we neglect the order of its arguments. Consequently, if there exists a histogram that is mapped to different multisets of values by ϕ_1 and ϕ_2, there is no possibility to permute the arguments such that both tables read identical values from top to bottom. □

In other words, Item 1 ensures that the function domains of the two factors are identical when neglecting the order of their arguments, which implies that both factors entail the same set of possible histograms (τ does not have to be unique). Item 2 demands that the mapping from order-independent inputs to outputs is equivalent for both factors—otherwise, there exists no permutation of the arguments such that both factors have identical potentials.

Example 2. *Applying Theorem 1 to our example from Fig. 3, there are three different possible histograms, which are identical for ϕ_1 and ϕ_2 as their arguments satisfy Item 1. We can also verify that all three histograms yield identical multisets of mapped values for ϕ_1 and ϕ_2: $[2, 0] \mapsto \{\varphi_1\}$, $[1, 1] \mapsto \{\varphi_2, \varphi_3\}$, and $[0, 2] \mapsto \{\varphi_4\}$.*

Note that we use a multiset instead of a set for each histogram because it is possible for a factor to map a histogram to the same value multiple times. For example, in a scenario with two factors ϕ'_1 and ϕ'_2 where ϕ'_1 maps some histogram $[i, j] \mapsto \{\varphi'_1, \varphi'_1, \varphi'_2\}$ and ϕ'_2 maps $[i, j] \mapsto \{\varphi'_1, \varphi'_2, \varphi'_2\}$, the factors cannot represent identical potentials and a set with unique elements $\{\varphi'_1, \varphi'_2\}$ is not able to detect such a situation. The histograms allow us to avoid a naive check of all possible permutations, reducing the required computational effort as we only need to take a look at permutations if the initial histogram-check is passed successfully.

Corollary 1. *If two factors ϕ_1 and ϕ_2 do not satisfy Theorem 1, they cannot represent equivalent potentials.*

In case two factors satisfy Theorem 1, a sufficient condition for them to represent identical potentials is that there exists a permutation of the arguments of one factor such that their tables read identical values from top to bottom.

Corollary 2 (Sufficient Condition for Identical Potentials)**.** *Two factors $\phi_1(R_1, \ldots, R_n)$ and $\phi_2(R'_1, \ldots, R'_n)$ 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 $\phi_1(r_1, \ldots, r_n) = \phi_2(r_{\pi(1)}, \ldots, r_{\pi(n)})$.*

After rearranging the order of arguments and the order of potentials of a factor accordingly to match the potentials of another factor, the CP algorithm can be applied without further changes. Running CP on the FG shown in Fig. 3 after rearrangement, both A and C as well as ϕ_1 and ϕ_2 are grouped together. Using histograms to filter candidates before comparing permutations keeps the computational effort as small as possible and enables us to reuse the histograms for the commutativity check *within* a factor over the whole argument list. Thus, performing an initial histogram-check does not require additional computational costs and avoids naively comparing all permutations for order independence. Next, we gather the presented solutions to handle symmetries within and between factors into the ACP algorithm and show how logvars are introduced to obtain a valid PFG.

4 Advanced Colour Passing

In this section, we combine the insights from the previous section into a modified version of the CP algorithm, called ACP. Algorithm 1 presents the entire ACP algorithm, which is explained in more detail in the following.

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

[1]The original CP paper does not exactly specify how the check for identical potentials should be implemented.

Algorithm 1 Advanced Colour Passing

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.

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**
      $\triangleright$ 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

factors represent identical potentials if they satisfy the conditions given in Theorem 1 and 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. As shown in the previous section, ACP uses histograms to detect factors with identical potentials regardless of the order of their arguments. In case the arguments of a factor have to be rearranged to obtain identical tables during the comparison, ACP uses the positions of the arguments after the rearrangement throughout the message passing procedure afterwards. ACP rearranges each factor's arguments at most once. The message passing in ACP differs from the message passing in CP in the sense that every factor $\phi(R_1, \ldots, R_j)$ that is commutative with respect to a subset of its arguments $\boldsymbol{S} \subseteq \{R_1, \ldots, R_j\}$ passes the position $p(R, \phi)$ of a randvar R in ϕ's argument list only to randvars $R \notin \boldsymbol{S}$. Every factor ϕ passes zero instead of the actual position $p(R, \phi)$ to all randvars $R \in \boldsymbol{S}$ to mark commutativity. $\boldsymbol{S}$ is a maximal subset for which $|\boldsymbol{S}| > 1$ must hold as a single argument is always commutative with itself. All randvars receiving the position zero in their message are commutative and thus, ACP groups them using a CRV, as we have seen in Section 3.1. In case there are multiple maximal subsets $\boldsymbol{S}$, ACP chooses any of them. ACP iterates the message passing until convergence. In the end, ACP transforms all groups of randvars and factors into PRVs with logvars and parfactors, respectively, to obtain a valid PFG G' entailing equivalent semantics as the initial FG G. The construction of G' from the obtained groupings is explained in detail below.

We give the mapping from groups to PRVs and parfactors for the domain-liftable fragment (Van den Broeck 2011), i.e., for PFGs containing only parfactors in which at most two logvars appear as well as for PFGs containing only PRVs having at most one logvar, because lifted inference algorithms such as LVE and LJT are proven to be complete for this fragment (Taghipour et al. 2013b; Braun 2020).

Each group of factors $\boldsymbol{F}$ is replaced by a parfactor ϕ' and each group of randvars $\boldsymbol{A}$ is replaced by a PRV R' such that $gr(\phi') = \boldsymbol{F}$ and $gr(R') = \boldsymbol{A}$. The PFG G' then contains an edge between a PRV R' and a parfactor ϕ' (i.e., R' appears in the argument list of ϕ') if there is a randvar $R \in gr(R')$ which is connected to a factor $\phi \in gr(\phi')$ in the initial FG G. For each PRV R', the logvars are introduced depending on the groundings $gr(R')$. For the introduction of logvars, we only need to consider PRVs that are not parameterless, i.e., PRVs that represent a group consisting of at least two randvars. The exact conditions used by ACP for introducing logvars are given in the following definition.

Definition 6 (Introduction of Logvars in Randvar Groups)**.** *Let $\phi'(R'_1(X_{1,1}, \ldots, X_{1,k}), \ldots, R'_j(X_{j,1}, \ldots, X_{j,k}))$ be a new parfactor, build from $\boldsymbol{F} = \{\phi_1(R_{1,1}, \ldots, R_{1,s}), \ldots, \phi_\ell(R_{\ell,1}, \ldots, R_{\ell,s})\}$ and let $\boldsymbol{S} = \{S'_1, \ldots, S'_z\}$ denote the subset of ϕ''s arguments with more than one grounding. Then, ACP introduces the logvars of $S'_1, \ldots, S'_z$ as follows.*

1. *If $\forall S'_i \in \boldsymbol{S}$: $|\boldsymbol{F}| = |gr(S'_i)|$, all $S'_i \in \boldsymbol{S}$ have exactly one logvar which is identical for all $S'_i \in \boldsymbol{S}$.*
2. *If $\forall S'_i \in \boldsymbol{S}$: $|\boldsymbol{F}| \neq |gr(S'_i)|$, $S'_1, \ldots, S'_z$ have exactly one logvar. $S'_a \in \boldsymbol{S}$ and $S'_b \in \boldsymbol{S}$ share the same logvar if and only if $|gr(S'_a)| = |gr(S'_b)|$ and there exists a bijection τ: $gr(S'_a) \to gr(S'_b)$ such that τ maps every $S_a \in gr(S'_a)$ to $S_b \in gr(S'_b)$ with $\mathcal{F}(S_a) = \mathcal{F}(S_b)$ where $\mathcal{F}(S) = \{\phi(R_1, \ldots, R_s) \in \boldsymbol{F} \mid S \in \{R_1, \ldots, R_s\}\}$.*
3. *If $\exists S'_u \in \boldsymbol{S}$: $|\boldsymbol{F}| \neq |gr(S'_u)| \wedge \exists S'_v \in \boldsymbol{S}$: $|\boldsymbol{F}| = |gr(S'_v)|$, all $S'_i \in \boldsymbol{S}$ with $|\boldsymbol{F}| = |gr(S'_i)|$ have two logvars. The remaining $S'_i \in \boldsymbol{S}$ have exactly one logvar and share the same logvar under the same conditions as in Item 2.*

For each PRV $S'_i \in \boldsymbol{S}$ with a single logvar X, choose $\mathcal{D}(X)$ such that $|gr(S'_i)| = |\mathcal{D}(X)|$. For each PRV $S'_i \in \boldsymbol{S}$ with two logvars X_1 and X_2, choose $\mathcal{D}(X_1)$ and $\mathcal{D}(X_2)$ such that $|gr(S'_i)| = |\mathcal{D}(X_1)| \cdot |\mathcal{D}(X_2)|$.

The intuition behind Definition 6 is that after building ϕ' from $\boldsymbol{F}$, it must hold that $gr(\phi') = \boldsymbol{F}$ to ensure that G' entails equivalent semantics as G. Introducing logvars according to Definition 6 ensures that grounding G' results in a model equivalent to G for the domain-liftable fragment.

Theorem 2. *ACP returns a valid PFG entailing equivalent semantics as the initial FG for the domain-liftable fragment.*

We give a proof for Theorem 2 in Appendix B. Theorem 2 gives us the theoretical guarantees that ACP intro-

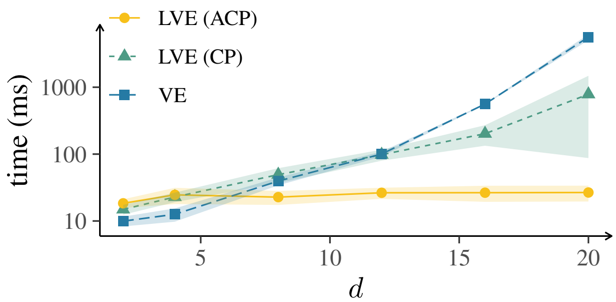


Figure 4: Average query times and their standard deviation of LVE on the output of CP, LVE on the output of ACP, and variable elimination (VE) on the initial (propositional) FG for input FGs containing a single commutative factor.

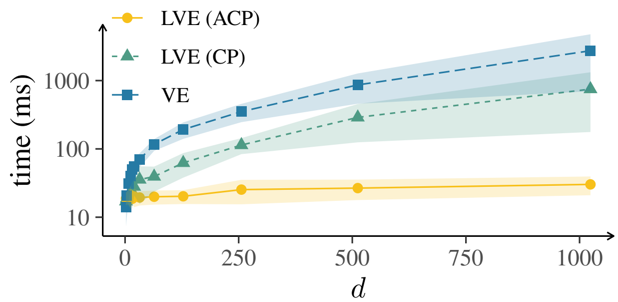


Figure 5: Average query times and their standard deviation of LVE on the output of CP, LVE on the output of ACP, and VE on the initial (propositional) FG for input FGs where about three percent of the factors have permuted arguments.

duces logvars correctly to obtain a valid PFG. In the next section, we demonstrate the effectiveness ACP in practice.

5 Experiments

In addition to the theoretical results, we show that ACP is able to drastically reduce online query times in practice. We evaluate the impact of symmetries within factors and between factors on query times separately. Figures 4 and 5 display the experimental results. In both plots, we report average query times of LVE on the resulting PFG after running ACP, denoted as LVE (ACP), of LVE on the resulting PFG after running CP[2], denoted as LVE (CP), and of variable elimination on the initial FG (VE). The average query times are given by the lines and the ribbon around the lines indicates the standard deviation. In both plots, the y-axis uses a logarithmic scale. We provide the data set generators along with our source code in the supplementary material.

The data set used in Fig. 4 consists of FGs containing between five and 102 factors of which exactly one factor is commutative. More specifically, for each domain size $d = 2, 4, 8, 12, 16, 20$, there are between $2d + 1$ and $d \cdot \lfloor \log_2(d) \rfloor + 2d + 2$ Boolean randvars and between $2d + 1$ and $d \cdot \lfloor \log_2(d) \rfloor + d + 2$ factors in the FGs. All factors representing identical potentials have exactly the same tables when comparing them row by row, that is, there are no symmetries between factors which cannot be detected by CP and hence, this is the ideal scenario for CP (although unrealistic in practice). The maximum number of arguments of a factor is $d + 1$, i.e., there are at most 2^{d+1} input-output pairs for a factor. For each choice of d, we evaluate multiple FGs by posing two queries per FG and then report the average run time and standard deviation over all queries for that choice of d. Figure 4 demonstrates that ACP yields a significant speedup (up to factor 30) compared to CP even though there is just a single commutative factor in the input FG and the potentials of the factors are specified in an optimal way for CP. The results indicate that CP imposes significant scalability issues for FGs containing commutative factors, even for rather small domain sizes. Thus, ACP is a major step to get a grip on scalability issues. Unsurprisingly, VE is the slowest of all algorithms. Appendix C provides additional results for FGs containing more than one commutative factor.

Figure 5 contains the results for FGs where the arguments of about three percent of the factors (randomly chosen) are permuted and there are no commutative factors. For each domain size $d = 2, 4, 8, 12, 16, 20, 32, 64, 128, 256, 512, 1024$, there are between $5d + 1$ and $d \cdot \lfloor \log_2(d) \rfloor + 2d + 1$ Boolean randvars and between $2d$ and $d \cdot \lfloor \log_2(d) \rfloor + d + 1$ factors in the input FGs. The maximum number of arguments of a factor is seven, i.e., the largest table contains 2^7 rows and thus, we are able to evaluate larger values for d as well (in the previous scenario, the number of rows increased exponentially with d). Again, we evaluate multiple FGs for each choice of d by posing two queries per FG and then report the average run time and standard deviation over all queries for that choice of d. The results depicted in Fig. 5 show that ACP can easily handle large domains, indicating that ACP is expected to handle FGs with tens of thousands of nodes without a hassle. In particular, ACP is able to achieve speedups of up to factor 25 compared to the state of the art CP, which, again, faces serious scalability issues. We give additional results for higher percentages of permuted factors in Appendix C and also provide an evaluation investigating when the additional offline overhead of ACP amortises there.

6 Conclusion

We introduce the ACP algorithm providing a fully-fledged pipeline from propositional FG to a valid PFG independent of a specific inference algorithm. ACP builds on the well-known CP algorithm, which does not handle commutative factors (i.e., factors with inherent symmetries resulting in mapping input arguments to unique values regardless of the order of those arguments) and is dependent on the order of the factors' argument lists. By using CRVs and histograms, ACP is able to efficiently encode commutative factors and handle factors representing identical potentials independent of the order of their argument lists. ACP not only provides significant speedups for online query answering but also solves serious scalability issues of CP in practice.

A fundamental problem for future research is to learn a PFG directly from a relational database without having to construct the ground model first. In this regard, CRVs provide a crucial component to keep the size of the PFG small.

[2]CP itself does not construct a valid PFG, so we additionally applied the steps from Definition 6 on the result obtained from CP.

Acknowledgements

This work is supported by the BMBF project AnoMed 16KISA057 and 16KISA050K. The authors also thank the anonymous reviewers for their valuable feedback.

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. 2020. *Rescued from a Sea of Queries: Exact Inference in Probabilistic Relational Models*. Ph.D. thesis, University of Lübeck.

Braun, T.; and Möller, R. 2016. Lifted Junction Tree Algorithm. In *Proceedings of KI 2016: Advances in Artificial Intelligence (KI-16)*, 30–42. Springer.

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.

Bui, H. H.; Huynh, T. N.; and Riedel, S. 2013. Automorphism Groups of Graphical Models and Lifted Variational Inference. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence (UAI-13)*, 132–141. AUAI Press.

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.

Gehrke, M.; Möller, R.; and Braun, T. 2020. Taming Reasoning in Temporal Probabilistic Relational Models. In *Proceedings of the Twenty-Fourth European Conference on Artificial Intelligence (ECAI-20)*, 2592–2599. IOS Press.

Holtzen, S.; Millstein, T.; and Van den Broeck, G. 2020. Generating and Sampling Orbits for Lifted Probabilistic Inference. In *Proceedings of The Thirty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI-20)*, 985–994. PMLR.

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.

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. 2012. Markov Chains on Orbits of Permutation Groups. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence (UAI-12)*, 624–633. AUAI 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.

Singla, P.; and Domingos, P. 2008. Lifted First-Order Belief Propagation. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-08)*, 1094–1099. AAAI Press.

Taghipour, N. 2013. *Lifted Probabilistic Inference by Variable Elimination*. Ph.D. thesis, KU Leuven.

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

Taghipour, N.; Fierens, D.; Van den Broeck, G.; Davis, J.; and Blockeel, H. 2013b. Completeness Results for Lifted Variable Elimination. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics (AISTATS-13)*, 572–580. PMLR.

Van den Broeck, G. 2011. On the Completeness of First-Order Knowledge Compilation for Lifted Probabilistic Inference. In *Advances in Neural Information Processing Systems 24 (NIPS-11)*, 1386–1394. Curran Associates, Inc.

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

Appendix

A Formal Description of the Colour Passing Algorithm

The colour passing (CP) algorithm (originally named "CompressFactorGraph") (Kersting, Ahmadi, and Natarajan 2009; Ahmadi et al. 2013) solves the problem of constructing a lifted representation for an input factor graph (FG). The idea of CP is to first find symmetries in a propositional FG and then group together symmetric subgraphs. CP 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 gives a formal description of the CP algorithm, which proceeds as follows. As an initialisation step, each variable node (randvar) is assigned a colour depending on its range and evidence, meaning that randvars with identical ranges and identical evidence are assigned the same colour, and each factor is assigned a colour depending on its potentials, i.e., factors with the same potentials get the same colour. Afterwards, the colour passing procedure begins. The colours are first passed from every variable node to its neighbouring factor nodes and each factor ϕ collects all colours of neighbouring randvars in the order of their appearance in the argument list of ϕ. Based on the collected colours and their own colour, factors are grouped together and reassigned a new colour (to reduce communication overhead). Then, colours are passed from factor nodes to variable nodes and each message from a factor ϕ to a randvar R includes the position $p(R, \phi)$ of R in ϕ in the message. Again, based on the collected colours and their own colour, randvars are grouped together and reassigned a new colour. The colour passing procedure is iterated until groupings do not change anymore and in the end, all randvars and factors, respectively, are grouped together based on their colour signatures (that is, the messages they received from their neighbours plus their own colour).

Algorithm 2 Colour Passing

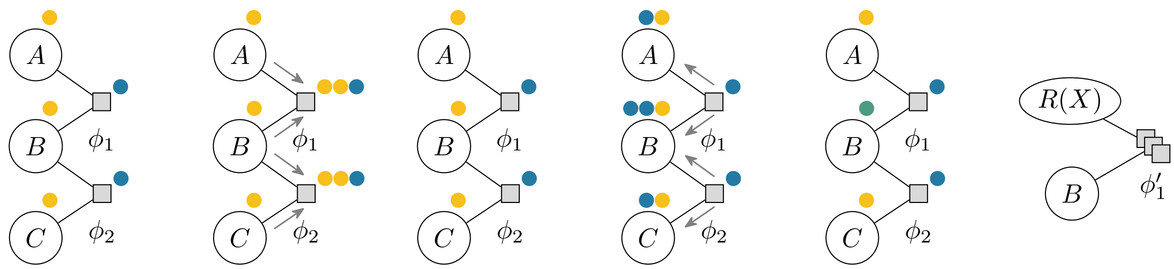
Input: An FG G with randvars $\boldsymbol{R} = \{R_1, \dots, R_n\}$, factors $\boldsymbol{\Phi} = \{\phi_1, \dots, \phi_m\}$, and evidence $\boldsymbol{E} = \{R_1 = r_1, \dots, R_k = r_k\}$.
Output: Groups of randvars and factors, respectively.

```
Assign each R_i a colour according to ℛ(R_i) and E
Assign each φ_i a colour according to its potentials
repeat
    for each factor φ ∈ Φ do
        signature_φ ← []
        for each randvar R ∈ neighbours(G, φ) do
                                         ▷ In order of appearance in φ
            append(signature_φ, R.colour)
        append(signature_φ, φ.colour)
    Group together all φs with the same signature
    Assign each such cluster a unique colour
    Set φ.colour correspondingly for all φs
    for each randvar R ∈ R do
        signature_R ← []
        for each factor φ ∈ neighbours(G, R) do
            append(signature_R, (φ.colour, p(R, φ)))
        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
```

Figure 6 illustrates the CP algorithm on an example FG (Ahmadi et al. 2013). In this example, A, B, and C are Boolean randvars with no evidence and thus, they all receive the same colour (e.g., yellow). As the potentials of ϕ_1 and ϕ_2 are identical, ϕ_1 and ϕ_2 are assigned the same colour as well (e.g., blue). The colour passing then starts from variable nodes to factor nodes, that is, A and B send their colour (yellow) to ϕ_1 and B and C send their colour (yellow) to ϕ_2. ϕ_1 and ϕ_2 are then recoloured according to the colours they received from their neighbours to reduce the communication overhead. Since ϕ_1 and ϕ_2 received identical colours (two times the colour yellow), they are assigned the same colour during recolouring. Afterwards, the colours are passed from factor nodes to variable nodes and this time not only the colours but also the position of the randvars in the argument list of the corresponding factor are shared. Consequently, ϕ_1 sends a tuple $(\text{blue}, 1)$ to A and a tuple $(\text{blue}, 2)$ to B, and ϕ_2 sends a tuple $(\text{blue}, 2)$ to B and a tuple $(\text{blue}, 1)$ to C (positions are not shown in Fig. 6). Since A and C are both at position one in the argument list of their respective neighbouring factor, they receive identical messages and are recoloured with the same colour. B is assigned a different colour during recolouring than A and C because B received different messages than A and C. The groupings do not change in further iterations and hence the algorithm terminates. The output can be represented by the lifted representation shown on the right in Fig. 6 where both A and C as well as ϕ_1 and ϕ_2 have been grouped together.

B Introducing Logical Variables in Groups of Random Variables

After running the colour passing procedure, randvars and factors belong to groups depending on their assigned colour. In its original form, the CP algorithm is used to run a lifted belief propagation algorithm and thus there is no set of rules specified on how to construct a PFG given the groups after running CP (Kersting, Ahmadi, and Natarajan 2009; Ahmadi et al. 2013). In the following, we describe how the resulting groups after running advanced colour passing (ACP)



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 the CP algorithm 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 procedure is iterated until convergence and the parametric factor graph (PFG) corresponding to the resulting groupings of randvars and factors is depicted on the right. This example builds on Figure 2 from (Ahmadi et al. 2013).

can be used to obtain a PFG entailing equivalent semantics as the initial FG used as input for ACP.

As lifted inference algorithms such as lifted variable elimination (LVE) and the lifted junction tree (LJT) algorithm have been shown to be complete[3] for PFGs containing only parametric factors (parfactors) with at most two logical variables (logvars) and for PFGs containing only parameterised randvars (PRVs) having at most one logvar (Taghipour et al. 2013b; Braun 2020), we concentrate on these two model classes, referred to as $\mathcal{M}^{2lv}$ and $\mathcal{M}^{prv1}$, respectively.

Definition 7. *The model class $\mathcal{M}^{2lv}$ contains every PFG in which each parfactor contains at most two logvars.*

Definition 8. *The model class $\mathcal{M}^{prv1}$ contains every PFG in which each PRV has at most one logvar.*

We refer to $\mathcal{M}^{2lv} \cup \mathcal{M}^{prv1}$ as the *domain-liftable* fragment. $\mathcal{M}^{2lv}$ and $\mathcal{M}^{prv1}$ are able to model a variety of relations and hence provide sufficient expressiveness for most practical applications. We now describe how a PFG is constructed using the groups obtained by running ACP for the domain-liftable fragment $\mathcal{M}^{2lv} \cup \mathcal{M}^{prv1}$.

Let $\phi'(R'_1(X_{1,1},\dots,X_{1,k}),\dots,R'_j(X_{j,1},\dots,X_{j,k}))$ be the new parfactor, build from $\boldsymbol{F} = \{\phi_1(R_{1,1},\dots,R_{1,s}), \dots, \phi_\ell(R_{\ell,1},\dots,R_{\ell,s})\}$ (i.e., $\boldsymbol{F}$ is a group of ℓ factors having the same colour after running ACP). To obtain a correct mapping from $\boldsymbol{F}$ to ϕ', it must hold that $gr(\phi') = \boldsymbol{F}$ and hence $|gr(\phi')| = |\boldsymbol{F}| = \ell$. Recall that $gr(\phi')$ refers to the groundings (i.e., the set of all instances) of ϕ'.

Example 3. *Consider the parfactor $\phi(S(X),T(Y))$ with two PRVs $S(X)$ and $T(Y)$. Let $\mathcal{D}(X) = \{x_1,x_2\}$ and $\mathcal{D}(Y) = \{y_1,y_2\}$. Then, we have $gr(S) = \{S(x_1),S(x_2)\}$, $gr(T) = \{T(y_1),T(y_2)\}$, and $gr(\phi) = \{\phi(S(x_1),T(y_1)), \phi(S(x_1),T(y_2)), \phi(S(x_2),T(y_1)), \phi(S(x_2),T(y_2))\}$.*

Example 4. *Consider the parfactor $\phi(S(X),T(X))$ with two PRVs $S(X)$ and $T(X)$ sharing a single logvar X. Let $\mathcal{D}(X) = \{x_1,x_2\}$. Then, it holds that $gr(S) = \{S(x_1),S(x_2)\}$, $gr(T) = \{T(x_1),T(x_2)\}$ and $gr(\phi) = \{\phi(S(x_1),T(x_1)),\phi(S(x_2),T(x_2))\}$.*

Note that it is also possible for a PRV to have no logvars at all. A PRV R' is parameterless if the group it represents contains only a single randvar—in this case, it holds that $gr(R') = \{R'\}$ and no logvar needs to be introduced.

Now, we are ready to prove Theorem 2, i.e., to prove that ACP introduces the logvars correctly to obtain a valid PFG entailing equivalent semantics as the initial FG for the domain-liftable fragment $\mathcal{M}^{2lv} \cup \mathcal{M}^{prv1}$. In particular, as ACP introduces logvars as specified in Definition 6, we have to prove that applying Definition 6 for the introduction of logvars yields a valid PFG entailing equivalent semantics as the initial FG for the domain-liftable fragment.

Proof of Theorem 2. To match the notation in Definition 6, let $\phi'(R'_1(X_{1,1},\dots,X_{1,k}),\dots,R'_j(X_{j,1},\dots,X_{j,k}))$ be a new parfactor, build from $\boldsymbol{F} = \{\phi_1(R_{1,1},\dots,R_{1,s}),\dots, \phi_\ell(R_{\ell,1},\dots,R_{\ell,s})\}$ and let $\boldsymbol{S} = \{S'_1,\dots,S'_z\}$ denote the subset of ϕ''s arguments with more than one grounding. We prove the correctness of Items 1 to 3 given in Definition 6:

1. We first show that it is not possible for two PRVs $S'_a \in \boldsymbol{S}$ and $S'_b \in \boldsymbol{S}$, $a \neq b$, to have different logvars. For the sake of contradiction, assume that there are $S'_a(X) \in \boldsymbol{S}$ and $S'_b(Y) \in \boldsymbol{S}$ with $X \neq Y$. Then, it holds that $|gr(\phi')| \geq |gr(S'_a)| \cdot |gr(S'_b)|$ because the groundings of ϕ' include all combinations of groundings for PRVs with distinct logvars appearing in ϕ'. A contradiction to our assumption that $|gr(\phi')| = |\boldsymbol{F}| = |gr(S'_a)| = |gr(S'_b)|$ with $|gr(S'_a)| > 1$ and $|gr(S'_b)| > 1$.

[3] A lifted inference algorithm $\mathcal{A}$ is *complete* for a model class $\mathcal{M}$ if $\mathcal{A}$ is domain lifted (that is, $\mathcal{A}$ runs in polynomial time with respect to domain sizes in $\mathcal{M}$) for each query, evidence, and PFG in $\mathcal{M}$ (Van den Broeck 2011).

Now, we show that if there were a PRV $S'_a \in \boldsymbol{S}$ with two logvars, all other PRVs in ϕ' must have the same two logvars and in this case, it is possible to equivalently represent the groundings using a single shared logvar for all PRV in $\boldsymbol{S}$. If there is a PRV $S'_a(X,Y) \in \boldsymbol{S}$ with two logvars X and Y, $X \neq Y$, there can be no logvars other than X and Y in ϕ' due to the restrictions of the domain-liftable fragment. Consequently, as $|gr(S'_a)| = |\mathcal{D}(X)| \cdot |\mathcal{D}(Y)|$ holds and we assumed that all $S'_i \in \boldsymbol{S}$ have the same number of groundings, all $S'_i \in \boldsymbol{S}$ must have the same logvars X and Y. In case all $S'_i \in \boldsymbol{S}$ have the same logvars X and Y, we can equivalently represent the groundings using a single shared logvar Z with $|\mathcal{D}(Z)| = |\mathcal{D}(X)| \cdot |\mathcal{D}(Y)|$ for all $S'_i \in \boldsymbol{S}$.

2. We begin by proving that all PRVs in $\boldsymbol{S}$ have a single logvar. For the sake of contradiction, assume that there is a PRV $S'_a(X,Y) \in \boldsymbol{S}$ with two logvars X and Y such that $X \neq Y$. Then, X and Y are the only logvars in ϕ' due to the restrictions of the domain-liftable fragment. It follows that $|\boldsymbol{F}| = |gr(\phi')| = |\mathcal{D}(X)| \cdot |\mathcal{D}(Y)| = |gr(S'_a)|$, which is a contradiction to our assumption that $\forall S'_i \in \boldsymbol{S}$: $|\boldsymbol{F}| \neq |gr(S'_i)|$. Next, we prove the condition for two PRVs to share the same logvar in two directions.
For the first direction, we show that if two PRVs $S'_a \in \boldsymbol{S}$ and $S'_b \in \boldsymbol{S}$ share the same logvar X, then $|gr(S'_a)| = |gr(S'_b)|$ holds and a bijection τ satisfying the specified condition exists. If S'_a and S'_b share the same logvar X, $|gr(S'_a)| = |\mathcal{D}(X)| = |gr(S'_b)|$ holds. Further, as $gr(S'_a) = \{S'_a(x) \mid x \in \mathcal{D}(X)\}$ and $gr(S'_b) = \{S'_b(x) \mid x \in \mathcal{D}(X)\}$, choosing τ such that $S'_a(x)$ is mapped to $S'_b(x)$ for each $x \in \mathcal{D}(X)$ satisfies the given condition.
For the second direction, we show that if two PRVs $S'_a \in \boldsymbol{S}$ and $S'_b \in \boldsymbol{S}$ have different logvars X and Y, then no bijection τ satisfying the specified condition exists. Let $S'_a(X) \in \boldsymbol{S}$ and $S'_b(Y) \in \boldsymbol{S}$ with $X \neq Y$. As $X \neq Y$, each combination of $S'_a(x)$ and $S'_b(y)$ with $x \in \mathcal{D}(X)$ and $y \in \mathcal{D}(Y)$ appears in exactly one factor in $\boldsymbol{F}$ and thus, for each pair of randvars $(S'_a(x), S'_b(y)) \in gr(S'_a) \times gr(S'_b)$, it holds that $\mathcal{F}(S'_a(x)) \triangle \mathcal{F}(S'_b(y)) \neq \emptyset$ where $\triangle$ denotes the symmetric difference[4] of two sets. Therefore, it is not possible for a bijection τ satisfying the specified condition to exist.

3. We first prove that there are both PRVs with one logvar as well as PRVs with two logvars in $\boldsymbol{S}$. By definition, it holds that $|\boldsymbol{F}| = \prod_{X \in lv(\phi')} |\mathcal{D}(X)|$. Consequently, all PRVs $S'_a \in \boldsymbol{S}$ with $|\boldsymbol{F}| = |gr(S'_a)| = \prod_{X \in lv(S'_a)} |\mathcal{D}(X)|$ contain all logvars occurring in ϕ'. Hence, each $S'_b \in \boldsymbol{S}$ with $|\boldsymbol{F}| \neq |gr(S'_b)|$ cannot contain all logvars occurring in ϕ' and must therefore contain less logvars than the $S'_a \in \boldsymbol{S}$. Due to the restrictions of the domain-liftable fragment, we know that all $S'_a \in \boldsymbol{S}$ then have two logvars and all $S'_b \in \boldsymbol{S}$ have one logvar. Furthermore, there are exactly two distinct logvars in ϕ' and hence, all $S'_a \in \boldsymbol{S}$ share the same two logvars X_1 and X_2 and all $S'_b \in \boldsymbol{S}$ have either X_1 or X_2 as their logvar. Item 2 completes the proof for the conditions for two PRVs to share the same logvar.

The correctness for introducing the domain sizes follows by definition as the number of groundings of a PRV equals the size of the group of randvars it represents. □

A visualisation of the three cases is given in Fig. 7. In Fig. 7a, it holds that $|\boldsymbol{F}| = |gr(S'_1)| = |gr(S'_2)| = 2$ and thus S'_1 and S'_2 share the same logvar X with $|\mathcal{D}(X)| = 2$. Figure 7b shows an example of a parfactor containing two PRVs with distinct logvars X and Y with $|\mathcal{D}(X)| = 2$ and $|\mathcal{D}(Y)| = 2$ as $4 = |\boldsymbol{F}| \neq |gr(S'_1)| = |gr(S'_2)| = 2$. A similar situation is depicted in Fig. 7c where a single PRV has two logvars X and Y with $|\mathcal{D}(X)| = 2$ and $|\mathcal{D}(Y)| = 2$. Thus, it holds that $4 = |\boldsymbol{F}| = |gr(S'_2)| \neq |gr(S'_1)| = 2$.

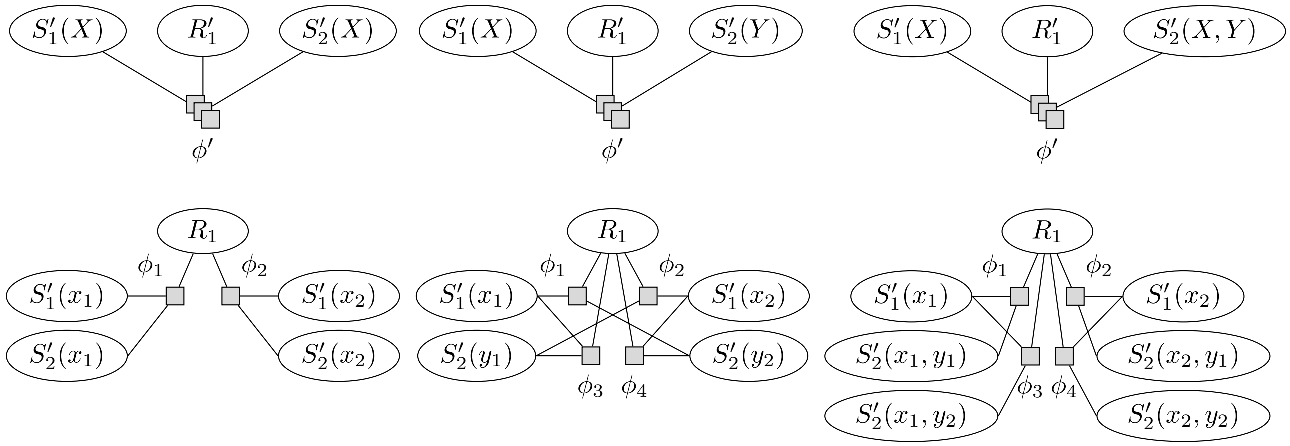
Note that Definition 6 can also be applied on the resulting groups obtained from running CP instead of ACP to yield a valid PFG. So far, we have proven that ACP introduces logvars correctly. Next, we give the conditions for ACP to count convert a PRV and then show that the introduction of counting randvars (CRVs) is also handled correctly by ACP.

To check whether a CRV is required in the argument list of $\phi'(R'_1(X_{1,1}, \dots, X_{1,k}), \dots, R'_j(X_{j,1}, \dots, X_{j,k}))$, it is sufficient to compare the number j of arguments of ϕ' to the number s of arguments of the $\phi_i \in \boldsymbol{F} = \{\phi_1(R_{1,1}, \dots, R_{1,s}), \dots, \phi_\ell(R_{\ell,1}, \dots, R_{\ell,s})\}$. Note that all ϕ_i have the same colour and thus the same number of arguments. However, it is possible for ϕ' to have less arguments than each ϕ_i if there are randvars in the argument lists of each ϕ_i that are in the same group.

If the number of arguments of ϕ' is equal to the number of arguments of all ϕ_i (i.e., $j = s$), there is no need to introduce a CRV in ϕ' as the potentials of the ϕ_i can be copied into ϕ' (potentials are identical for all ϕ_i as they are in the same group). However, if ϕ' has less arguments than the ϕ_i (i.e., $j < s$), a CRV is required to equivalently represent the potentials of the ϕ_i in ϕ'. If $j < s$, there are at least two randvars in the argument lists of each ϕ_i that are in the same group and hence these randvars must be in the set of commutative arguments of the ϕ_i (otherwise, they would have received different messages and thus would not be in the same group). Therefore, if $j < s$, all ϕ_i are guaranteed to be commutative with respect to their arguments that are in the same group and thus a CRV can be used to equivalently represent the potentials of the ϕ_i in ϕ'.

As a final remark regarding the introduction of CRVs, we note that it is also conceivable to recursively check the remaining argument lists of the $\phi_i \in \boldsymbol{F}$ for commutativity after introducing a CRV. However, it is not immediately clear how the preconditions for count conversion (Taghipour 2013) are ensured then. As the preconditions get more restrictive for additional CRVs in a single parfactor and the total number of arguments of a factor is usually small in practice, we expect that there are very few scenarios in which the extra complexity of searching for additional CRVs pays off.

[4]The symmetric difference of two sets A and B is defined as $A \triangle B = (A \setminus B) \cup (B \setminus A)$.



(a) (b) (c)

Figure 7: A visualisation of the three cases given in Definition 6 based on small examples. The domains of the two logvars X and Y are given by $\mathcal{D}(X) = \{x_1, x_2\}$ and $\mathcal{D}(Y) = \{y_1, y_2\}$, respectively. In (a), there is a parfactor ϕ' with a single logvar X shared across two PRVs S'_1 and S'_2, (b) shows a parfactor ϕ' containing two PRVs S'_1 and S'_2 with distinct logvars X and Y, and (c) displays a parfactor ϕ' with a PRV S'_2 containing two logvars X and Y. The groundings of the parfactors are illustrated below the respective parfactor. In all three cases, the parfactor ϕ' contains an additional parameterless PRV R'_1.

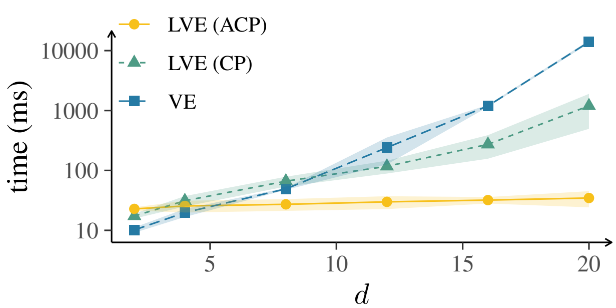


Figure 8: Average query times and their standard deviation of LVE on the output of CP, LVE on the output of ACP, and VE on the initial (propositional) FG for input FGs containing three commutative factors.

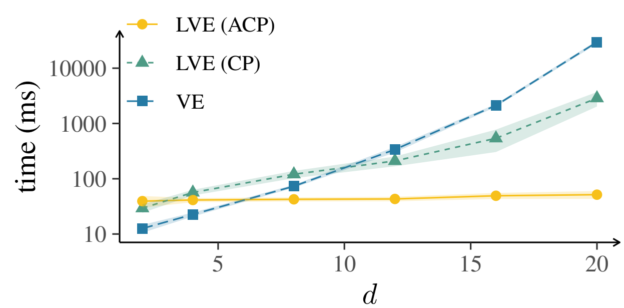


Figure 9: Average query times and their standard deviation of LVE on the output of CP, LVE on the output of ACP, and VE on the initial (propositional) FG for input FGs containing seven commutative factors.

C Further Experimental Results

In addition to the experimental results provided in Section 5, we also report further results for input FGs containing different proportions of commutative factors and factors having permuted arguments. Again, we evaluate the impact of commutative factors and factors with permuted arguments on query times separately by comparing the average query times of LVE on the resulting PFG after running ACP, denoted as LVE (ACP), of LVE on the resulting PFG after running CP, denoted as LVE (CP), and of variable elimination (VE) on the initial FG.

Figures 8 and 9 depict the results for FGs containing between eight and 104 factors of which $k = 3$ and $k = 7$ factors are commutative, respectively. In total, there are between $2d + (k-2) \cdot (d+1) + 2$ and $d \cdot \lfloor \log_2(d) \rfloor + 2d + 2 + (k-1) \,/\, 2 \cdot (d+1)$ Boolean randvars and between $d \cdot \lfloor \log_2(d) \rfloor + d + k + 1$ and $2d + (k-2) \cdot (d+1) + 2$ factors for each domain size $d = 2, 4, 8, 12, 16, 20$. There are no symmetries between factors which cannot be detected by CP (i.e., there are no permuted arguments) and the maximum number of arguments of a factor is $d + 1$. For each choice of d, we again measure the run times for two queries on multiple FGs and report the average query times and their standard deviation. The results shown in Figs. 8 and 9 are similar to the results given in Fig. 4 (y-axes are log-scaled again). Overall, the run times are generally slightly higher than for FGs containing only one commutative factor because the FGs are a bit larger. The results show that ACP handles larger FGs without any problems while CP runs again into scalability issues even for rather small domain sizes. As expected, VE is again the slowest of all algorithms.

Further experimental results for input FGs containing symmetries between factors with permuted argument lists can be found in Figs. 10 to 12 (y-axes are log-scaled again).

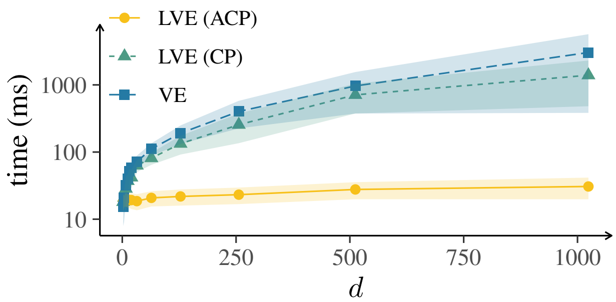


Figure 10: Average query times and their standard deviation of LVE on the output of CP, LVE on the output of ACP, and VE on the initial (propositional) FG for input FGs where about five percent of the factors have permuted arguments.

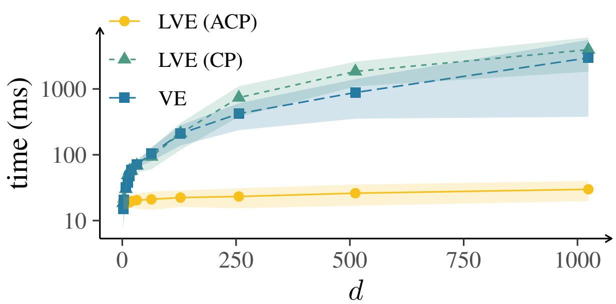


Figure 11: Average query times and their standard deviation of LVE on the output of CP, LVE on the output of ACP, and VE on the initial (propositional) FG for input FGs where about ten percent of the factors have permuted arguments.

In the input FGs depicted in Figs. 10 to 12, about five, ten, and 15 percent of the factors (randomly chosen) have permuted argument lists, respectively, and there are no commutative factors. As in Fig. 5, there are between $5d+1$ and $d \cdot \lfloor \log_2(d) \rfloor + 2d + 1$ Boolean randvars and between $2d$ and $d \cdot \lfloor \log_2(d) \rfloor + d + 1$ factors in the input FGs for each domain size $d = 2, 4, 8, 12, 16, 20, 32, 64, 128, 256, 512, 1024$. The maximum number of arguments of a factor is again seven and we evaluate two queries per input FG for each domain size d and then report the average run time and standard deviation over all queries for that choice of d. Not surprisingly, the results shown in Fig. 10 exhibit similar patterns as the results in Fig. 5. The advantage of running LVE on the PFG obtained by running CP compared to running VE on the initial propositional FG is now less pronounced, as CP is not able to detect the symmetries between factors with permuted argument lists. Therefore, there are more groups in total after running CP and hence the resulting PFG is less compact. As ACP is able to detect symmetries between factors with permuted argument lists, the percentage of factors with permuted argument lists does not impact the compression obtained by running ACP and hence the run times of LVE on the output of ACP are not negatively affected by a higher percentage of permuted factors. When further increasing the percentage of permuted factors, the run times of LVE on the output of CP drastically increase, as shown

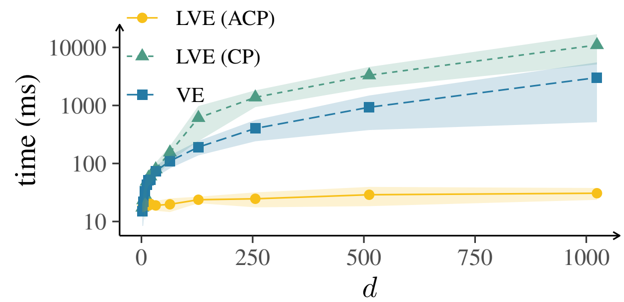


Figure 12: Average query times and their standard deviation of LVE on the output of CP, LVE on the output of ACP, and VE on the initial (propositional) FG for input FGs where about 15 percent of the factors have permuted arguments.

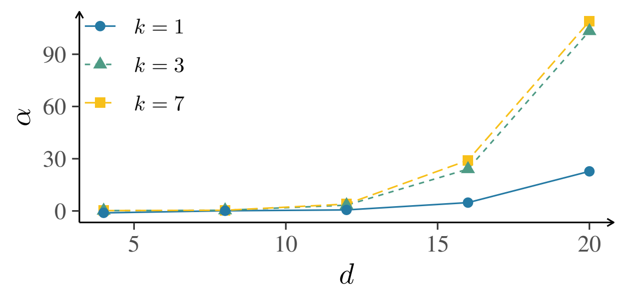


Figure 13: Average number α of queries after which the additional offline effort amortises for input FGs containing a total of k commutative factors.

in Figs. 11 and 12. Interestingly, running VE on the initial FG becomes even faster than running LVE on the PFG obtained by running CP if at least ten percent of the factors have permuted argument lists. The result that LVE (CP) becomes slower than VE can by explained by the fact that LVE introduces some overhead compared to VE. Even though the overhead induced by LVE is rather small, CP groups just a few randvars and factors, respectively, and LVE does not benefit a lot from a vast amount of small groups. Therefore, the results suggest that the compression obtained by CP does not compensate for the overhead of LVE. Figures 11 and 12 emphasise that checking for permutations is indispensible for CP to be effective in practical applications.

Moreover, we report the average number α of queries after which the additional offline effort of ACP compared to CP amortises in Figs. 13 and 14. In particular, it holds that $\alpha = \frac{\Delta_o}{\Delta_g}$ with Δ_o ("offline overhead") denoting the difference of the offline time required by ACP and CP to obtain the PFG and Δ_g ("online gain") denoting the difference of the time required by LVE on the output of CP and ACP to answer a query. Thus, after α queries, the additional time needed during the offline step by ACP is saved by the faster query times of LVE on the output of ACP. Figure 13 shows the average α for different domain sizes $d = 4, 8, 12, 16, 20$ on the input FGs from Figs. 4, 8 and 9. Each line in Fig. 13

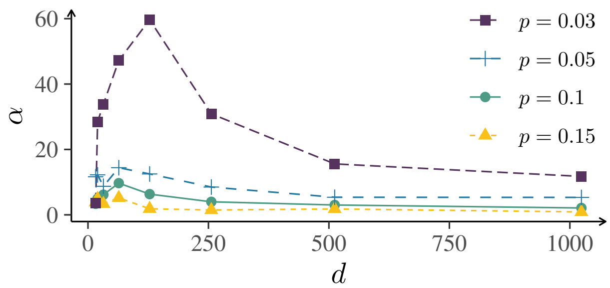


Figure 14: Average number α of queries after which the additional offline effort amortises for input FGs where a proportion p of the factors has permuted arguments.

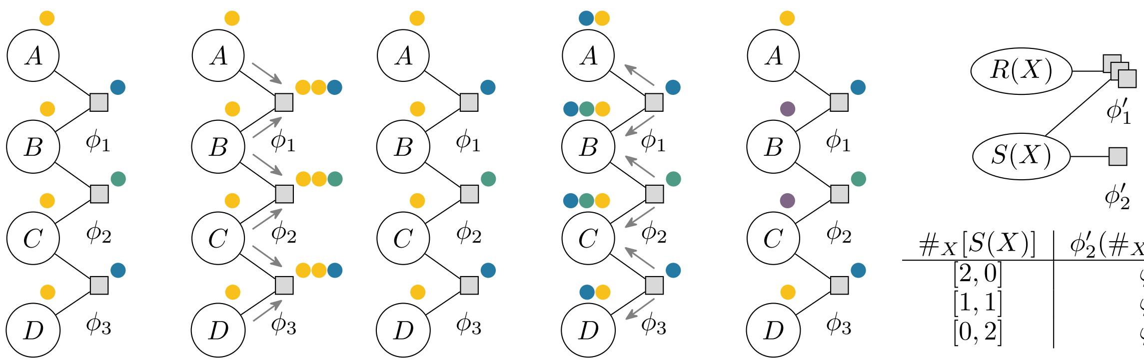
corresponds to a different number k of commutative factors present in the input FG. For domain sizes $d < 16$, far less than ten queries are sufficient to save the additional time needed by ACP during the offline step for all choices of k. At $d = 16$ and $d = 20$, α increases slightly for $k = 1$ while it increases more steeply for $k = 3$ and $k = 7$, showing that larger graphs require quite some offline effort (the graph size increases with increasing values for k). However, it is important to keep in mind that the offline step can be performed in advance before a system is deployed and hence, the additional offline overhead is still worth the effort in practical applications, especially if computing resources are limited during online inference. We also remark that the standard deviation, which is not graphically represented in Fig. 13, greatly varies between different domain sizes d. In particular, the standard deviation is about 30 for $d = 20$ while it is mostly far below ten for all other choices of d. Figure 14 shows the average α for different domain sizes $d = 16, 20, 32, 64, 128, 256, 512, 1024$ on the input FGs from Figs. 5 and 10 to 12. Each line corresponds to a different proportion p of factors with permuted arguments. Note that the graph size does not depend on p and therefore it is not surprising that the average α decreases the larger the proportion p of factors with permuted arguments is (because the offline overhead of ACP remains identical while at the same time the online gain increases for larger values of p). Except for some outliers for $p = 0.03$, the average α stays below 20 for all choices of d. It is also noteworthy that α mostly remains below ten, indicating that the detection of permuted factors often amortises faster than the detection of commutative factors. We report a rather high standard deviation for most of the displayed scenarios, which supports the intuition that permutations are sometimes found very fast and sometimes require quite some time.

D Example Run of Advanced Colour Passing

Figure 15 illustrates ACP on an example input FG containing both a commutative factor ϕ_2 and two factors ϕ_1 and ϕ_3 representing identical potentials but the potentials are not written in the same order into their tables. In this example, all randvars are Boolean and there is no evidence (i.e., $\boldsymbol{E} = \emptyset$). Initially, ACP assigns all randvars the same colour (e.g., yellow) because they have the same range (Boolean) and the same observed event (no event). ϕ_2 is assigned its own colour (e.g., green) and ϕ_1 and ϕ_3 are assigned the same colour (e.g., blue) because they represent identical potentials when swapping, e.g., the order of C and D in ϕ_3. Hence, ϕ_3 includes position one into its message to D and position two into its message to C. 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. Therefore, after recolouring the factors, the colour assignments are identical to the initial assignments. 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, else the position is replaced by zero. During this step, A receives a message $(\text{blue}, 1)$ from ϕ_1, B receives a message $(\text{blue}, 2)$ from ϕ_1 and a message $(\text{green}, 0)$ from ϕ_2, C receives a message $(\text{green}, 0)$ from ϕ_2 and a message $(\text{blue}, 2)$ from ϕ_3, and D receives a message $(\text{blue}, 1)$ from ϕ_3 (remember that the positions of C and D have been swapped at the beginning). Hence, A and D as well as B and C receive identical messages (positions are not shown in Fig. 15). After the recolouring step, A and D share a colour and B and C share a different colour. The groupings do not change in later iterations and the resulting PFG is shown on the right.

Both groups of randvars are represented by a PRV ($R(X)$ for A and D, and $S(X)$ for B and C), respectively, ϕ_1 and ϕ_3 are replaced by a parfactor $\phi'_1(R(X), S(X))$, and ϕ_2 is replaced by a parfactor $\phi'_2(\#_X[S(X)])$ in which $S(X)$ appears count converted. The PFG contains an edge between ϕ'_1 and $R(X)$ because there exists a randvar in the group represented by $R(X)$ which is connected to a factor in the group represented by ϕ'_1 in the original FG (A is connected to ϕ_1 and D is connected to ϕ_3). Moreover, there are edges between ϕ'_1 and $S(X)$ and between ϕ'_2 and $S(X)$ because there are edges between, for example, ϕ_1 and B as well as between ϕ_2 and B in the original FG.

Since ϕ'_1 has the same number of arguments as ϕ_1, there are no CRVs in ϕ'_1. In particular, there are two PRVs with a shared logvar because $|gr(R(X))| = |gr(S(X))| = |gr(\phi'_1)| = 2$ (all groups contain exactly two elements, that is, A and D, B and C, as well as ϕ_1 and ϕ_3, respectively). As the number of arguments of ϕ'_2 has changed compared to the number of arguments of ϕ_2, a CRV is used to represent the potentials of ϕ_2 in ϕ'_2. In this example, ϕ_2 is commutative with respect to both B and C and therefore, ϕ'_2 has only a single argument counting over all elements of the group $\{B, C\}$. Note that $S(X)$ appears "normal" in ϕ'_1 and count converted in ϕ'_2. Running CP on the PFG given in Fig. 15 ends up without grouping together anything.



$\#_X[S(X)]$	$\phi'_2(\#_X[S(X)])$
$[2, 0]$	φ_5
$[1, 1]$	φ_6
$[0, 2]$	φ_7

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_5
true	false	φ_6
false	true	φ_6
false	false	φ_7

C	D	$\phi_3(C, D)$
true	true	φ_1
true	false	φ_3
false	true	φ_2
false	false	φ_4

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

Figure 15: A visualisation of the steps undertaken by Algorithm 1 on an input FG with only Boolean randvars and no evidence (left). The general routine of the CP algorithm remains the same, but additionally commutative factors are recognised and potentials are compared independently of the factors' arguments order. The resulting PFG is depicted on the right, where two PRVs over a logvar X with $|\mathcal{D}(X)| = 2$ replace the two groups of size two (A, D and B, C). Note that the PRV $S(X)$ appears "normal" in ϕ'_1 and count converted in ϕ'_2.