Lifted Reasoning for Combinatorial Counting

Submission ID: 7715

Abstract

Combinatorics math problems are often used as a benchmark to test human cognitive and logical problem-solving skills. These problems are concerned with counting the number of solutions that exist in a specific scenario that is often sketched in natural language. Humans are adept at solving such problems as they can identify commonly occurring structures in the questions for which a closed-form formula exists for computing the answer. These formulas exploit the exchangeability of objects and symmetries to avoid a brute-force enumeration of all possible solutions. Unfortunately, current AI approaches are still unable to solve combinatorial problems in this way. This paper aims to fill this gap by developing novel AI techniques for representing and solving such problems. It makes the following three contributions. First, we propose a declarative language that is able to represent a wide range of combinatorial reasoning problems. Second, we propose a novel lifted solving algorithm bridging probabilistic inference techniques and constraint programming. Third, we show empirically how our solver can outperform existing approaches for solving combinatorial reasoning problems.

1 Introduction

Some of the most interesting challenges in AI come from tasks that humans are able to solve elegantly by means of abstract reasoning. One of the fundamental goals of AI is to outperform humans on tests associated to intelligence and advanced cognitive skills, like puzzles (Chesani, Mello, and Milano 2017), games (Silver et al. 2016) and math and probability problems (Mitra and Baral 2016; Roy and Roth 2018; Dries et al. 2017). For this reason, an important source of inspiration to develop better AI tools comes from quizzes, exercises and exams that test cognitive skills, such as SAT, GMAT or GRE. These tests require logical and mathematical proficiency along with text comprehension skills. The classes of problems adopted include combinatorics math problems. The task is to count the number of possible configurations of a setting, described in natural language. Some problems can be solved by simply identifying a counting rule associated to the setting, as Problem 1. Others provide additional constraints that prevent the application of a closed-form formula (Problem 2).

Combinatorics problem occupy an interesting position between *counting constraint satisfaction problems* (#CSP) and probability. On the one hand, they can, as we will show, be modeled as instances of #CSPs and we will develop a declarative language for this. On the other hand, solving combinatorial problems requires reasoning about high-level symmetries, in a way that resembles *lifted* probabilistic inference (Poole 2003) for probabilistic logics. This raises the question as to whether the lifted reasoning techniques for probabilistic inference could be adapted for combinatorics problems, and given that we represent combinatorics problems as counting constraint satisfaction problems, whether lifted inference can be adapted to these problems as well. This question is answered positively in this paper by combining the strengths of constraint languages and lifted inference. In particular, we present the following contributions:

- We define a declarative language, CoLa, to express combinatorial problems (Section 4).
- We define concepts and algorithms that allow an efficient (lifted) solving strategy for counting constraint satisfaction problems (Section 5)
- We implement a solver, CoSo, for a class of combinatorics problem (Section 6) showing an application of our approach.
- We compare CoLa and CoSo to the existing probabilistic inference and constraint programming tools and languages (Section 7).

2 Combinatorial Problems

First, we descirbe a framework for defining combinatorics math problems. (Stanley 2012) presents a characterization of the most common combinatorial problems (*Twelvefold Way*) as the problem of counting the number of functions $f : X \to Y$ between two finite sets X, Y ($|X| = x, |Y| = y$). Imposing different restrictions over the functions yields

There are 5 doors to a lecture room. In how many ways can a student enter the room through a door and leave the room by a different door?

Problem 1: GMAT question example

	$\mathcal{F}_{\text{any}}$	$\mathcal{F}_{\text{inj}}$	$\mathcal{F}_{\text{sur}}$
$\neq$	x-sequence of Y y^x	x-permutation of Y $y^{\underline{x}}$	comp. X into y subs. $\left\{ {x \atop y} \right\} y!$
$=^X$	x-multisubset of Y $\binom{y+x-1}{x}$	x-subsets of Y $\binom{y}{x}$	comp. x into y subs. $\binom{x-1}{x-y}$
$=_Y$	part. X into $\leqslant$ y subs. $\sum_{k=0}^{x} \left\{ {x \atop k} \right\}$	$[x \leqslant y]$	part. X into y subs. $\left\{ {x \atop y} \right\}$
$=_Y^X$	part. x into $\leqslant$ y subs. $\sum_{k=0}^{x} p_k(y)$	$[x \leqslant y]$	part. x into y subs. $p_y(x)$

Table 1: Basic combinatorial structures

different classes of combinatorial problems. Functions can be subject to two kinds of restrictions: the type of function itself and the distinguishability of the elements in X and Y. There are three types of functions: arbitrary ($\mathcal{F}_{\text{any}}$), injective ($\mathcal{F}_{\text{inj}}$) or surjective ($\mathcal{F}_{\text{sur}}$). Two functions $f, g : X \to Y$ are said to be *equivalent with X indistinguishable* ($=^X$) if there exists a bijection $b : X \to X$ such that $f(b(a)) = g(a)$ for all $a \in X$. The definitions for the other cases with Y indistinguishable ($=_Y$) and both X, Y indistinguishable ($=_Y^X$) are similar (see (Stanley 2012)). The distinguishability of the elements of the two sets defines the equivalence relation $=$ over functions: the counting problem is to find the number of equivalence classes induced by $=$. When both sets are distinguishable ($\neq$) this corresponds to counting the number of functions. Table 1 displays the combinations of function type and distinguishability, each corresponding to a particular arrangement of elements in the traditional nomenclature of combinatorics: sequences, subsets, compositions and partitions (we consider permutations as a special case of sequences). We refer to them as *combinatorial structures*; the *size* of a combinatorial structure is x for sequences and subsets, y for compositions and partitions. For each of these structures a closed-form counting rule is available. We use the following notation: $y^{\underline{x}} = x(x-1)\dots(x-n-1)$, $[e] = 1$ if e is true, 0 otherwise, $\left\{ {\cdot \atop \cdot} \right\}$ is the Stirling number of the second kind, and $p_i(n)$ is the number of integer partitions of n with i terms.

Example 1. *Figure 1 shows two sets $X = \{1, 2\}$ and $Y = \{red, green, blue\}$. On the left we enumerated the injective functions $f : X \to Y$ where both X and Y are distinguishable, on the right where the elements in X are indistinguishable. The left part of the figure corresponds to 2-permutations of Y, while the right side shows the 2-subsets.*

Counting Constraint Satisfaction Problems

We briefly introduce #CSP problems.

Definition 1. *Let $\mathcal{V}$ be a set of finite variables and D be a finite set of values (domain). A* constraint *is a pair $(\boldsymbol{v}, R)$ where $\boldsymbol{v}$ is an n-tuple of $\mathcal{V}$ and $R \subseteq D^n$ is a relation of arity n over D. Let $f : \mathcal{V}^n \to D^n : f((V_1, \dots, V_n)) = (d_1, \dots, d_n)$ be an* assignment *function. An assignment f satisfies a constraint $c = (\boldsymbol{v}, R)$ when $f(\boldsymbol{v}) \in R$:$f(\boldsymbol{v}) \vdash c$. f is a* partial *assignment when $n < |\mathcal{V}|$.*

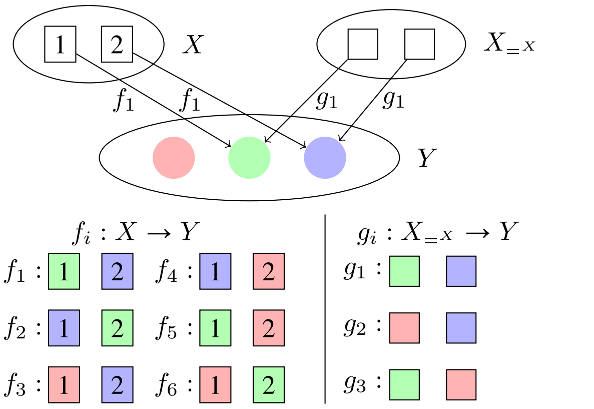

Figure 1: Functions corresponding to the six 2-partitions (left) and the three 2-subsets (right) of Y

When the arity of a constraint c is equal to the number of variables we call c a *global constraint*.

Definition 2. *A* constraint satisfaction problem *(*CSP*) is a tuple $\langle \mathcal{V}, D, C \rangle$ where V is a set of variables, D a domain and C a set of constraints. An assignment f* satisfies *a* CSP, *if f satisfies all the constraints in C. The task in CSPs is to determine whether a satisfying assignment exists.*

The set C of constraints is usually expressed by means of a *constraint modelling language*. A satisfying assignment is also called a *solution* or *model* for the CSP. We refer to the set of models as $M(\mathcal{V}, D, C)$.

Definition 3. *A #CSP problem (model counting problem) is the following: given a* CSP *problem $\langle \mathcal{V}, D, C \rangle$, find the number $MC(\mathcal{V}, D, C)$ of satisfying assignments.*

Combinatorics problems can be expressed as #CSPs as follows: given the family of functions $f : X \to Y$, the elements of the set X are encoded as a set of variables $\mathcal{V}$ whose domain is the set Y. The constraint language includes $\mathcal{F}_{\text{inj}}, \mathcal{F}_{\text{sur}}$ and can be further extended as we will show in Section 4. #CSPs are a generalizaion of #SAT, which can be formulated as follows: logic terms are variables mapped to a domain $D = \{\top, \bot\}$ and C is a set of constraints expressed as predicates in propositional logic. Viceversa, #SAT is known to be a complete problem for the class of counting problems #P (Valiant 1979), therefore any counting problem, including #CSPs, can be reduced to an instance of #SAT. In the next section we discuss why lifted reasoning techniques from probabilistic inference can be used to also solve general model counting problems and why constraint programming can inspire better lifted algorithms for the class of counting problems.

3 Model Counting

Probabilistic inference relies on #SAT reductions and solving techniques to efficiently compute the number of assignments satisfying a given propositional formula in a (probabilistic) logic theory. We distinguish two approaches to probabilistic inference: variable elimination

(VE) (Zhang and Poole 1994) and knowledge compilation (Cadoli and Donini 1997). *Lifted* probabilistic inference techniques (Poole 2003; Niepert and den Broeck 2014) exploit high-level symmetries of first-order theories that would otherwise get lost by reasoning on a propositional level. GC-FOVE (Taghipour et al. 2012) generalizes First-Order VE (de Salvo Braz, Amir, and Roth 2005) using *constraint trees* to represent the set of admissible tuples. While they optimize set-operations on sets of satisfying assignments, we manipulate constraints without explicitely representing assignments. First-order weighted model counting (FOWMC) (den Broeck et al. 2011) lifts the knowledge compilation approach by introducing first-order knowledge compilation rules. The compilation rules use a constraint language that allows expressing binary constraints on logical variables according to expressions of the form $t_1 = t_2$ where t_i is either a variable or a constant, and $v \in D$ where v is a variable and D is a set of constants. Algebraic Model Counting (AMC) (Kimmig, den Broeck, and De Raedt 2017) generalizes probabilistic inference, #SAT, and (weighted) model counting and shows how (first-order) knowledge compilation solves an AMC task. (den Broeck, Meert, and Darwiche 2014; Kazemi et al. 2016) show how first-order knowledge compilation leads to polynomial complexity results for some classes of first-order theories, including theories up to two logical variables. However, in Section 7, we argue that FOWMC's constraint language is too limited to model and fully exploit the symmetries of combinatorial problems.

Symmetries are an important source of search optimization for constraint satisfaction problems (CSPs). In particular global constraints (Régin 2010) expose symmetries that allow for more efficient search strategies compared to a set of equivalent binary constraints (Régin 1996). $\mathcal{F}_{\text{inj}}$ and $\mathcal{F}_{\text{sur}}$ are examples of global constraints, in particular injectivity corresponds to the famous *alldifferent* constraint, and surjectivity can be expressed as a *global cardinality constraint* where the interval is $\{1, \ldots, |Y|\}$ for each value in Y.

Combinatorial counting problems represent an interesting meeting point of the two worlds, where the constraint language can be restricted to the foundational constraints ($\mathcal{F}_{\text{inj}}, \mathcal{F}_{\text{sur}}$) and simple counting constraints (see Section 4). This restriction allows to investigate how the concepts of lifted reasoning from probabilistic inference can be combined with a more expressive constraint language that goes beyond simple binary constraints. Motivated by this, we present a language that defines such a framework (Section 4) and general lifted reasoning concepts that apply to #CSPs (Section 5). Finally, we show how the two allow the development of solving techniques for combinatorial problems that mimic human reasoning: a solution is computed by (recursively) decomposing the problem into simpler subproblems where it is possible to apply the known counting rules.

There are 8 students: 3 speak both Dutch and French, 3 only Dutch, and 2 only French. How many rows of four students can we form where the third student speaks French and there are at least two Dutch-speaking students?

Problem 2: Combinatorial problem with constraints

4 Modelling Combinatorial Problems

We introduce a declarative modeling language for specifying combinatorial counting problems (CoLa). *Elements, domains and structures* are CoLa's central constructs inspired by the *Twelvefold Way* framework. In addition, we use a simple constraint language to express more sophisticated problems, such as Problem 2.

Elements, Domains and Structures

Elements are constants corresponding to atomic objects. For instance, in Problem 2, the constants $s_1, \ldots, s_8$ represent students. *Domains* are sets of elements: a *universe* set is a domain including all the elements of the problem. For example, $student = \{s_1, \ldots, s_8\}$ is a universe set, $dutch = \{s_1, \ldots, s_6\}$ and $french = \{s_3, \ldots, s_8\}$ identify properties of elements. *Structures* are pairs (δ, ϕ) where $\delta \in \{\neq, =^X, =_Y, =^X_Y\}$ and $\phi \in \{\mathcal{F}_{\text{any}}, \mathcal{F}_{\text{inj}}, \mathcal{F}_{\text{sur}}\}$: they define an arrangement of elements. A statement $\#s = n$ defines the size of a structure s. In Problem 2, students are arranged into a structure s, $structure(s, \neq, \mathcal{F}_{\text{inj}}, student)$ (permutation), of size 4, $\#s = 4$.

Constraints

The constraint langage is based on: domain formulas, choice constraints and counting constraints.

Domain formulas. Domain formulas model properties of elements by means of the traditional set operations. A domain formula is a statement the form $\varphi_1 \cap \varphi_2$, $\varphi_1 \cup \varphi_2$ or $\overline{\varphi_1}$, where φ_i is either a domain or a domain formula. $^{-}$ is the set-complement w.r.t. the universe. An element satisfies a domain formula if it belongs to the corresponding domain. For instance, the domain formula $dutch \cap \overline{french}$ defines the set of Dutch-speaking students who do not speak French.

Choice constraints. Choice constraints fix the mapping of one or more elements. Given a structure s, $pos(s, n, \varphi)$ constrains the element in position n of a sequence s to satisfy a domain formula φ. $in(s, e)$ constrains element e to belong to the subset s. $composition(s, n, e)$ constrains element e to belong to the n^{th} composition and $partition(s, e_1, e_2)$ constrains elements e_1 and e_2 to belong to the same partition. For instance, the constraint from Problem 2 about the third student of the sequence is expressed as $pos(s, 3, french)$.

Counting constraints. Counting constraints are statements of the form $\#_s\, c \star n$: the number of elements of a structure s satisfying constraint c is related to n by $\star \in \{<, >, \leqslant, \geqslant, =\}$. Constraint c depends on the structure: if s is a sequence or a subset, within s we observe properties of elements, hence domain formulas, otherwise, in the case of partitions or combinations we observe properties of sets, therefore we observe counting constraints. For instance, the last sentence in Problem 2 is translated to $\#_s\, dutch \geqslant 2$.

Example 2. *We model Problem 2 in CoLa as:*
$student = \{s_1, \ldots, s_8\}$.
$dutch = \{s_1, \ldots, s_6\}$.

$french = \{s_3, \ldots, s_8\}$.
$structure(s, \neq, \mathcal{F}_{inj}, student)$.
$\#s = 4$.
$pos(s, 3, french)$.
$\#_s\, dutch \geqslant 2$.

5 Lifted Reasoning over Counting Problems

In this section we present lifted reasoning strategies for general counting *constraint satisfaction problem* (#CSP) that allow defining efficient counting algorithms. In Section 6 we show how these concepts can be implemented to compute the solution of combinatorics problems.

Definition 4. *Given two tuples of disjoint variables $\boldsymbol{v}_1, \boldsymbol{v}_2$ and two assignments $f_1(\boldsymbol{v}_1) = (d_1, \ldots, d_m), f_2(\boldsymbol{v}_2) = d_{m+1}, \ldots, d_n)$, the* union *of the two assignments $f_1 + f_2$ is the tuple: $(d_1, \ldots, d_m, d_{m+1}, \ldots, d_n)$. We generalize to the union of two sets M_1, M_2 of partial assignments as $M_1 + M_2 = \bigcup_{\boldsymbol{d}_1 \in M_1} \bigcup_{\boldsymbol{d}_2 \in M_2} \{\boldsymbol{d}_1 + \boldsymbol{d}_2\}$.*

Property 1. *Given two* CSP*s $\langle \mathcal{V}_1, D, C_1 \rangle$,$\langle \mathcal{V}_2, D, C_2 \rangle$,$\mathcal{V}_1 \cap \mathcal{V}_2 = \varnothing$, $MC(\mathcal{V}_1 \cup \mathcal{V}_2, D, C_1 \cup C_2) = MC(\mathcal{V}_1, D, C_1) \cdot MC(\mathcal{V}_2, D, C_2)$ (multiplication rule).*

Exchangeability is a fundamental concept for lifted reasoning, since it allows reasoning over groups of (exchangeable) variables rather than individuals: recognizing and exploiting exchangeability brings exponential improvements compared to grounding and handling each exchangeable assignment separtaely.

Definition 5. *A tuple of variables $\mathcal{V} = (V_1, \ldots, V_n)$ is* exchangeable *($\equiv^{\mathcal{V}}$) w.r.t. a* CSP *$\langle \mathcal{V}, D, C \rangle$ if for all satisfying assignments $(V_1 = d_1, \ldots, V_n = d_n\})$ and all permutations π of $\{1, \ldots, n\}$, $\{V_1 = d_{\pi(1)}, \ldots, V_n = d_{\pi(n)}\}$ is a satisfying assignment as well.*

Property 2. *Given a* CSP *$\langle \mathcal{V}, D, C \rangle$ and satisfying assignment $(V_1 = d_1, \ldots, V_n = d_n)$, if $(V_1, \ldots, V_n)$ is exchangeable then there are other $n! - 1$ satisfying assignments.*

We also define *distinguishable assignments*:

Definition 6. *Given a* CSP *$\langle \mathcal{V}, D, C \rangle$ indistinguishability is a relation "$=$" such that $= \subseteq M(\mathcal{V}, D, C) \times M(\mathcal{V}, D, C)$. We say that $\boldsymbol{d} = (d_1, \ldots, d_n)$ is distinguishable from $\boldsymbol{d'} = (d'_1, \ldots, d'_n)$ if $(\boldsymbol{d}, \boldsymbol{d'}) \notin =$, indistinguishable otherwise.*

In this paper the combinatorics setting requires us to discriminate indistinguishability, in the same sense as in the Twelvefold Way, from exchangeability. Note that in the probabilistic inference literature exchangeable objects are often referred as indistinguishable or interchangeable (Taghipour et al. 2012; Milch et al. 2008). From a constraint modelling perspective, *symmetry breaking*, i.e. constraints excluding assignments that are considered to represent the same solution of some satisfying assignment, are often related to distinguishability. For instance, in the famous n-queens problem solutions that correspond to vertical, horizontal and rotational symmetries of the chessboard can be considered to be indistinguishable. The following variation of a #CSP incorporates the distinguishability relation:

Definition 7. *A #CSP with distinguishability, $\#CSP^{=}$ is the problem of counting the number of* distinguishable *satisfying assignments for the associated* CSP.

We define lifted reasoning w.r.t. #CSP (and $\#CSP^{=}$):

Definition 8. *Domain-lifted model counting algorithms run in time polynomial in the size of the domain.*

This definition, as well as Definition 5, follows the notions about lifted probabilistic inference presented in (Niepert and den Broeck 2014). Similarly, we define two fundamental operations of lifted inference: splitting and shattering.

Definition 9. *Given a #CSP $MC(\mathcal{V}_1 \cup \mathcal{V}_2, D, C)$, $\mathcal{V}_1 \cap \mathcal{V}_2 = \varnothing$, a* split *is pair of* $\#$CSP*s $(\mathbb{S}_1, \mathbb{S}_2)$, $\mathbb{S}_i = MC(\mathcal{V}_i, D, C_i)$, such that $\mathbb{S}_1 \cdot \mathbb{S}_2 = MC(\mathcal{V}, D, C)$.*

Example 3. *A #CSP $MC(\mathcal{V}, D, C)$, $V = \{V_1, V_2, V_3\}$, $D = \{1, 2, 3\}$, $C = \{V_1 < 3, V_2 \neq V_3\}$, can be split into $(\mathbb{S}_1 = \langle \{V_1\}, D, \{V_1 < 3\} \rangle, \mathbb{S}_2 = \langle \{V_2, V_3\}, D, \{V_2 \neq V_3\} \rangle$: $MC(\mathcal{V}, D, C) = \mathbb{S}_1 \cdot \mathbb{S}_2 = 2 \cdot 6 = 12$. Note that V_2 and V_3 are exchangeable.*

Shattering generalizes a split: as in probabilistic lifted inference it can be defined as the repeated application of the splitting operation:

Definition 10. *Given a #CSP $MC(\mathcal{V}, D, C)$ a* shattering*, is a set $\{\mathbb{S}_1, \ldots, \mathbb{S}_n\}$, $\mathbb{S}_i = MC(\mathcal{V}_i, D, C_i)$, such that $MC(\mathcal{V}, D, C) = \prod_{i=1}^{n} \mathbb{S}_i$.*

The multiplication rule holds only if the subproblems $\mathbb{S}_i$ are independent, i.e. a satisfying assignment for variables $\mathcal{V}_i$ on $\mathbb{S}_i$ is a partial assignment for the initial problem $MC(\mathcal{V}, D, C)$ independent from the others. That is, C does not contain any non-trivial constraint relating the value of a variable V_1 to the value of a variable V_2 with $V_1 \in \mathbb{S}_i$, $V_2 \in \mathbb{S}_j$ and $i \neq j$. Nonetheless, the solution of the $\#$CSP can still be expressed as the combination of a set of splits of the problem, by splitting and shattering such constraints along with the problem.

Definition 11. *Given a* CSP *$\langle \mathcal{V}_1 \cup \mathcal{V}_2, D, C \rangle$, $\mathcal{V}_1 \cap \mathcal{V}_2 = \varnothing$, a* constraint split *is a pair (C_1, C_2) such that for each satisfying assignment f_1, f_2 for, respectively, $\langle \mathcal{V}_1, D, C_1 \rangle$ and $\langle \mathcal{V}_2, D, C_2 \rangle$:*

$$f_1(\mathcal{V}_1) \vdash C_1 \wedge f_2(\mathcal{V}_2) \vdash C_2 \implies f_1(\mathcal{V}_1) + f_2(\mathcal{V}_2) \vdash C$$

Example 4. *Consider Example 3: $\mathbb{S}_2$ cannot be further split since the constraint $V_2 \neq V_3$ relates the choices for the two variables. A constraint split for $\{V_2 \neq V_3\}$ is $(\{V_2 = 1\}, \{V_3 \neq 1\})$: any union of satisfying assignments for the subproblems $\langle \{V_2\}, D, \{V_2 = 1\} \rangle$ and $\langle \{V_3\}, D, \{V_3 \neq 1\} \rangle$ satisfies $\mathbb{S}_2$ as well.*

While constraint splits preserve satisfiability, they do not preserve the model count, as Example 4 shows: the set of satisfying assignments for the split is a subset of the set of solutions for $\mathbb{S}_2$. A *constraint shattering* of $\langle \mathcal{V}, D, C \rangle$ is a set of constraint splits $\mathbb{C}$ such that each split corresponds to a partition of the set of satisfying assignments for the original problem.

Definition 12. *Two constraint splits (C_1^1, C_2^1) and (C_1^2, C_2^2) for a problem $\langle \mathcal{V}_1 \cup \mathcal{V}_2, D, C \rangle$, $\mathcal{V}_1 \cap \mathcal{V}_2 = \varnothing$,* overlap *if there exist two satisfying assignment f_1 and f_2 such that $f_1(\mathcal{V}_1) \vdash C_1^1, f_1(\mathcal{V}_2) \vdash C_2^1, f_2(\mathcal{V}_1) \vdash C_1^2, f_2(\mathcal{V}_2) \vdash C_2^2$ and $f_1(\mathcal{V}_1) + f_1(\mathcal{V}_2) = f_2(\mathcal{V}_1) + f_2(\mathcal{V}_2)$.*

Algorithm 1 Combinatorics solver CoSo

```
Precondition: $structure(s, \delta, \phi, D)$, $\delta \in \{\neq, =^X\}$, $\#s = x$, choice constraints $CH$, counting constraints $C$
    $\mathcal{V} = \{V_1, \dots, V_x\}$
    $V_i \leftarrow D, i \in \{1, \dots, x\}$
    if $\delta$ is $\neq$ then: $=$ is $\varnothing$
    else: $\mathbf{d} = \mathbf{d'}$ if $\mathbf{d}$ is a permutation of $\mathbf{d'}$
    $\mathcal{V} \leftarrow$ propagate $CH$
    shatter $\#CSP^{=}$ $MC(\mathcal{V}, \delta, \phi, C)$
```

Definition 13. *Given a* CSP $\langle \mathcal{V}_1 \cup \mathcal{V}_2, D, C\rangle$, $\mathcal{V}_1 \cap \mathcal{V}_2 = \varnothing$, *a constraint shattering* $\mathbb{C}$*, is a set of non-overlapping constraint splits* (C_1, C_2) *such that the union of the satisfying assignments of the splits is equal to* $M(\mathcal{V}_1 \cup \mathcal{V}_2, D, C)$*:*

$$M(\mathcal{V}_1 \cup \mathcal{V}_2, D, C) = \bigcup_{(C_1, C_2) \in \mathbb{C}} M(\mathcal{V}_1, D, C_1) + M(\mathcal{V}_2, D, C_2)$$

Property 3. *Given a* $\#$CSP $MC(\mathcal{V}_1 \cup \mathcal{V}_2, D, C)$ *and a constraint shattering* $\mathbb{C}$ *:*

$$MC(\mathcal{V}_1 \cup \mathcal{V}_2, D, C) = \sum_{(C_1, C_2) \in \mathbb{C}} MC(\mathcal{V}_1, D, C_1) \cdot MC(\mathcal{V}_2, D, C_2)$$

The nature of the shattering $\mathbb{C}$ defining the split constraints depends on the set C, as we show in Section 6 and in the following example:

Example 5. *Consider Example 4: the constraint* $V_2 \neq V_3$ *can be shattered as* $\{(\langle\{V_2\}, D, \{V_2 = i\}\rangle, \langle\{V_3\}, D, \{V_3 \neq i\}\rangle) \mid i \in D\}$ *. Each split* i *has a model count of* 2*, therefore summing out the subproblems from the shattering equals* 6*.*

6 Solving Combinatorial Problems

We show the efficacy of a lifted approach to counting constraint satisfaction problems by implementing a solver, CoSo, based on exchangeability and constraint shattering. We define a set of operators that enable decomposing a combinatorial problem into subproblems where closed-form counting rules are applicable. We consider the cases from CoLa where $\delta \in \{\neq, =^X\}$, to show how distinguishability is taken into account, $\phi \in \{\mathcal{F}_{\text{any}}, \mathcal{F}_{\text{inj}}\}$, to show the reasoning over a traditional global constraint, and counting constraints to show the shattering of a problem with multiple global constraints. The instances of this class of problems (expressed in CoLa) are mapped to a $\#CSP^{=}$ as described in Algorithm 1: variables are domains, instantiated to the universe set; their scope is then restricted with the classical operation of propagation from constraint processing, according to the choice constraints CH . After that, the problem is shattered by *shatter* according to the counting constraints C and the preconditions met by the problem.

The preconditions distinguish between A) exchangeable ($\equiv^{\mathcal{V}}$) and B) non-exchangeable ($\not\equiv^{\mathcal{V}}$) variables. In both cases we consider first 1) counting constraints alone ($|C| \geqslant 1, \phi = \mathcal{F}_{\text{any}}$), then 2) injectivity alone ($\phi = \mathcal{F}_{\text{inj}}, |C| = 0$) and finally 3) the combination of the two. We begin with describing the base cases, where there are no counting constraints ($|C| = 0$). For the problems under investigation, variables are exchangeable when the set of admissible values (domain) is the same.

Base rules. The base cases occur when either all the variables $\mathcal{V}$ are exchangeable with no counting constraints ($\phi = \mathcal{F}_{\text{inj}} \wedge \equiv^{\mathcal{V}}, |C| = 0$), or the variables are independent ($\phi = \mathcal{F}_{\text{any}}, |C| = 0$). We refer to the domain of a set of exchangeable variables $\mathcal{V}$ as $dom(\mathcal{V})$.

Precondition: $\phi = \mathcal{F}_{\text{any}} \vee (\phi = \mathcal{F}_{\text{inj}} \wedge \equiv^{\mathcal{V}}), |C| = 0$
Operator: $\text{BASE}(\mathcal{V}, \delta, \phi, C) =$
counting rule from Table 1 corresponding to (δ, ϕ)*:*
$x = |\mathcal{V}|$ and $y = |dom(\mathcal{V})|$.
Postcondition: $MC(\mathcal{V}, \delta, \phi, C) = \text{BASE}(\mathcal{V}, \delta, \phi, C)$

A. Exchangeable Variables

Exchangeable variables allow to lift the count of solutions.

A1. Counting constraints. Consider a set of variables $\mathcal{V}$, and a counting constraint $c : \#_{\mathcal{V}}\, \varphi = n$. We focus on equality as the other operators derive from it. In order to constrain $\mathcal{V}$ to satisfy c it is sufficient to constrain n variables in $\mathcal{V}$ to belong to the domain formula φ. Here exchangeability and distinguishability come into play: under distinguishability there are $e = \binom{|\mathcal{V}|}{n}$ exchangeable choices of variables to propagate c, otherwise ($=^X$), the choices are indistinguishable ($e = 1$).

Precondition: $\phi = \mathcal{F}_{\text{any}} \wedge \equiv^{\mathcal{V}}, C = \{\#_{\mathcal{V}}\, \varphi = n\}$
Operator: $\text{COUNT}_{\equiv}^{any}(\mathcal{V}, \delta, \phi, C)$
with $\mathcal{V}_{sat}, \mathcal{V}_{unsat}$ **s.t.**:
$|\mathcal{V}_{sat}| = n, |\mathcal{V}_{unsat}| = |\mathcal{V}| - n$
$V = dom(\mathcal{V}) \cap \varphi, V \in \mathcal{V}_{sat}$
$V = dom(\mathcal{V}) \cap \overline{\varphi}, V \in \mathcal{V}_{unsat}$
$e =$ **if** δ **is** $\neq$: $\binom{|\mathcal{V}|}{n}$ **else**: 1
$\mathbb{S}_1 = MC(\mathcal{V}_{sat}, \delta, \phi, \varnothing)$
$\mathbb{S}_2 = MC(\mathcal{V}_{unsat}, \delta, \phi, \varnothing)$
Postcondition:
$\{\mathbb{S}_1, \mathbb{S}_2\}$ is a split for $\langle \mathcal{V}_{sat} \cup \mathcal{V}_{unsat}, \delta, \phi, C\rangle$
$MC(\mathcal{V}, \delta, \phi, C) = e \cdot \mathbb{S}_1 \cdot \mathbb{S}_2$

After propagating c, the variables are no longer exchangeable: when $|C| > 1$ it is sufficient to propagate one of the counting constraints and recursively solve the problem on non-exchangeable variables.

A2. Injectivity constraint. As previously described, the count for $\mathcal{F}_{\text{inj}}$ over exchangeable variables is lifted with a counting rule.

A3. Both constraints. The same approach of propagation of a counting constraint followed by recursion can be adopted when the variables need to be all different.

B. Non-exchangeable Variables

When exchangeability cannot be exploited, we shatter the problem (and the constraints) into subproblems where variables are exchangeable and we exploit Property 3 to compute the model count of the constraint shattering. Consider the equivalence classes induced by the exchangeability relation $\equiv$ over $\mathcal{V}$: $\equiv$ induces a partition $\mathcal{V}/\equiv$ into sets (classes) of exchangeable variables ($[V] = \{W \in \mathcal{V} \mid W \equiv V\}$). Each class $[V]$ is thus a set of exchangeable variables: when $\mathcal{V}$ is not exchangeable $|\mathcal{V}/\equiv| > 1$. In order to shatter the problem we consider a split on a class $[V] \in \mathcal{V}/\equiv$ and the remaining variables in $rest^{\not\equiv}_V = \{V \in [W] | [W] \in \mathcal{V}/\equiv \setminus\{[V]\}\}$: the first subproblem is on exchangeable variables and can be solved accordingly, the second, on $rest^{\not\equiv}_V$, can be further decomposed independently from the split class.

B1. Shattering counting constraints. A counting constraint $c = \#_{\mathcal{V}}\,\varphi = n$ is shattered into the possible combinations of the number of elements belonging to φ in the split class $[V]$ and the other variables in $rest^{\not\equiv}_V$. A shattering $\mathbb{C}_1$ for a problem where $C = \{c\}$ is the following set of splits:

$$\mathbb{C}_1(c) = \{(\#_{\mathcal{V}_1}\,\varphi = i, \#_{\mathcal{V}_2}\,\varphi = n - i) \mid i \in \{0, \ldots, n\}\}$$

Each pair (C^i_1, C^i_2) is a constraint split: $i + n - i = n$, moreover each split i does not overlap with the others since the observed count is a different i (or $n - i$) in each subproblem.

```
Precondition: $\phi = \mathcal{F}_{\text{any}} \wedge \not\equiv^{\mathcal{V}}, C = \{\#_{\mathcal{V}}\,\varphi = n\}$
Operator: $\text{COUNT}^{any}_{\not\equiv}(\mathcal{V}, \delta, \phi, C)$
  let $[V] \in \mathcal{V}/\equiv, i \in \{0, \ldots, n\}$ :
    $C^i_{[V]} = \{\#_{[V]}\,\varphi = i\}$
    $C^i_{rest^{\not\equiv}_V} = \{\#_{rest^{\not\equiv}_V}\,\varphi = n - i\}$
    $\mathbb{S}^i_{[V]} = MC([V], \delta, \phi, C^i_{[V]})$
    $\mathbb{S}^i_{rest^{\not\equiv}_V} = MC(rest^{\not\equiv}_V, \delta, \phi, C^i_{rest^{\not\equiv}_V})$
Postcondition:
  $\{(\mathbb{S}^i_{[V]}, \mathbb{S}^i_{rest^{\not\equiv}_V}) \mid i \in \{0, \ldots, n\}\}$ shatters $\langle \mathcal{V}, \delta, \phi, C\rangle$
```

Example 6. *Consider Example 2: let $\mathcal{V} = \{V_1, V_2, V_3, V_4\}$: after the propagation of the choice constraints $[V_1] = \{V_1, V_2, V_4\}$, $dom([V_1]) = student$ and $rest^{\not\equiv}_{V_1} = [V_3] = \{V_3\}$, $dom([V_3]) = dutch$. The shattering of $\#_s dutch \geqslant 2$ is: $\{(\#_{[V_1]}dutch = 0, \#_{[V_3]}dutch \geqslant 2), (\#_{[V_1]}dutch = 1, \#_{[V_3]}dutch \geqslant 1), (\#_{[V_1]}dutch \geqslant 2, \#_{[V_3]}dutch \geqslant 0)\}$.*

When more than one counting constraint has to be split between two subproblems then it is necessary to take into account the combinations of each shattering of the different counting constraints. The generalized shattering is the following cartesian product: $\mathbb{C}_n(C) = \times_{c \in C}\,\mathbb{C}_1(c)$.

```
Precondition: $\phi = \mathcal{F}_{\text{any}} \wedge \not\equiv^{\mathcal{V}}, |C| > 1$
Operator: $\text{COUNT}^{any}_{\not\equiv}(\mathcal{V}, \delta, \phi, C)$
  let $[V] \in \mathcal{V}/\equiv$, $\mathbf{c} \in \mathbb{C}_n(C)$:
    $C^{\mathbf{c}}_{[V]} = \{c_1 \mid \mathbf{c}_i = (c_1, c_2),\ i \in \{1, \ldots, |\mathbf{c}|\}\}$
    $C^{\mathbf{c}}_{rest^{\not\equiv}_V} = \{c_2 \mid \mathbf{c}_i = (c_1, c_2),\ i \in \{1, \ldots, |\mathbf{c}|\}\}$
    $\mathbb{S}^{\mathbf{c}}_{[V]} = MC([V], \delta, \phi, C^{\mathbf{c}}_{[V]})$
    $\mathbb{S}^{\mathbf{c}}_{rest^{\not\equiv}_V} = MC(rest^{\not\equiv}_V, \delta, \phi, C^{\mathbf{c}}_{rest^{\not\equiv}_V})$
Postcondition:
  $\{(\mathbb{S}^{\mathbf{c}}_{[V]}, \mathbb{S}^{\mathbf{c}}_{rest^{\not\equiv}_V}) \mid \mathbf{c} \in \mathbb{C}_n\}$ shatters $\langle \mathcal{V}, \delta, \phi, C\rangle$
```

B2. Shattering injectivity. In order to split the injectivity constraint into independent subproblems we consider that for each class $[W] \in rest^{\not\equiv}_V$ the subproblem over $[V]$ could be solved by choosing some of the elements in W, $[V] \cap [W] \neq \varnothing$ (i.e. $french$ and $dutch$ in Problem 2). Therefore, a non-overlapping shattering fixes for each split the number of elements from $rest^{\not\equiv}_V$ used in the subproblem on $[V]$, and removes those elements from the domains of the variables in $rest^{\not\equiv}_V$. When $|rest^{\not\equiv}_V| > 1$ an element can belong to none, some or all the domains $\varphi \in rest^{\not\equiv}_V$. We express such cases as set-combinations $\Phi = \bigcap\{c^n \mid c \in rest^{\not\equiv}_V, n \in \{-1, 1\}\}$ where $c^1 = c$ and $c^{-1} = \overline{c}$. If $|rest^{\not\equiv}_V| = 1$ then $\Phi = \{dom(rest^{\not\equiv}_V)\}$. We can *partition* the variables in $[V]$ according to those who belong to the same set φ: since they are *exchangeable* it is sufficient to fix the number for each set-combination and not the specific identities. The possible distribution of the variables over the set-combinations in Φ are thus the integer partitions $P_{|[V]|}(|\Phi|)$. An *histogram* $h \in \Phi \times P_{|[V]|}(|\Phi|)$ corresponds to a set of counting constraints $C(h) = \{\#_{[V]}\varphi = i \mid (\varphi, i) \in h\}$. The problem on $rest^{\not\equiv}_V$ is solved by filtering out elements corresponding to the histogram h from the variables in $rest^{\not\equiv}_V$ ($rest^{\not\equiv}_V/h$). The shattering for the injectivity constraint is:

$$\mathbb{C}_{inj} = \{(C(h), rest^{\not\equiv}_V/h) \mid h \in \Phi \times P_{|[V]|}(|\Phi|))\}$$

```
Precondition: $\phi = \mathcal{F}_{\text{inj}} \wedge \not\equiv^{\mathcal{V}}, |C| = 0$
Operator: $\text{ALLDIFF}_{\not\equiv}(\mathcal{V}, \delta, \phi, C) =$
  let $[V] \in \mathcal{V}/\equiv, (C(h), rest^{\not\equiv}_V/h) \in \mathbb{C}_{inj}$ :
    $C^h_{[V]} = C(h)$
    $C^h_{rest^{\not\equiv}_V} = rest^{\not\equiv}_V/h$
    $\mathbb{S}^h_{[V]} = MC([V], \delta, \phi, C^h_{[V]})$
    $\mathbb{S}^h_{rest^{\not\equiv}_V} = MC(rest^{\not\equiv}_V, \delta, \phi, C^h_{rest^{\not\equiv}_V})$
Postcondition:
  $\{(\mathbb{S}^h_{[V]}, \mathbb{S}^h_{rest^{\not\equiv}_V}) \mid h \in \Phi \times P_{|[V]|}(|\Phi|)\}\}$ shatters $\langle \mathcal{V}, \delta, \phi, C\rangle$
```

Example 7. *Following Example 6 we split injectivity between $[V_1]$ and $[V_3]$: in $[V_1]$ up to 3 Dutch-speaking students can be chosen, hence the shattering is the following: $\{(\#_{[V_1]}dutch = i, [V_3]/\{(dutch, i)\} \mid i \in \{1, \ldots, 3\})\}$*

B3. Shattering both. When both injectivity and counting constraints need to be split ($\phi = \mathcal{F}_{\text{inj}}, |C| > 1$) the cases from the two shatterings are combined together. We give an informal description since the shattering cases are the same for injectivity and counting constraints as previously described, and the two are coupled in a shattering by means of a Cartesian product $\mathbb{C}_1 \times \mathbb{C}_{inj}$ or $\mathbb{C}_n \times \mathbb{C}_{inj}$.

Example 8. *The constraint shattering for Example 2 is: $\{(\#_{[V_1]}dutch = 1, [V_3]/\{(dutch, 1)\}), (\{\#_{[V_1]}dutch \geqslant 2, \#_{[V_1]}dutch = 2\}, [V_3]/\{(dutch, 2)\}), (\{\#_{[V_1]}dutch \geqslant 2, \#_{[V_1]}dutch = 3\}, [V_3]/\{(dutch, 3)\})\}$ Note that many shattering combinations are inconsistent therefore ignored.*

7 Experiments

The goal of the experiments is to provide both a qualitative (language) and quantitative (solving time) comparison of the different approaches to solving combinatorial problems. We focus in particular on four aspects that the previous sections suggest to be the most relevant when reasoning over combinatorial problems: defining and reasoning over domains for variables, set manipulation, the support for global constraints and a domain lifted procedure to compute the model count (Table 2). We compare to aProblog as the representative of the propositional model counting tools developed in the context of probabilistic inference, to Forclift (den Broeck et al. 2011) and GC-FOVE (Taghipour et al. 2012) as the lifted version of such tools, and to MiniZinc (Nethercote et al. 2007) and Gecode (Schulte, Tack, and Lagerkvist 2010) for a constraint-based modelling and resolution.

Domains. Domains are not natively supported by aProblog and GC-FOVE. aProbLog allows to express general-purpose first-order logic programs: we can encode a variable and a domain $\{c_1, \dots, c_n\}$ as a unary predicate v by means of annotated disjunctions, that is, a statement of the form $w_1 :: v(c_1); \dots w_n :: v(c_n)$ where w_i is a weight (1 in this context) expressing that the validity of $\{v(c_1), \dots, v(c_n)\}$ is mutually exclusive.

Sets. Set manipulation is not supported by the probabilistic inference frameworks. A set of variables can be encoded by means of symmetry breaking, i.e. by fixing a permutation of the variables $V_1, \dots, V_n$ with a rule $V_1 < \dots < V_n$. However, as we mentioned in Section 7, the constraint language of FOL-DC is limited, and the relation "<" is explicitly forbidden in the compilation rules.

Global constraints. Both aProblog and Forclift support only binary constraints, failing again to incorporate distinctive aspects from constraint programming.

Lifted. Forclift and GC-FOVE support lifted reasoning. To the best of our knowledge there are no #CSP solvers supporting a general constraint programming language.

Implementations

Unfortunately Forclift's input language is Markov Logic Networks and GC-FOVE's language is based on BLOG (Milch et al. 2008), which are designed to model joint distributions of random variables and thus are not suitable to encode combinatorial problems. This limits the comparison of the solving performances to the grounding-based approaches of aProbLog and MiniZinc 2.4.3 coupled with Gecode 6.1.1. We compared the tools on problems with and without constraints (Figure 2), excluding aProblog from the first tests given the lack of constraint support. We tested counting sequences (i.e. Problem 3), permutations and subsets of increasing size (between 8 and 24 for subsets, between 4 and 15 otherwise) and of increasing universe size (between 10 and 30 for subsets, between 5 and 15 otherwise). The x axis shows the number of solutions in log-scale, the y axis the solving time. We set a timeout of 240 seconds denoted by the letter x in Figure 2.

As expected, the enumeration-based CSP approach degrades exponentially as the number of solutions increases. aProblog's knowledge compilation approach performs poorly for problems with few solutions but bigger structure sizes, since the size of the compiled data structure depends on the number of variables (annotated disjunctions). For instance, the number of 10-subsets of 14 elements is equal to the number of 4-subsets (1001). On the structure with 4 variables, aProblog counts the subsets in $60s$, but on 10 variables it times out.

	Domains	Sets	Global const.	Lifted
aProbLog	x	x	x	x
Forclift	✓	x	x	✓
GC-FOVE	x	x	✓	✓
CSP	✓	✓	✓	x
CoSo	✓	✓	✓	✓

Table 2: Frameworks comparison

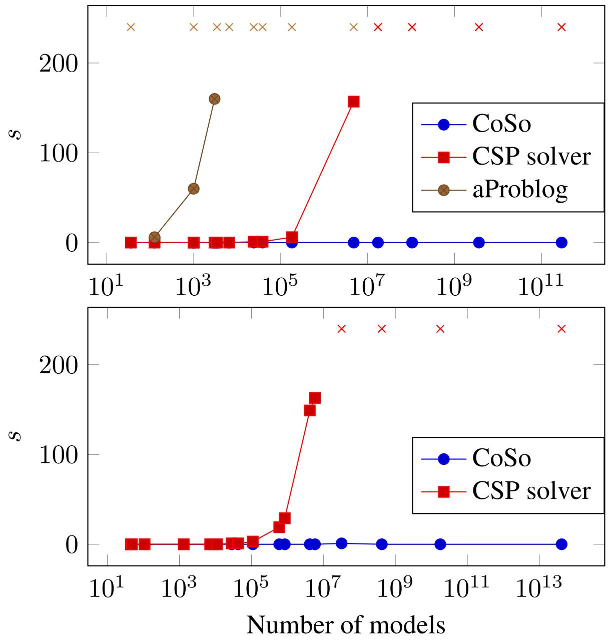

Figure 2: Comparison on combinatorial problems with (bottom) and without (top) constraints.

8 Conclusion

We presented a framework of combinatorial problems where the existing algorithms and tools fail to provide an efficient solving strategy. We addressed this limitation by adapting lifted reasoning techniques from probabilistic inference to a more expressive constraint language. We introduced a declarative language, CoLa, to model combinatorics problems and constraints and presented new lifted inference concepts for counting constraint satisfaction problems. Finally, we proved the validity of our approach by implementing a solver, CoSo, for a class of combinatorial problems on which the existing approaches are outperformed. Future work will focus on expanding CoSo with more constraints and shattering operators, and on combining the novel constraint manipulation techniques with lifted probabilistic inference tools.

1. $u = \{1, \dots, 15\}$.
2. $d_1 = \{7, \dots, 13\}$.
3. $d_2 = \{5, \dots, 15\}$.
4. $structure(s, \neq, \mathcal{F}_{\text{any}}, u)$.
5. $\#s = 4$.
6. $pos(s, 4, d_2 \cap \overline{d}_1)$.
7. $\#_s\, d_2 > 2$.

Problem 3: Example of test instance

9 Acknowledgements

References

Cadoli, M.; and Donini, F. M. 1997. A Survey on Knowledge Compilation. *AI Commun.* 10(3-4): 137–150. URL http://content.iospress.com/articles/ai-communications/aic133.

Chesani, F.; Mello, P.; and Milano, M. 2017. Solving Mathematical Puzzles: A Challenging Competition for AI. *AI Mag.* 38(3): 83–96. doi:10.1609/aimag.v38i3.2736. URL https://doi.org/10.1609/aimag.v38i3.2736.

de Salvo Braz, R.; Amir, E.; and Roth, D. 2005. Lifted First-Order Probabilistic Inference. In Kaelbling, L. P.; and Saffiotti, A., eds., *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, 1319–1325. Professional Book Center. ISBN 0938075934. URL http://ijcai.org/Proceedings/05/Papers/1548.pdf.

den Broeck, G. V.; Meert, W.; and Darwiche, A. 2014. Skolemization for Weighted First-Order Model Counting. In Baral, C.; Giacomo, G. D.; and Eiter, T., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014*. AAAI Press. ISBN 978-1-57735-657-8. URL http://www.aaai.org/ocs/index.php/KR/KR14/paper/view/8012.

den Broeck, G. V.; Taghipour, N.; Meert, W.; Davis, J.; and De Raedt, L. 2011. Lifted Probabilistic Inference by First-Order Knowledge Compilation. In Walsh, T., ed., *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, 2178–2185. IJCAI/AAAI. ISBN 978-1-57735-516-8. doi:10.5591/978-1-57735-516-8/IJCAI11-363. URL https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-363.

Dries, A.; Kimmig, A.; Davis, J.; Belle, V.; and De Raedt, L. 2017. Solving Probability Problems in Natural Language. In Sierra, C., ed., *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, 3981–3987. ijcai.org. ISBN 978-0-9992411-0-3. doi:10.24963/ijcai.2017/556. URL https://doi.org/10.24963/ijcai.2017/556.

Kazemi, S. M.; Kimmig, A.; den Broeck, G. V.; and Poole, D. 2016. New Liftable Classes for First-Order Probabilistic Inference. In Lee, D. D.; Sugiyama, M.; von Luxburg, U.; Guyon, I.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 3117–3125. URL http://papers.nips.cc/paper/6603-new-liftable-classes-for-first-order-probabilistic-inference.

Kimmig, A.; den Broeck, G. V.; and De Raedt, L. 2017. Algebraic model counting. *J. Appl. Log.* 22: 46–62. doi:10.1016/j.jal.2016.11.031. URL https://doi.org/10.1016/j.jal.2016.11.031.

Milch, B.; Zettlemoyer, L. S.; Kersting, K.; Haimes, M.; and Kaelbling, L. P. 2008. Lifted Probabilistic Inference with Counting Formulas. In Fox, D.; and Gomes, C. P., eds., *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, 1062–1068. AAAI Press. ISBN 978-1-57735-368-3. URL http://www.aaai.org/Library/AAAI/2008/aaai08-168.php.

Mitra, A.; and Baral, C. 2016. Learning To Use Formulas To Solve Simple Arithmetic Problems. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics. ISBN 978-1-945626-00-5. doi:10.18653/v1/p16-1202. URL https://doi.org/10.18653/v1/p16-1202.

Nethercote, N.; Stuckey, P. J.; Becket, R.; Brand, S.; Duck, G. J.; and Tack, G. 2007. MiniZinc: Towards a Standard CP Modelling Language. In Bessiere, C., ed., *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, volume 4741 of *Lecture Notes in Computer Science*, 529–543. Springer. ISBN 978-3-540-74969-1. doi:10.1007/978-3-540-74970-7_38. URL https://doi.org/10.1007/978-3-540-74970-7_38.

Niepert, M.; and den Broeck, G. V. 2014. Tractability through Exchangeability: A New Perspective on Efficient Probabilistic Inference. In Brodley, C. E.; and Stone, P., eds., *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*, 2467–2475. AAAI Press. ISBN 978-1-57735-661-5. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8615.

Poole, D. 2003. First-order probabilistic inference. In Gottlob, G.; and Walsh, T., eds., *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, 985–991. Morgan Kaufmann. URL http://ijcai.org/Proceedings/03/Papers/142.pdf.

Régin, J. 1996. Generalized Arc Consistency for Global Cardinality Constraint. In Clancey, W. J.; and Weld, D. S., eds., *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, USA, August 4-8, 1996, Volume 1*, 209–215. AAAI Press / The MIT Press. ISBN 0-262-51091-X. URL http://www.aaai.org/Library/AAAI/1996/aaai96-031.php.

Roy, S.; and Roth, D. 2018. Mapping to Declarative Knowledge for Word Problem Solving. *Trans. Assoc. Comput. Linguistics* 6: 159–172. URL https://transacl.org/ojs/index.php/tacl/article/view/1319.

Régin, J.-C. 2010. *Global Constraints: A Survey*, 63–134. doi:10.1007/978-1-4419-1644-0_3.

Schulte, C.; Tack, G.; and Lagerkvist, M. Z. 2010. Modeling and programming with gecode. *Schulte, Christian and Tack, Guido and Lagerkvist, Mikael* 1.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529: 484–503. URL http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html.

Stanley, R. 2012. *Enumerative combinatorics*. New York: Cambridge University Press. ISBN 9781107602625.

Taghipour, N.; Fierens, D.; Davis, J.; and Blockeel, H. 2012. Lifted Variable Elimination with Arbitrary Constraints. In Lawrence, N. D.; and Girolami, M. A., eds., *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2012, La Palma, Canary Islands, Spain, April 21-23, 2012*, volume 22 of *JMLR Proceedings*, 1194–1202. JMLR.org. URL http://proceedings.mlr.press/v22/taghipour12.html.

Valiant, L. G. 1979. The Complexity of Computing the Permanent. *Theor. Comput. Sci.* 8: 189–201. doi:10.1016/0304-3975(79)90044-6. URL https://doi.org/10.1016/0304-3975(79)90044-6.

Zhang, N. L.; and Poole, D. 1994. A simple approach to Bayesian network computations.