On Discovering Interesting Combinatorial Integer Sequences

Martin Svatoš[1] , **Peter Jung**[1] , **Jan Tóth**[1] , **Yuyi Wang**[2,3] and
Ondřej Kuželka[1]
[1]Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic
[2]CRRC Zhuzhou Institute, China
[3]ETH Zurich, Switzerland
{svatoma1, jungpete, tothjan2, ondrej.kuzelka}@fel.cvut.cz, yuyiwang920@gmail.com

Abstract

We study the problem of generating *interesting* integer sequences with a combinatorial interpretation. For this we introduce a two-step approach. In the first step, we generate first-order logic sentences which define some combinatorial objects, e.g., undirected graphs, permutations, matchings etc. In the second step, we use algorithms for lifted first-order model counting to generate integer sequences that count the objects encoded by the first-order logic formulas generated in the first step. For instance, if the first-order sentence defines permutations then the generated integer sequence is the sequence of factorial numbers $n!$. We demonstrate that our approach is able to generate interesting new sequences by showing that a non-negligible fraction of the automatically generated sequences can actually be found in the Online Encyclopaedia of Integer Sequences (OEIS) while generating many other similar sequences which are not present in OEIS and which are potentially interesting. A key technical contribution of our work is the method for generation of first-order logic sentences which is able to drastically prune the space of sentences by discarding large fraction of sentences which would lead to redundant integer sequences.

1 Introduction

In this paper we are interested in integer sequences. As its name suggests, an *integer sequence* is a sequence of integers $a_0, a_1, a_2, \ldots$, where $a_i \in \mathbb{Z}$ for all $i \in \mathbb{N}$. Integer sequences are fundamental mathematical objects that appear almost everywhere in mathematics, ranging from enumerative combinatorics, where they count objects with certain properties, to mathematical analysis, where they define functions by means of Taylor series, and in many other areas as well. There is even an encyclopedia of them, called Online Encyclopedia of Integer Sequences (OEIS),[1] whose offline predecessor was established in 1964 by Neil Sloane [OEIS Foundation Inc., 2023]. It contains more than 359k integer sequences, as of January 2023. OEIS contains sequences that are of interest to professional or amateur mathematicians.

A typical mode of use of the OEIS database is as follows. Say, you work on a combinatorial problem, counting undirected graphs on n vertices that have certain property that you care about, e.g., having all vertex-degrees equal to 3. You manage to compute the numbers of these graphs for several small values of n and you start wondering if someone did not study the same sequence of numbers. So you take the values you computed and insert them into the search box on the OEIS homepage and hit search. After that you receive all hits into OEIS and if you are lucky, one of them will tell you something interesting about your problem—maybe somebody has already solved it or at least computed more elements of the sequence.[2]

How do sequences get into OEIS? Sequences that are deemed *interesting* are manually submitted to OEIS by users. Here, what is *interesting* is obviously subjective to a large extent. However, this is also a limitation of OEIS—the first person to study a certain sequence will not get much help by looking it up in OEIS. Many quite natural sequences are not contained in OEIS. For instance, as observed by [Barvínek *et al.*, 2021], it contains sequences counting 2-regular graphs properly colored by 2 colors, but not 2-regular graphs properly colored by 3 colors. There are many similar examples of interesting sequences missing from OEIS, which might be potentially useful for some users. This is also the motivation for the work we present here in which we develop an automated method for discovering arguably *interesting* integer sequences.

We focus on combinatorial sequences, i.e., sequences which count objects of size n that have some given property, which are the subject of interest of enumerative combinatorics [Stanley, 1986]. Examples of such combinatorial sequences include sequences counting: subsets of an n-element set, graphs on n vertices, connected graphs on n vertices, trees on n vertices, permutations on n elements without fixpoints etc. In particular, we focus on combinatorial sequences

[1]https://oeis.org, for a popular account of the place of OEIS in mathematics, we also refer to the article in Quanta Magazine: https://www.quantamagazine.org/neil-sloane-connoisseur-of-number-sequences-20150806/.

[2]For instance, for undirected graphs with all vertex degrees equal to 3, one of the hits in OEIS would be sequence A002829: Number of trivalent (or cubic) labeled graphs with $2n$ nodes.

of structures *that can be described using a first-order logic sentence*.

There are several advantages of working with combinatorial enumeration problems expressed in first-order logic. First, even though it may sometimes require some effort, the first-order logic sentences can be interpreted by the human users. For instance, the sentence $\forall x\ \neg R(x, x) \wedge \forall x \forall y\ R(x, y) \Rightarrow R(y, x)$ can be interpreted as encoding undirected graphs without loops. Second, despite the fact that counting the models of first-order logic sentences is generally intractable [Beame *et al.*, 2015], there are well-characterized non-trivial fragments of first-order logic for which counting is tractable [Van den Broeck, 2011; Van den Broeck *et al.*, 2014; Kuželka, 2021] with fast implementations available [Van den Broeck, 2011; van Bremen and Kuželka, 2021].[3] This means that we are able to compute the respective combinatorial sequences fast.[4]

Our method has two stages. First, we generate first-order logic sentences from a tractable fragment. Second, we compute sequences for each of the generated sentences and filter out sentences which give rise to redundant sequences. It turns out that the first step is critical. As we demonstrate experimentally later in this paper, if we generated sentences naively, i.e., if we attempted to generate all sentences of length at most k that differ syntactically, we would have to compute such huge numbers of sequences, most of them redundant, that we would never be able to get to the interesting ones. In the present paper, we therefore focus mostly on describing the sentence-generating component of our system.

The rest of this paper is structured as follows. Section 2 describes the preliminaries from first-order logic. Section 3 describes our approach to construct a database of sentences and the respective integer sequences, which is evaluated in Section 4. The paper ends with related work in Section 5 and conclusion in Section 6.[5]

2 Preliminaries

We work with a function-free subset of first-order logic. The language is defined by a finite set of *constants* Δ, a finite set of *variables* $\mathcal{V}$ and a finite set of *predicates* $\mathcal{P}$. An *atom* has the form $P(t_1, t_2, \ldots, t_k)$ where $P \in \mathcal{P}$ and $t_i \in \Delta \cup \mathcal{V}$. A *literal* is an atom or its negation. A *formula* is a literal. More complex formulas may be formed from existing formulas by logical connectives, or by surrounding them with a universal ($\forall x$) or an existential ($\exists x$) quantifier where $x \in \mathcal{V}$. A variable x in a formula is called *free* if the formula contains no quantification over x. A formula is called a *sentence* if it contains no free variables. A formula is called *ground* if it contains no variables.

[3]https://github.com/UCLA-StarAI/Forclift, https://www.comp.nus.edu.sg/~tvanbr/software/fastwfomc.tar.gz

[4]Computational complexity of integer sequences that count combinatorial objects is an active research direction in enumerative combinatorics, see, e.g., [Pak, 2018].

[5]This paper is accompanied by a technical report available at https://arxiv.org/abs/2302.04606 and codes available at https://github.com/martinsvat/SentenceFinder.

As is customary in computer science, we adopt the *Herbrand semantics* [Hinrichs and Genesereth, 2006] with a finite domain. We use HB to denote the *Herbrand base*, i.e., the set all ground atoms. We use ω to denote a *possible world*, i.e., any subset of HB. Elements of a possible world are assumed to be true, all others are assumed to be false.

2.1 Weighted First-Order Model Counting

To compute the combinatorial integer sequences, we make use of the weighted first-order model counting (WFOMC) problem [Van den Broeck *et al.*, 2011].

Definition 1. *(Weighted First-Order Model Counting) Let ϕ be a sentence over some relational language $\mathcal{L}$. Let* HB *denote the Herbrand base of $\mathcal{L}$ over some domain of size $n \in \mathbb{N}$. Let $\mathcal{P}$ be the set of the predicates of the language $\mathcal{L}$ and let* $\mathsf{pred} : \mathsf{HB} \mapsto \mathcal{P}$ *map each atom to its corresponding predicate symbol. Let $w : \mathcal{P} \mapsto \mathbb{R}$ and $\overline{w} : \mathcal{P} \mapsto \mathbb{R}$ be a pair of weightings assigning a positive and a negative weight to each predicate in $\mathcal{L}$. We define* $\mathsf{WFOMC}(\phi, n, w, \overline{w}) =$

$$\sum_{\omega \subseteq \mathsf{HB} : \omega \models \phi} \prod_{l \in \omega} w(\mathsf{pred}(l)) \prod_{l \in \mathsf{HB} \setminus \omega} \overline{w}(\mathsf{pred}(l)).$$

Example 1. *Consider the sentence*

$$\phi = \forall x\ \neg E(x, x)$$

and the weights $w(E) = \overline{w}(E) = 1$. Since all the weights are unitary, we simply count the number of models of ϕ. We can interpret the sentence as follows: Each constant of the language is a vertex. Each atom $E(A, B) \in$ HB with $A, B \in \Delta$ denotes an edge from A to B. Furthermore, the sentence prohibits reflexive atoms, i.e, loops. Overall, the models of ϕ will be all directed graphs without loops on n vertices. Hence, we obtain

$$\mathsf{WFOMC}(\phi, n, w, \overline{w}) = 2^{n^2 - n}.$$

Example 2. *Consider the sentence*

$$\phi = \exists x\ Heads(x)$$

and the weights $w(Heads) = 4, \overline{w}(Heads) = 1$. Now, we can consider each domain element to be the result of a coin flip. The sentence requires that there is at least one coin flip with the value of "heads" (there exists a constant $A \in \Delta$ such that $Heads(A)$ is an element of the model). Suppose we have $i > 0$ "heads" in the model. Then, the model's weight will be $4^i \cdot 1^{n-i} = 4^i$ and there will be $\binom{n}{i}$ such models. Therefore,

$$\mathsf{WFOMC}(\phi, n, w, \overline{w}) = \sum_{i=1}^{n} 4^i \cdot \binom{n}{i} = 5^n - 1.$$

2.2 Tractable Language

In order to make our calculations tractable, we limit the number of variables in each sentence to at most two. Such language is known as $\mathbf{FO}^2$ and it allows computing WFOMC in time polynomial in the domain size [Van den Broeck, 2011; Van den Broeck *et al.*, 2014].

We further add *counting quantifiers* to our syntactic constructs. Counting quantifiers are a generalization of the traditional existential quantifier. For a variable $x \in \mathcal{V}$, we allow usage of a quantifier of the form $\exists^{\bowtie k}x$, where $\bowtie \in \{ \leq, =, \geq \}$ and $k \in \mathbb{N}$. Satisfaction of formulas with counting quantifiers is defined naturally. For example, $\exists^{=k}x\ \psi(x)$ is satisfied in ω if there are exactly k constants $\{ A_1, A_2, \ldots, A_k \} \subseteq \Delta$ such that $\omega \models \psi(A_i)$ if and only if $1 \leq i \leq k$. Such language is known as $\mathbf{C}^2$ and it still permits WFOMC computation time polynomial in the domain size [Kuželka, 2021].

Handling counting quantifiers leads to some weights being symbolic.[6] Although, we defined WFOMC only for real-valued weights, the extension to (multivariate) polynomials is natural and does not break anything.

3 Constructing the Sequence Database

Our aim is to build a database consisting of first-order logic sentences and the respective integer sequences that are generated by these sentences. We do not want the database to be exhaustive in terms of sentences. For any integer sequence, there may be many sentences that generate it[7] and we want only one sentence per integer sequence. If there are multiple sentences that generate the same integer sequence, we call them *redundant*. We generally try to avoid generating redundant sentences.

The database is constructed in two steps. In the first step, we generate first-order logic sentences and in the second step we compute the integer sequences that count the models of these sentences. In this section, we describe these two steps in the reverse order. First we describe how the sequences, which we will call *combinatorial spectra*, are computed from first-order logic sentences, which can be done using existing lifted inference methods. Then we describe our novel sentence-generation algorithm which strives to generate as few redundant sentences as possible.

3.1 Computing the Integer Sequences

Given a first-order logic sentence, we need to compute a number sequence such that its k-th member is the model count of a first-order logic sentence on the domain of size k. The set of domain sizes, for which the sequence member would be non-zero is called a *spectrum* of the sentence, i.e., the *spectrum* of a logical sentence ϕ is the set of natural numbers occurring as size of some finite model of ϕ [Börger *et al.*, 2001]. Since the sequence that we seek builds, in some sense, on top of the spectrum, and since the sequence can also be described as the result of the combinatorial interpretation of the original sentence, we call the sequence *combinatorial spectrum* of the sentence.

Definition 2 (Combinatorial Spectrum)**.** *The combinatorial spectrum of a logical sentence ϕ, denoted as $\mathfrak{S}(\phi)$, is a sequence of model counts of ϕ on finite domains of sizes taking on values* $1, 2, 3, 4, \ldots$

[6]In [Kuželka, 2021], symbolic weights are not used explicitly. The author uses polynomial interpolation instead, but that is for all our purposes in this paper equivalent to using symbolic weights.

[7]Trivially, if we allowed arbitrary predicate names, we could even say that there are infinitely many sentences that generate the same sequence.

Example 3. *Consider again the sentence*

$$\phi = \exists x\ Heads(x).$$

Then, all subsets of HB *are a model of* ϕ *except for the empty set. Hence, for a domain of size* n*, there will be* $2^n - 1$ *models, i.e,*

$$\mathfrak{S}(\phi) = 1, 3, 7, 15, \ldots$$

Combinatorial spectra can be computed using a WFOMC algorithm. In this work we use our implementation of the algorithm from [van Bremen and Kuželka, 2021], which is a state-of-the-art algorithm running in time polynomial in n for $\mathbf{FO}^2$. We use it together with our implementation of the reductions from [Kuželka, 2021] which allow us to compute spectra of any $\mathbf{C}^2$ sentence in time polynomial in n.

3.2 Generating the First-Order Logic Sentences

In general, we aim to generate sentences that have the following syntactic form:[8]

$$\bigwedge_{i=1}^{M} Q_{i,1}x\ Q_{i,2}y\ \Phi_i(x,y) \wedge \bigwedge_{i=1}^{M'} Q_i x\ \Phi_i(x) \qquad (1)$$

where $Q_i, Q_{i,1}, Q_{i,2} \in \{\forall, \exists, \exists^{=1}, \ldots, \exists^{=K}\}$, each $\Phi_i(x,y)$ is a quantifier-free disjunction of literals containing only the logical variables x and y and, similarly, each $\Phi_i(x)$ is a quantifier-free disjunction of literals containing only the logical variable x. The integers K, M and M' are parameters.

Examples of sentences that have the form (1) are:

- $\forall x \exists^{=1} y\ R(x,y) \wedge \forall x \exists^{=1} y\ R(y,x)$,
- $\forall x\ \neg R(x,x) \wedge \forall x \forall y\ \neg R(x,y) \vee R(y,x)$.

Here the first sentence defines bijections (i.e., permutations) and the second sentence defines undirected graphs without loops.

Note 1. *We will slightly abuse terminology and use the term* clause *for the quantified disjunctions of the form* $Q_{i,1}xQ_{i,2}y\ \Phi_i(x,y)$ *and* $Q_j x\ \Phi_j(x)$*, even though the term* clause *is normally reserved only for universally quantified disjunctions.*

Do We Cover All of $\mathbf{C}^2$?

A natural question to ask is: Do we get all possible combinatorial spectra of $\mathbf{C}^2$ sentences if we restrict ourselves to sentences in the form of (1)? The answer seems to be negative, as we explain next, but it hardly matters in our opinion because the task that we set for ourselves in this paper is not to generate all combinatorial sequences of $\mathbf{C}^2$ sentences—this would

[8]There are at least two Scott normal forms for $\mathbf{C}^2$ appearing in the literature [Grädel and Otto, 1999; Pratt-Hartmann, 2009], which would allow us to use less quantifier prefixes. However, these normal forms were not designed for combinatorial counting—they were designed only to guarantee equisatisfiability of $\mathbf{C}^2$ sentences and their normal forms and they do not guarantee *combinatorial equivalence*. That is why they would not be directly useful for us in this paper.

not be feasible anyways because the number of different integer sequences generated as spectra of $\mathbf{C}^2$ sentences is infinite.[9] Instead, what we want to achieve is to generate as many *interesting* integer sequences as possible within a limited time budget. Of course, here what is *interesting* does not have a crisp definition and we use a heuristic. Roughly, a sequence is "interesting" for us if it counts some combinatorial structures that a short sentence in first-order logic can describe ("Occam's razor for interestingness") and which cannot be trivially decomposed into simpler combinatorial sequences.

Now we briefly explain why sentences of the form (1) do not guarantee that we would be able to find all $\mathbf{C}^2$ combinatorial spectra. First of all, we cannot rely on normal forms from [Grädel and Otto, 1999; Pratt-Hartmann, 2009] because those were not designed to preserve model counts. While the transformation presented in [Kuželka, 2021] allows one to reduce the computation of model counts of any $\mathbf{C}^2$ sentence to a computation with sentences that are in the form of (1), it requires some of the predicates to have negative weights. We do not allow negative weights in the generated sentences because they make the post-hoc combinatorial explanation of the sentences significantly more difficult.

Traversing the Sentence Space

We use a standard breadth-first search algorithm to traverse the space of $\mathbf{C}^2$ sentences. The algorithm starts with the empty sentence. In each layer of the search tree it generates all possible sentences that can be obtained by adding a literal to one of the sentences generated in the previous layer. The literal may be added into an existing clause or it can be added to the sentence as a new clause, in which case it also needs to be prefixed with quantifiers.

Example 4. *Suppose we have the sentence $\varphi = \forall x \exists y\ R(x,y)$, which we want to extend. Suppose also that the only predicate in our language is the binary predicate R and that the only allowed quantifiers are $\forall$ and $\exists$ (for simplicity). To extend φ, the first option we have is to add a new R-literal to the clause $\forall x \exists y\ R(x,y)$. There are 8 ways to do this resulting in the following sentences: $\varphi_1 = \forall x \exists y\ (R(x,y) \lor R(x,x))$, $\varphi_2 = \forall x \exists y\ (R(x,y) \lor R(x,y))$, $\varphi_3 = \forall x \exists y\ (R(x,y) \lor R(y,x))$, $\varphi_4 = \forall x \exists y\ (R(x,y) \lor R(y,y))$, $\varphi_5 = \forall x \exists y\ (R(x,y) \lor \neg R(x,x))$, $\varphi_6 = \forall x \exists y\ (R(x,y) \lor \neg R(x,y))$, $\varphi_7 = \forall x \exists y\ (R(x,y) \lor \neg R(y,x))$, $\varphi_8 = \forall x \exists y\ (R(x,y) \lor \neg R(y,y))$. The second option is to create a new single-literal clause and add it to φ. In this case we have the following: $\varphi_9 = \forall x \exists y\ R(x,y) \land \forall x\ R(x,x)$, $\varphi_{10} = \forall x \exists y\ R(x,y) \land \exists x\ R(x,x)$, $\varphi_{11} = \forall x \exists y\ R(x,y) \land \forall x\ \neg R(x,x)$, $\varphi_{12} = \forall x \exists y\ R(x,y) \land \forall x\ \neg R(x,x)$, and then sentences of one of the following types: $\forall x \exists y\ R(x,y) \land Q_1 x Q_2 y\ R(x,y)$, $\forall x \exists y\ R(x,y) \land Q_1 x Q_2 y\ R(y,x)$, $\forall x \exists y\ R(x,y) \land Q_1 x Q_2 y\ \neg R(x,y)$, and $\forall x \exists y\ R(x,y) \land Q_1 x Q_2 y\ \neg R(y,x)$ where $Q_1, Q_2 \in \{\forall, \exists\}$.*

[9]This is easy to see. In fact, even $\mathbf{FO}^2$ sentences generate infinitely many integer sequences. Take, for instance the sentences φ_k of the form $\varphi_k = \exists x\ \bigvee_{i=1}^{k} U_i(x)$. Their combinatorial spectra are $\mathfrak{S}(\varphi_k) = \left((2^k)^n - 1\right)_{n=1}^{\infty}$. Hence, we have infinitely many combinatorial spectra even for these simple sentences—one for each $k \in \mathbb{N}$.

As can be seen from this example, the branching factor is large even when the first-order language of the sentences contains just one binary predicate. However, if we actually computed the combinatorial spectra of these sentences, we would see that many of them are *redundant* (we give a precise definition of this term in the next subsection). Furthermore, if we were able to detect which sentences are *redundant* without computing their spectra, we would save a significant amount of time. This is because the computation of combinatorial spectra, even though polynomial in n, is still computationally expensive. Moreover, if we were able to remove some sentences from the search, while guaranteeing that all *non-redundant* sentences would still be generated, we would save even more time. In the remainder of this section, we describe such techniques—either techniques that mark sentences as just *redundant*, in which case we will not compute their combinatorial spectra, or also as *safe-to-delete*, in which case we will not even use them to generate new sentences. We will use the term *not-safe-to-delete* when we want to refer to sentences which are *redundant* but not *safe-to-delete*.

What Does It Mean That a Sequence Is Redundant?

Given a collection of sentences $\mathcal{S}$, a sentence $\varphi \in \mathcal{S}$ is considered *redundant* if there is another sentence $\varphi' \in \mathcal{S}$ and $\mathfrak{S}(\varphi) = \mathfrak{S}(\varphi')$, i.e., if the other sentence generates the same integer sequence. Since checking whether two sentences have the same combinatorial spectrum is computationally hard,[10] we will only search for sufficient conditions for when two sentences generate the same spectrum.

Apart from the above notion of redundancy, we also consider a sentence $\varphi \in \mathcal{S}$ redundant if there are two other sentences φ', φ'' such that $\mathfrak{S}(\varphi) = \mathfrak{S}(\varphi') \cdot \mathfrak{S}(\varphi'')$, where the product $\cdot$ is taken element-wise. The rationale is that when this happens, the set of models of φ likely corresponds to the elements of the Cartesian product of the models of φ' and φ'' (or, at least, there is a bijection between them), which is not combinatorially very interesting.[11]

Detecting Redundant Sentences

Now that we explained what we mean by *redundant sentences*, we can move on to methods for detecting whether a sentence is redundant and if it is then whether it is also *safe-to-delete*. We stress upfront that the methods described in this section will not guarantee detecting all redundancies. On the other hand, these methods will be sound—they will not mark non-redundant sentences as redundant. Some of the techniques will mark a sentence as redundant but they will not give us a witness for the redundancy, i.e., other sentences with the same combinatorial spectrum. This will be the case for techniques that guarantee that the witness is a shorter sentence (in the number of literals), which must have been generated earlier, thus, we will know that by pruning the longer redundant sentences, we will not affect completeness of the search.

[10]See the online technical report for details.

[11]After all, one can always create such sequences in a post-processing step and interpret them as elements of the respective Cartesian products if one so desires.

Pruning Technique Name	**Short Description of the Idea**
Sentences detected using the techniques described below are safe-to-delete.	
Isomorphic Sentences	Two sentences are isomorphic if one can be obtained from the other by renaming variables and predicate names.[12]
Decomposable Sentences	If a sentence φ can be written as a conjunction $\varphi = \varphi' \wedge \varphi''$ of two conjunctions with disjoint sets of predicates then φ is redundant. *See the main text for a justification.*
Negations	If two sentences can be made isomorphic by negating all occurrences of literals of some predicates, then they generate the same combinatorial spectra. For example, $\mathfrak{S}(\forall x \exists y\ R(x,y)) = \mathfrak{S}(\forall x \exists y\ \neg R(x,y))$.
Permuting Arguments	*Argument-flip* on a predicate R is a transformation which replaces all occurrences of $R(x,y)$ by $R(y,x)$ and all occurrences of $R(y,x)$ by $R(x,y)$. If two sentences φ and φ' can be made isomorphic using argument flips, then they generate the same combinatorial spectra, i.e., $\mathfrak{S}(\varphi) = \mathfrak{S}(\varphi')$.
Sentences detected by techniques described below are not-safe-to-delete. *We do not compute combinatorial spectra for those sentences and do not store them in the database.*	
Tautologies & Contradictions	Any sentence which contains an always-true (i.e., tautological) clause is redundant—there exists a shorter sentence with the same combinatorial spectrum. All unsatisfiable sentences (contradictions) produce the same combinatorial spectrum consisting of zeros and, hence, are also redundant.
Trivial Constraints	Suppose a sentence φ contains a clause of the form $\forall x\ U(x)$ or $\forall x \forall y\ R(x,y)$, which we call a *trivial constraint*. Then the sentence φ' obtained from φ by dropping the trivial constraint and replacing all occurrences of U or R, respectively by *true*, has the same combinatorial spectrum as φ.
Reflexive Atoms	If a binary literal R appears in a sentence φ only as $R(x,x)$, $\neg R(x,x)$, $R(y,y)$ or $\neg R(y,y)$ and φ has at least two literals, then the sentence is redundant. *See the main text for a justification.*
Subsumption	If a sentence φ contains two clauses $Q_1 x Q_2 y\ \alpha(x,y)$ and $Q_1 x Q_2 y\ \beta(x,y)$, with the same quantifier prefix, and if there is a substitution $\theta : \{x,y\} \to \{x,y\}$ such that $\alpha\theta \subseteq \beta$, then φ is redundant—the sentence φ' obtained from φ by dropping $Q_1 x Q_2 y\ \beta(x,y)$ generates the same combinatorial spectrum.
Cell Graph Isomorphism	*See the main text.*

Table 1: Pruning techniques used in the algorithm for generation of non-redundant sentences.

The pruning methods that are used by our algorithm for generation of sentences are summarized in Table 1. Some of them are rather straightforward and do not require much further justification here.

The method called *Isomorphic Sentences* is a straightforward extension of the methods for enumeration of non-isomorphic patterns, known from data mining literature (see, e.g., [Nijssen and Kok, 2001]), where the main difference is that when checking isomorphism, we allow renaming of predicates.[12]

The method called *Decomposable Sentences* is based on the following observation, which is well-known among others in lifted inference literature [Van den Broeck, 2011]: Let $\varphi = \varphi_1 \wedge \varphi_2$ be a first-order logic sentence. If φ_1 and φ_2 use disjoint sets of predicates then it is not hard to show that $\mathfrak{S}(\varphi) = \mathfrak{S}(\varphi_1) \cdot \mathfrak{S}(\varphi_2)$, where the product is taken element-wise and $\mathfrak{S}(\varphi_1)$ and $\mathfrak{S}(\varphi_2)$ are understood to be computed only over the languages consisting of the predicates contained in φ_1 and φ_2, respectively.

While the method called *Permuting Arguments* may not need a more detailed explanation per se, we will still illustrate it here on an example to provide a better intuition. Suppose that we have two sentences: $\varphi_1 = \forall x \exists y\ E(x,y)$ and $\varphi_2 = \forall x \exists y\ E(y,x)$. The first one can be interpreted as modelling directed graphs in which no vertex has out-degree 0 and the second one as modelling directed graphs in which no vertex has in-degree 0. This interpretation was based on our decision to interpret $E(x,y)$ as an edge form x to y, yet we could have also interpreted it as an edge from y to x and this would change nothing about the combinatorial spectrum of the sentence (which does not depend on how we interpret the sentence). If we generalize this observation, we realize that sentences that differ only in the order of arguments of some predicates (like φ_1 and φ_2 above) must generate the same combinatorial spectrum.

Next, we give a little more detail on the method called *Reflexive Atoms*. If a sentence φ contains atoms of some binary predicate R only in the form $R(x,x)$ or $R(y,y)$ then all the ground atoms $R(i,j)$, where i and j are domain elements and $i \neq j$, are unconstrained by φ. It follows that $\mathfrak{S}(\varphi) = \mathfrak{S}(\varphi') \cdot \mathfrak{S}(\varphi'')$ where $\varphi' = \forall x\ \neg R(x,x)$ and φ'' is a sentence obtained by replacing all occurrences of $R(x,x)$ by $U_R(x)$ and occurrences of $R(y,y)$ by $U_R(y)$ where U_R is a fresh predicate. Here, φ' accounts for all possible configurations of the atoms $R(i,j)$ with arguments $i \neq j$. It follows

[12]The details are described in the online technical report.

that such a sentence φ is redundant.

The methods *Tautologies & Contradictions*, *Negations*, *Trivial Constraints* and *Subsumption*, do not need any further explanation beyond what is in Table 1. The only remaining method, namely *Cell Graph Isomorphism*, is described in the next subsection.

Cell Graph Isomorphism

The final pruning method of Table 1 called *Cell Graph Isomorphism* relies on a concept from the area of lifted inference. In [van Bremen and Kuželka, 2021], the authors introduced a special structure called a *cell graph* to help them compute WFOMC faster. A cell graph is a complete undirected graph (including loops) where each vertex represents one *cell*[13] [Beame *et al.*, 2015] of the sentence whose WFOMC we seek to compute. Each edge of the cell graph is then labelled by certain values computed from the weights $(w, \overline{w})$.

Since the WFOMC computation is completely determined by the cell graph, it is simple to show that if two sentences have isomorphic cell graphs, then they also have the same combinatorial spectrum. See the technical report for details, including how we define isomorphism of cell graphs.

Therefore, it is enough to output just one sentence from each equivalence class induced by cell graph isomorphism. However, as is already stated in Table 1, sentences with isomorphic cell graphs are *not-safe-to-delete*, meaning that combinatorial spectrum is computed only for one member of the induced equivalence class, but all the members are used to further expand the search space.

4 Experiments

In this section, we experimentally evaluate the effectiveness of the techniques for constructing the database of integer sequences described in Section 3.2 within a reasonable amount of time. Furthermore, we take a closer look at a few interesting generated $\mathbf{C}^2$ sentences whose combinatorial spectra appear in OEIS.

4.1 Filling the Database of Integer Sequences

We ran two separate experiments with generators of $\mathbf{FO}^2$ and $\mathbf{C}^2$ sentences. We set a time-limit of five minutes for the computation of combinatorial spectra per sentence; results for these experiments are depicted in Figure 1. Our aim with these experiments was to assess the effect of the pruning techniques that we proposed. We tested several setups which differed in which of the pruning techniques were used. The first setup, the very baseline (shown in black), consisted of just the method that filters out sentences which are isomorphic (using the standard notion of isomorphism used in pattern mining literature, which does not consider renaming predicates [Nijssen and Kok, 2001]) and with pruning of *decomposable sentences*—these are the very essentials any reasonable method would probably implement. The second setup (shown in orange) used the full method *Isomorphic Sentences*, allowing renaming of predicates, *Decomposable Sentences* and *Tautologies & Contradictions*. The third setup (shown in purple) contained all of the methods from the previous setups together with *Negations*. The fourth setup (shown in pink) added the *Permuting Arguments* to it. In the fifth setup (shown in blue) we used all methods except *Cell Graph Isomorphism*. Finally, the last setup (shown in green) contained all the proposed pruning methods. It can be seen that our methods reduce both the runtime and the number of generated sentences by orders of magnitude.

The pruning techniques help to scale up the process of filling the database in two ways. Whereas the naive approach (e.g. *baseline*) generates a lot of sentences fast, soon consuming all available memory, *safe-to-delete* techniques lower the memory requirements significantly. All pruning techniques consume some computation time, but that is negligible compared to the time needed for computing combinatorial spectra, which is the most time-demanding part of the task. Since the pruning methods, including those which are *not-safe-to-delete*, reduce the number of computations of combinatorial spectra, their use quickly pays off, as can be clearly seen from Figure 1 which shows the estimated[14] time to fill in the database.

We refer to the technical report for more experiments with detailed information about their setups.

4.2 An Initial Database Construction

Apart from the experiments in which we compared the benefits of the proposed pruning methods, we also used our algorithm to generate an initial database of combinatorial sequences. For that we let the sentence generator run for five days to obtain a collection of sentences and their combinatorial spectra on a machine with 500 GB RAM, 128 processors (we used multi-threading). We used a five-minute time limit for combinatorial spectrum computation of a sequence.

The result was a database containing over 26,000 unique integer sequences. For each of the sequences in our database, we queried OEIS to determine if the sequence matches a sequence which is already in OEIS. We found that 301 of the sequences were present in OEIS—this makes $\approx$1.2% of the sequences we generated. This may not sound like much, but it is certainly non-negligible. Moreover, our goal was to generate primarily new sequences. We show several interesting generated sequences that happened to be in OEIS in Table 2.

An example of an interesting sequence is the last one in Table 2. This sequence does not have any combinatorial characterization in OEIS. We can obtain such a characterization from the $\mathbf{C}^2$ sentence that generated it:[15] $(\forall x \forall y\ U(x) \lor \neg U(y) \lor \neg B(x,y)) \land (\forall x \exists^{=1} y\ B(x,y))$. This can be interpreted as follows: *We are counting configurations consisting of a function* $b : [n] \to [n]$ *and a set* $U \subseteq [n]$ *that satisfy that if* $y = b(x)$ *and* $y \in U$ *then* $x \in U$. While this may not be

[13]Cells are also referred to as 1-types in logics literature. They are a maximal consistent conjunction of literals formed from atoms of the input formula using only a single variable.

[14]Since the methods which do not use the full set of our pruning techniques, generate an extremely high number of (mostly redundant) sentences, computing their spectra would take thousands of hours. Therefore, we only estimated the runtime by computing the spectra only for a random sample of sentences for these methods.

[15]For easier readability, we replaced the predicate B by its negation, which does not change the spectrum.

Sentence	OEIS ID	OEIS name
$(\forall x \exists^{=1} y\ B(x,y)) \land (\forall x \exists^{=1} y\ B(y,x)) \land (\forall x \forall y B(x,x) \lor B(x,y) \lor \neg B(y,x))$	A85	Number of self-inverse permutations on n letters, also known as involutions; number of standard Young tableaux with n cells.
$(\forall x \exists^{=1} y\ B(x,y)) \land (\forall x \exists^{=1} y\ B(y,x))$	A142	Factorial numbers: $n! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot \ldots \cdot n$ (order of symmetric group S_n, number of permutations of n letters).
$(\forall x \exists^{=1} y\ \neg B(x,y)) \land (\forall x \exists^{=1} y\ \neg B(y,x)) \land (\forall x\ B(x,x))$	A166	Subfactorial or rencontres numbers, or derangements: number of permutations of n elements with no fixed points.
$(\forall x \forall y\ B(x,y) \lor \neg B(y,x)) \land (\exists x\ B(x,x)) \land (\forall x \exists^{=1} y\ \neg B(x,y))$	A1189	Number of degree-n permutations of order exactly 2.
$(\forall x \forall y\ U(x) \lor B(x,y)) \land (\forall x \forall y\ \neg U(x) \lor B(y,x))$	A47863	Number of labeled graphs with 2-colored nodes where black nodes are only connected to white nodes and vice versa.
$(\forall x\ B(x,x)) \land (\forall x \exists y\ \neg B(x,y)) \land (\forall x \exists y\ \neg B(y,x))$	A86193	Number of $n \times n$ matrices with entries in $\{0,1\}$ with no zero row, no zero column and with zero main diagonal.
$(\forall x \forall y\ U(x) \lor \neg U(y) \lor B(x,y)) \land (\forall x \exists^{=1} y\ \neg B(x,y))$	A290840	$a(n) = n! \cdot [x^n] \frac{\exp(n \cdot x)}{1 + LambertW(-x)}$.

Table 2: A sample of sequences that are combinatorial spectra of sentences generated by our algorithm that also appear in OEIS.

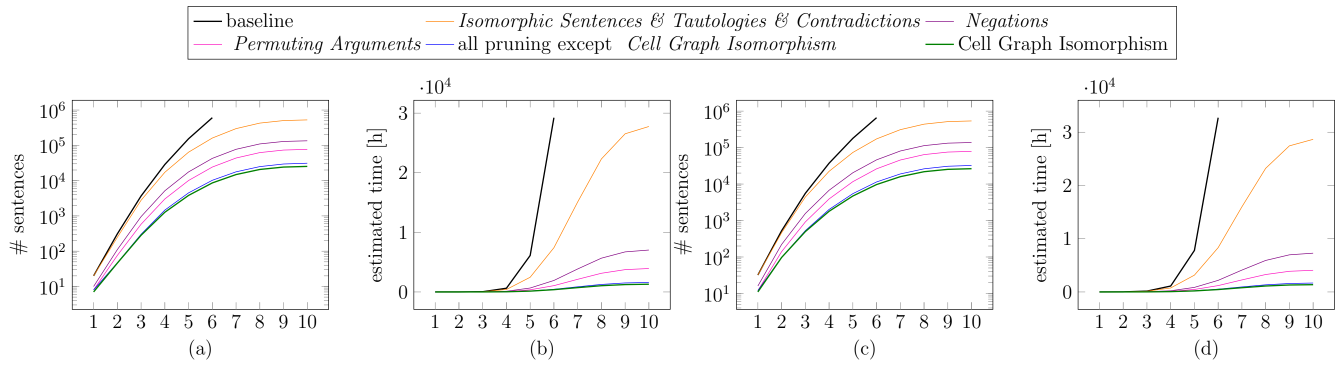

Figure 1: Number of $\mathbf{FO}^2$ (a) and $\mathbf{C}^2$ (c) sentences with at most x literals. The estimated time needed to generate $\mathbf{FO}^2$ (b) and $\mathbf{C}^2$ (d) sentences and compute their combinatorial spectra with a five-minute time limit per spectrum. Each of these sentences can have at most one unary, and one binary predicate and at most two clauses, five literals per clause which result in the flattening of the (a) and (c) curves. Some generators, e.g. the baseline, did not get to the final levels of the search, since they exceeded the time or memory limit.

a profound combinatorial problem, it provides a combinatorial interpretation for the sequence at hand—we would not be able to find it without the database.

Next we discuss several examples of arguably natural combinatorial sequences that were constructed by our algorithm which are not present in OEIS. The first of these examples is the sequence 0, 0, 6, 72, 980, 15360, ... generated by the sentence $(\forall x\ \neg B(x,x)) \land (\exists x \forall y\ \neg B(y,x)) \land (\forall x \exists^{=1} y\ B(x,y))$. We can interpret it as counting the number of functions $f : [n] \to [n]$ without fixed points and with image not equal to $[n]$. Another example is the sequence 1, 7, 237, 31613, 16224509, 31992952773, ..., which corresponds to the sentence $(\forall x \exists y\ B(x,y)) \land (\exists x \forall y\ B(x,y) \lor B(y,x))$ and counts directed graphs on n vertices in which every vertex has non-zero out-degree and there is a vertex that is connected to all other vertices (including to itself) by either an outgoing or incoming edge. Yet another example is the sequence 1, 5, 127, 12209, 4329151, 5723266625, ..., corresponding to the sentence $(\forall x \exists y\ B(x,y)) \land (\exists x \forall y\ B(x,y))$, which counts directed graphs where every vertex has non-zero out-degree and at least one vertex has out-degree n, which is also the same as the number of binary matrices with no zero rows and at least one row containing all ones. These examples correspond to the simpler structures in the database, there are others which are more complex (and also more difficult to interpret). For example, another sequence 0, 3, 43, 747, 22813, 1352761, ... constructed by our algorithm, given by the sentence $(\forall x\ \neg B(x,x)) \land (\forall x \forall y\ \neg B(x,y) \lor B(y,x)) \land (\exists x \forall y\ \neg B(x,y) \lor \neg U(y)) \land (\exists x \exists y\ B(x,y))$, counts undirected graphs without loops with at least one edge and with vertices labeled by two colors, red and black (red corresponding to $U(x)$, and black corresponding to $\neg U(x)$) such that there is at least one vertex not connected to any of the red vertices (note that this vertex can itself be red). We could keep on listing similar sequences, but we believe the handful we showed here give sufficient idea about the kind of sequences one could find in the database constructed by our system.

5 Related Work

To our best knowledge, there has been almost no prior work on automated generation of combinatorial sequences, with the work [Albert *et al.*, 2022] which focuses on sequences generated by permutations avoiding certain patterns being an exception. However, there were works that intersect with the work presented in this paper in certain aspects. The most closely related are works on lifted inference [Poole, 2003; Gogate and Domingos, 2011; Van den Broeck, 2011; Van den Broeck *et al.*, 2014; Beame *et al.*, 2015; Kuželka, 2021]; this work would not be possible without lifted inference. We directly use the algorithms, even though re-implemented, as well as the concept of cell graphs from [van Bremen and Kuželka, 2021]. The detection of isomorphic sentences is similar to techniques presented in [van Bremen *et al.*, 2021], however, that work focused on propositional logic problems, whereas here we use these techniques for problems with first-order logic sentences. There were also works on automated discovery in mathematics, e.g. [Colton, 2002; Davies *et al.*, 2021], but as far as we know, none in enumerative combinatorics that would be similar to ours. The closest line of works at the intersection of combinatorics and artificial intelligence are the works [Suster *et al.*, 2021] and [Totis *et al.*, 2023]. However, those works do not attempt to generate new sequences or new combinatorics results, as they mostly aim at solving textbook-style combinatorial problems, which is still a highly non-trivial problem too, though.

6 Conclusion

We have introduced a method for construction of a database of integer sequences with a combinatorial interpretation and used it to generate a small initial database consisting of more than 26k unique sequences, of which a non-negligible fraction appears to have been studied, which is a sign that we are able to generate interesting integer sequences automatically. Our approach has two key components: an existing lifted-inference algorithm [van Bremen and Kuželka, 2021] that computes sequences from first-order logic sentences and the new method for generation of first-order sentences which successfully prunes huge numbers of redundant sentences.

Acknowledgments

The work of MS and OK was supported by the Czech Science Foundation project 20-19104Y and 23-07299S (part of the work was done before the start of the latter project). JT, MS, OK, PJ were also supported by the OP VVV project *CZ.02.1.01/0.0/0.0/16_019/0000765* "Research Center for Informatics". The authors acknowledge a generous donation by X-Order Lab.

References

[Albert *et al.*, 2022] Michael H. Albert, Christian Bean, Anders Claesson, Émile Nadeau, Jay Pantone, and Henning Ulfarsson. Combinatorial Exploration: An algorithmic framework for enumeration. https://arxiv.org/abs/2202.07715, 2022.

[Barvínek *et al.*, 2021] Jáchym Barvínek, Timothy van Bremen, Yuyi Wang, Filip Zelezný, and Ondrej Kuzelka. Automatic conjecturing of p-recursions using lifted inference. In Nikos Katzouris and Alexander Artikis, editors, *Inductive Logic Programming - 30th International Conference, ILP 2021, Proceedings*, volume 13191 of *Lecture Notes in Computer Science*, pages 17–25. Springer, 2021.

[Beame *et al.*, 2015] Paul Beame, Guy Van den Broeck, Eric Gribkoff, and Dan Suciu. Symmetric weighted first-order model counting. In *Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '15, page 313–328, New York, NY, USA, 2015. Association for Computing Machinery.

[Börger *et al.*, 2001] Egon Börger, Erich Grädel, and Yuri Gurevich. *The classical decision problem*, page 48. Springer Science & Business Media, 2001.

[Colton, 2002] Simon Colton. The hr program for theorem generation. In *International Conference on Automated Deduction*, pages 285–289. Springer, 2002.

[Davies *et al.*, 2021] Alex Davies, Petar Veličković, Lars Buesing, Sam Blackwell, Daniel Zheng, Nenad Tomašev, Richard Tanburn, Peter Battaglia, Charles Blundell, András Juhász, et al. Advancing mathematics by guiding human intuition with ai. *Nature*, 600(7887):70–74, 2021.

[Gogate and Domingos, 2011] Vibhav Gogate and Pedro M. Domingos. Probabilistic theorem proving. In Fábio Gagliardi Cozman and Avi Pfeffer, editors, *UAI 2011, Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, Barcelona, Spain, July 14-17, 2011*, pages 256–265. AUAI Press, 2011.

[Grädel and Otto, 1999] Erich Grädel and Martin Otto. On logics with two variables. *Theoretical computer science*, 224(1-2):73–113, 1999.

[Hinrichs and Genesereth, 2006] Timothy Hinrichs and Michael Genesereth. Herbrand logic. Technical Report LG-2006-02, Stanford University, Stanford, CA, 2006. http://logic.stanford.edu/reports/LG-2006-02.pdf.

[Kuželka, 2021] Ondřej Kuželka. Weighted first-order model counting in the two-variable fragment with counting quantifiers. *Journal of Artificial Intelligence Research*, 70:1281–1307, 2021.

[Nijssen and Kok, 2001] Siegfried Nijssen and Joost N. Kok. Faster association rules for multiple relations. In Bernhard Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001*, pages 891–896. Morgan Kaufmann, 2001.

[OEIS Foundation Inc., 2023] OEIS Foundation Inc. The on-line encyclopedia of integer sequences. Published electronically at http://oeis.org, 2023. Accessed: 2023-05-23.

[Pak, 2018] Igor Pak. Complexity problems in enumerative combinatorics. In *Proceedings of the International Congress of Mathematicians: Rio de Janeiro 2018*, pages 3153–3180. World Scientific, 2018.

[Poole, 2003] David Poole. First-order probabilistic inference. In Georg Gottlob and Toby Walsh, editors, *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 985–991. Morgan Kaufmann, 2003.

[Pratt-Hartmann, 2009] Ian Pratt-Hartmann. Data-complexity of the two-variable fragment with counting quantifiers. *Information and Computation*, 207(8):867–888, 2009.

[Stanley, 1986] Richard P Stanley. What is enumerative combinatorics? In *Enumerative combinatorics*, pages 1–63. Springer, 1986.

[Suster *et al.*, 2021] Simon Suster, Pieter Fivez, Pietro Totis, Angelika Kimmig, Jesse Davis, Luc De Raedt, and Walter Daelemans. Mapping probability word problems to executable representations. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3627–3640, 2021.

[Totis *et al.*, 2023] Pietro Totis, Jesse Davis, Luc De Raedt, and Angelika Kimmig. Lifted reasoning for combinatorial counting. *Journal of Artificial Intelligence Research*, 76:1–58, 2023.

[van Bremen and Kuželka, 2021] Timothy van Bremen and Ondřej Kuželka. Faster lifting for two-variable logic using cell graphs. In *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*, volume 161 of *Proceedings of Machine Learning Research*, pages 1393–1402. PMLR, 27–30 Jul 2021.

[van Bremen *et al.*, 2021] Timothy van Bremen, Vincent Derkinderen, Shubham Sharma, Subhajit Roy, and Kuldeep S. Meel. Symmetric component caching for model counting on combinatorial instances. In *Thirty-Fifth AAAI Conference on Artificial Intelligence*, pages 3922–3930. AAAI Press, 2021.

[Van den Broeck *et al.*, 2011] Guy Van den Broeck, Nima Taghipour, Wannes Meert, Jesse Davis, and Luc De Raedt. Lifted probabilistic inference by first-order knowledge compilation. In Toby Walsh, editor, *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2178–2185. AAAI Press/International Joint Conferences on Artificial Intelligence, 2011.

[Van den Broeck *et al.*, 2014] Guy Van den Broeck, Wannes Meert, and Adnan Darwiche. Skolemization for weighted first-order model counting. In *Proceedings of the Fourteenth International Conference on Principles of Knowledge Representation and Reasoning*, KR'14, page 111–120. AAAI Press, 2014.

[Van den Broeck, 2011] Guy Van den Broeck. On the completeness of first-order knowledge compilation for lifted probabilistic inference. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, page 1386–1394, Red Hook, NY, USA, 2011. Curran Associates Inc.