Constrained Molecule Generation Modelled Using the Grammar Constraint

David Saikali, Gilles Pesant

Department of Computer and Software Engineering, Polytechnique Montréal, Montreal, Canada
david.saikali@polymtl.ca, gilles.pesant@polymtl.ca

Abstract

Drug discovery is a very time-consuming and costly endeavour due to its huge design space and to the lengthy and failure-fraught process of bringing a product to market. Automating the generation of candidate molecules exhibiting some of the desired properties can help. Among the standard formats to encode molecules, SMILES is a widespread string representation. We propose a constraint programming model showcasing the grammar constraint to express the design space of organic molecules using the SMILES notation. We show how some common physicochemical properties — such as molecular weight and lipophilicity — and structural features can be expressed as constraints in the model. We also contribute a weighted counting algorithm for the grammar constraint, allowing us to use a belief propagation heuristic to guide the generation. Our experiments indicate that such a heuristic is key to driving the search towards desired molecules.

Code — github.com/cravethedave/MiniCPBP/tree/AAAI26

1 Introduction

Drug discovery is a very time-consuming and costly endeavour due to its huge design space — estimated to contain between 10^{23} and 10^{60} different molecules (Polishchuk, Madzhidov, and Varnek 2013) — and to the lengthy and failure-fraught process of bringing a product to market. Automated molecule design is nowadays a vital part of drug discovery and material science, with computational approaches coming from deep generative models and combinatorial search methods (Du et al. 2022). It aims to extract from this huge design space the most likely candidates according to some desired properties. And even among these, only a few may lead to a usable product after extensive testing.

SMILES, a one-dimensional encoding of molecules, is one of the standards commonly used by this research community. It lends itself well to techniques used for natural language processing, such as sequential generative neural models, but also to constraint programming (CP). Using a context-free grammar and a few additional constraints, we show how to describe valid SMILES strings in a CP model. This allows us to explore the huge design space of possible molecules while adding constraints in order to restrict that space to suitable candidates.

Even though the grammar (CFG) constraint was introduced almost 20 years ago (Sellmann 2006; Quimper and Walsh 2006), it has generated little interest from the CP community so far: it does not appear in the dominant modeling standards MiniZinc and XCSP nor is it supported by mainstream constraint solvers. There may be two reasons for this: typical applications of CP seldom require it to model the problem (even though there are obvious applications such as natural language processing) and its filtering algorithm is relatively expensive to run (cubic in the number of variables in its scope). With this paper we contribute: i) a natural application of the CFG constraint; ii) a weighted counting algorithm for CFG to contribute to search guidance; iii) empirical evidence that CP (augmented with belief propagation) is an effective technique for *in silico* constrained generation of candidate molecules exhibiting prescribed characteristics.

The remainder of the paper is organized as follows. Section 2 provides the necessary background. Section 3 reviews the related work. Section 4 presents our CP model for constrained molecule generation. Section 5 describes the weighted counting algorithm for context-free grammar constraints. Section 6 evaluates our approach empirically. Finally Section 7 recalls our contributions and identifies some directions for future research.

2 Background

This section provides background on organic chemistry, on formal grammars, and on CP-based belief propagation. Atoms are the building blocks of molecules and the bonds they can make are what allows the formation of complex structures. The number of bonds an atom can make is limited by the electrons in its valence shell, also called valence electrons. This valence shell refers to the outermost layer of electrons. By making ionic or covalent bonds, an atom can reach a more stable state. If we take Hydrogen and Carbon as examples, two of the more common atoms in organic chemistry, they need one and four more electrons respectively to complete their valence shell. They can do this by making the corresponding number of covalent bonds (commonly represented as line segments between atoms; see e.g. Figure 1A).

Copyright © 2026, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

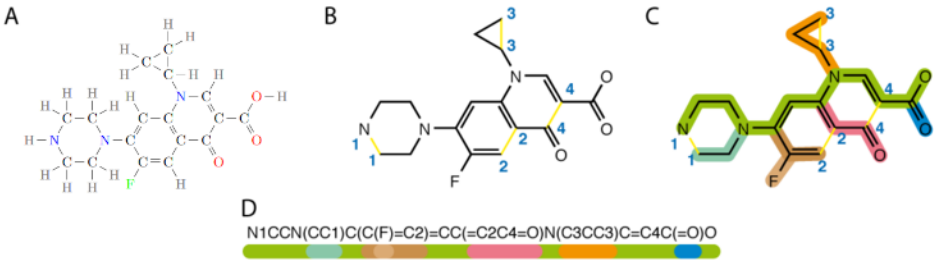

Figure 1: Deriving a SMILES representation for a molecule (reproduced in part from (Wikimedia 2007)). The structural formula of the molecule (A), its skeletal formula stripped of all hydrogen atoms and with broken cycles (B), the selected main path (shown in green) and branches (C), and the corresponding SMILES notation (D).

2.1 SMILES Representation Format

SMILES (Simplified Molecular-Input Line-Entry System) (Weininger 1988) is a standard to represent molecules as short ASCII strings. Characters include the usual symbols for atoms. For example, the water molecule (H_2O) is made up of two hydrogen atoms, both of which form a single bond with a central oxygen atom. In SMILES notation this can be abbreviated to a simple O. Such simplification relies on the fact that oxygen requires two bonds to reach a stable state (where they have a full valence shell). Any bond an atom seems to be missing to reach this stable state is implicitly made with a hydrogen atom.

Of course not all compounds are that simple, and, in particular, may contain cycles (see e.g. Figure 1). The first step in building a SMILES string is to break the cycles present in the molecule. To retain the broken bond's information, we add an identical numeric token following each of the previously connected atoms. For example cyclohexane (C_6H_{12}) is made up of six carbon atoms arranged in a cycle through single bonds and with two hydrogen atoms bound to each. Its representation is C1CCCCC1, indicating that the first and last carbon atoms in the chain are linked. Once cycles are broken, the structure forms a tree: we choose one path as the main path and the other ones become branches. In organic chemistry, the main path is typically the longest. A branch is written in parentheses before the main path continues. Note that this way of handling branches allows for two different SMILES strings to describe the same molecule. Like hydrogen, single bonds are implicit in the notation. Double and triple bonds are indicated using = and # respectively. For example ethylene (C_2H_4), written as C=C, has a carbon-carbon double bond. Figure 1 illustrates the conversion process for a more complex molecule. Note that we only covered the basics of SMILES notation. In reality, it is an incredibly in-depth system that can account for ions, isotopes, and so forth.

One challenge of molecule generation when using the SMILES notation is that not all of its strings are valid. In particular, branches are represented using parentheses and the language of balanced parentheses is well known for not being regular. A context-free grammar has the necessary expressive power and the flexibility to cover most rules in the SMILES syntax. So using a context-free grammar does help in guaranteeing that the string is syntactically valid, but it may still be chemically invalid. To resolve this issue, Kraev creates a grammar to ensure that atom valences are respected and balanced. He also introduces the concept of masking: each mask works as a secondary restriction on the generation, which prevents more invalid combinations than the previous configuration. For example, one mask ensures that cycles in the molecule are closed at the end of the generation. We use a slightly-adapted version of his grammar in our CP model (see Section 4). Some more recent string representations guarantee syntactic and chemical validity (e.g. SELFIES (Krenn et al. 2020)) but their use is not nearly as widespread.

2.2 Lipinski's Rule of 5

Lipinski's Rule of 5 (Lipinski et al. 1997) describes four physicochemical properties that molecules fit to be orally active drugs in humans tend to respect. Its name comes from the fact that all the threshold values used to express these properties are multiples of 5. The first rule is a limit on the molecular weight: a viable drug should be limited to 500 Da (or g/mol). The next two rules concern hydrogen bonds: fewer than 5 hydrogen-bond donors and fewer than 10 hydrogen-bond acceptors. Both donors and acceptors require electronegative atoms such as nitrogen, oxygen, sulfur, or fluorine. For one of these atoms to be a hydrogen-bond *acceptor*, it must have a lone electron pair to form the hydrogen-bond. In most cases, these atoms will be hydrogen-bond acceptors. The conditions for them to be a hydrogen-bond *donor* is for the atom to have a bond with a hydrogen atom. Finally, the last rule has to do with how hydrophilic or lipophilic (i.e. hydrophobic) a molecule is. This is computed as the log of the partition coefficient: a ratio between the concentrations of the solute in two different solvents. The higher the logP score is, the more lipophilic it is, meaning the solute prefers a non-polar solvent to water. Lipinski's rule of five says that the logP score of a viable molecule should not exceed 5. These rules are considered more as a guide than a hard limit since several exceptions can be found in drug databases.

2.3 Context-Free Grammar

A grammar is a set of rewrite rules to generate a set of strings. Formally, grammar $\mathcal{G} = (\mathcal{N}, \Sigma, \mathcal{R}, S)$ is defined by a set of nonterminal symbols $\mathcal{N}$, a set of terminal sym-

bols Σ (its alphabet), a set of production rules $\mathcal{R}$, and a start symbol S. We denote $L(\mathcal{G})$ the language recognized by $\mathcal{G}$ i.e. the set of strings that grammar can generate. In a context-free grammar the left-hand side of the production must be a single nonterminal, and the right-hand side must be a string of terminals and nonterminals. For example context-free grammar $\mathcal{G} = (\{S, A, B, C\}, \{\langle, \rangle\}, \{S \to SS, S \to AC, S \to BC, B \to AS, A \to \langle, C \to \rangle\}, S)$ recognizes correctly bracketed words such as "$\langle\langle\rangle\rangle$", obtained by the successive application of rules: $S \to BC \to ASC \to AS\rangle \to AAC\rangle \to A\langle C\rangle \to A\langle\rangle\rangle \to \langle\langle\rangle\rangle$. Some of these rules could have been applied in a different order, but all such orderings correspond here to the same parse tree (the red one in Figure 4 left).

In CP, given a context-free grammar $\mathcal{G}$ and a sequence of finite-domain variables $\langle X_1, X_2, \ldots, X_n\rangle$ with $X_i \in D(X_i) \subseteq \Sigma$, constraint $\text{CFG}(\mathcal{G}, \langle X_1, X_2, \ldots, X_n\rangle)$ holds if the sequence of values taken by $X_1, X_2, \ldots, X_n$ corresponds to a word of $L(\mathcal{G})$. Quimper and Walsh describe a domain-consistency algorithm for the CFG constraint based on the CYK parser. It requires that the grammar be in *Chomsky Normal Form*: all production rules are either of the form $A \to BC$ or $A \to a$ where A, B, C are nonterminals and a a terminal. This is not restrictive because any context-free grammar can be put into that form.

2.4 CP-based Belief Propagation

The MiniCPBP solver[1] generalizes standard constraint propagation in CP through a message-passing phase akin to belief propagation that outputs from each constraint probability mass functions (PMFs) over the domain of the individual variables in its scope, representing how frequently a domain value appears in a solution to that constraint (Pesant 2019). Such information is computed through weighted model counting on individual constraints. In Section 5 we contribute a weighted counting algorithm for the CFG constraint. This propagation of PMFs can approximate the marginals of individual variables for the whole CP model. Such marginals have been used to design branching heuristics to solve combinatorial problems (Babaki, Omrani, and Pesant 2020; Burlats and Pesant 2023) and to enable neurosymbolic architectures (Lafleur, Chandar, and Pesant 2022; Yin, Cappart, and Pesant 2024, 2025; Manibod, Saikali, and Pesant 2025).

3 Related Work

Drug discovery, and molecule design in general, is a vast topic. A recent survey by Du et al. presents various representation formalisms, some of the main problems tackled, and an array of computational methods used to solve them, mostly generative machine learning but also combinatorial solvers. Among the current challenges for deep generative models, they mention the difficulty of exploring little known/seen areas of the molecular design space (the common out-of-distribution generation issue) and the need for lots of training data (generation in low-data regime issue i.e. high sample complexity). They also mention as opportunity the generation of specialized molecules with more complex structure.

Among combinatorial solvers, the use of constraint programming in this area was pioneered 25 years ago by Krippahl and Barahona for protein structure determination (Krippahl and Barahona 1999). They showed that CP can help determine the position of atoms in a molecule. By approximating the distance between non-hydrogen atoms they infer the shape of the protein. Later work on protein docking (Krippahl and Barahona 2015) uses CP to prune the search space, allowing a trained Naive Bayes classifier to find solutions much faster.

Barbe, Schiex et al. use Cost Function Networks to solve computational protein design problems seeking sequences that fold to specific three-dimensional structures (Vucinic et al. 2020; Charpentier et al. 2019). These Cost Function Networks use energy contributions to find the three-dimensional shape of the molecule.

Several works consider a particular family of molecules, benzenoids, and exploit their special geometry when defining their representation in a CP model and expressing various properties as constraints. Carissan et al.; Varet et al. add constraints to benzenoid generation in order to model certain properties such as the number of carbon atoms or the shape of the molecule. They also formulate the problem of determining local aromaticity as a CSP. Peng and Solnon improve the enumeration of benzenoid graphs by representing them using short canonical codes that are invariant to symmetries and rotations, expressed in a CP model. They ensure the presence of a given pattern by completing a suitably prefixed code. The sequential nature of these codes, obtained through graph traversal, makes them similar in spirit to the SMILES notation, though much less general.

In the context of their work on constrained graph generation using CP, Omrani and Naanaa consider the generation of molecular graphs corresponding to a given molecular formula. Guo et al. propose a sample-efficient neural method for molecule generation that is based on learning a graph grammar.

So despite some prior work involving CP, none address the problem we consider and especially the use of the grammar constraint.

4 Model

This section describes the CP modeling of our problem. A molecule is described by a sequence of n variables whose domain is the alphabet of the SMILES notation.

4.1 SMILES Representation

As mentioned earlier, we use a variation of Kraev's grammar (Kraev 2018) to ensure that atom valences are respected in the generated molecules. One of the modifications we made to the grammar was to integrate the cycle length limit directly, something Kraev did using a mask. MOSES (Polykovskiy et al. 2020), a data set of about two million molecules, never exceeds length-6 cycles while another, Zinc_250k (Akhmetshin et al. 2021), features some length-8 cycles. We therefore chose to limit the cycle length

[1] https://github.com/PesantGilles/MiniCPBP

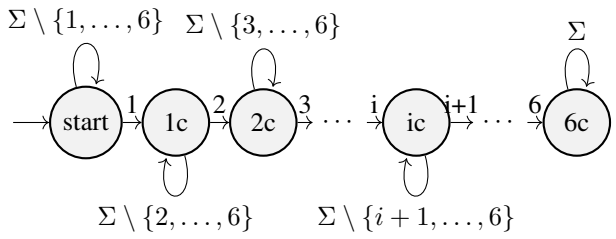

Figure 2: Automaton $\mathcal{A}$ which imposes ordinal order on cycle numbering.

to 8. We also limit the number of cycles in our grammar to 6 (a mere 8 of the 250k molecules in Zinc have more than 6 cycles). Finally for the purpose of using this grammar in our CP model, we add padding tokens that can complete the end of a molecule. This will allow our model to generate any molecule up to the size instead of giving it a fixed length.

The resulting context-free grammar features 538 productions, 194 nonterminal symbols and 32 terminal symbols (the SMILES alphabet). Its conversion into Chomsky normal form features the same number of terminals while the number of productions and nonterminals increase to 1996 and 640 respectively.

Let $\mathcal{G}_{\text{SMILES}} = (\mathcal{N}, \Sigma, \mathcal{R}, S)$ denote the final grammar and $\langle X_1, X_2, \ldots, X_n \rangle,\ X_i \in \Sigma$, the sequence of variables in our model. Constraint

$$\text{CFG}(\langle X_1, X_2, \ldots, X_n \rangle, \mathcal{G}_{\text{SMILES}})$$

ensures that the sequence of values taken by the variables corresponds to a word belonging to the grammar's language. Just as Kraev added masks so that the generated sequences followed some conventions, we add corresponding constraints. We first add a constraint to ensure that cycles in the SMILES string are numbered consecutively in ascending order starting at 1. To do this, we define an automaton $\mathcal{A}$ (see Fig. 2) which does not allow starting a cycle of a higher number until the one preceding it has been started and add constraint

$$\text{REGULAR}(\langle X_1, X_2, \ldots, X_n \rangle, \mathcal{A}).$$

Next, while the SMILES notation does allow for the same cycle number to be reused once the cycle has been closed, it can make the molecule harder to read. We avoid reusing cycle numbers by adding AMONG constraints which constrain the number of occurrences of a cycle number to be either 0 or 2:

$$\text{AMONG}(\langle X_1, X_2, \ldots, X_n \rangle, \{j\}, \{0, 2\}) \quad 1 \leq j \leq 6.$$

These conventions may be seen as applying static symmetry breaking.

In principle we could combine all the constraints of this section into a single CFG constraint but at the expense of a significant increase in size of an already large grammar. We could even in theory list all valid sequences in a single (huge) TABLE constraint since they have bounded length n. But factoring our model of the huge molecule design space into several constraints is the compromise sacrificing global consistency for lower space complexity. From a modeler's perspective, it is also convenient to work directly with a given grammar.

4.2 Targeting Regions of the Design Space

There are of course many ways in which one may wish to target the generation of molecules. We present here those we use in our experiments.

Structural Constraints We can add structural constraints restricting the *number of branches and cycles*. For branches we simply define a variable N_b to represent the number of occurrences of the branch opening symbol:

$$\text{AMONG}(\langle X_1, X_2, \ldots, X_n \rangle, \{\text{“(”}\}, N_b).$$

For cycles we also use AMONG constraints on cycle-number symbols. If we want at least n_c cycles, given that we label cycles consecutively from 1 and do not reuse cycle labels (Section 4.1), we add constraint

$$\text{AMONG}(\langle X_1, X_2, \ldots, X_n \rangle, \{n_c\}, 2).$$

If we want at most n_c, we add

$$\text{AMONG}(\langle X_1, X_2, \ldots, X_n \rangle, \{n_c + 1\}, 0).$$

To require an exact number we use both.

Physicochemical Properties Many useful properties can be estimated from a molecule's representation, even if it is one-dimensional. We show how each one of Lipinski's four rules can be added to our model as constraints.

Molecular Weight To limit the *molecular weight* we first define a table $\mathcal{T}^w$ linking each symbol in Σ to its corresponding weight. Note that this is not as simple as using the weight of each atom and zero for the other symbols since hydrogen atoms are not written in a SMILES string. We avoid most of this issue by adjusting the weights of the atoms in $\mathcal{T}^w$ to compensate for the missing hydrogen atoms. We also associate a corrective negative weight to the tokens representing double bonds, triple bonds and cycle numbers, to counteract the increased weight of atoms since we include the implicit hydrogen atoms. The result is not perfect, but the error on the molecular weight does not exceed 5% when tested on the large datasets. This is reasonable since, as mentioned previously, Lipinski's rules should be taken more as a guide than as a hard rule. We then index that table with variables X_i, linking each with an individual weight variable W_i, and sum them to obtain the weight W of the whole molecule:

$$\begin{aligned} \text{ELEMENT}(\mathcal{T}^w, X_i, W_i) \qquad & 1 \leq i \leq n \\ \text{SUM}(\langle W_1, W_2, \ldots, W_n \rangle, W) & \\ W \leq 500 & \end{aligned}$$

Hydrogen Bonds As explained in Section 2, we will count any nitrogen, oxygen and sulfur atom that appears in our molecules as a hydrogen-bond acceptor. So since we want at most 10 acceptors, we can add the constraints

$$\begin{aligned} \text{AMONG}(\langle X_1, X_2, \ldots, X_n \rangle, \{\text{“}N\text{”}, \text{“}O\text{”}, \text{“}S\text{”}\}, N_a) & \\ N_a \leq 10 & \end{aligned}$$

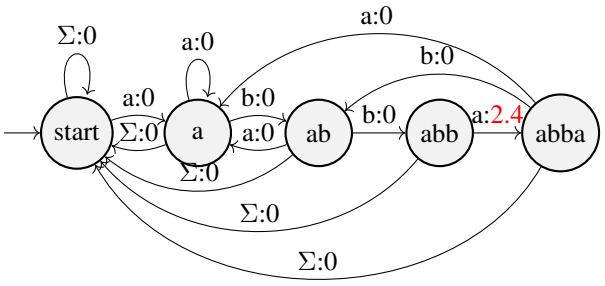

Figure 3: Simplified example of automaton $\mathcal{A}^p$, associating a weight to a sequence during generation. This example associates a weight of 2.4 to the 4-gram **abba**. For clarity and simplicity, we limit ourselves to one 4-gram as the graph quickly becomes very connected with all 4-grams. Transition labels are "symbol:weight". The only transitions with non-zero weight are the ones completing a 4-gram. For all transitions that have Σ associated to them, we assume that any token in another transition is excluded.

Accurately predicting the number of hydrogen-bond donors is harder since we need to track how many bonds each atom is making with other heavy atoms (non-hydrogen atoms). We can include this logic directly in our grammar by first adding new tokens to represent the donor version of the three atoms in question: $\tilde{N}$, $\tilde{O}$ and $\tilde{S}$. Some productions then have to be changed to differentiate between these two versions of the same atom. After these changes, our CFG grammar increases in size to 607 productions, 195 nonterminal symbols and 35 terminal symbols. Once converted to its Chomsky normal form, it features 1990 productions and 642 nonterminals. Similarly to the acceptors, we write

$$\text{AMONG}(\langle X_1, X_2, \ldots, X_n \rangle, \{\text{“}\tilde{N}\text{”}, \text{“}\tilde{O}\text{”}, \text{“}\tilde{S}\text{”}\}, N_d) \\ N_d \leq 5$$

logP To model the *logP* value we follow Vidal, Thormann, and Pons who use a linear regression to estimate the partial contribution, positive or negative, of sequences of four tokens (4-*grams*) on the *logP* score. In their work, they used a partial least squares regression to estimate the logP value of different molecules. Their results indicate that their model could accurately predict logP values.

We add to our model a single COSTREGULAR constraint with automaton $\mathcal{A}^p$, transition cost table $\mathcal{W}_{\mathcal{A}^p}$ indicating the weight associated to each state transition, and *logP* variable P. Automaton $\mathcal{A}^p$ contains a state for each non-zero weight 4-gram as well as states for every 3-gram, 2-gram and 1-gram necessary to get to the non-zero weight 4-grams. The transition to a 4-gram state is weighted with the partial contribution of that 4-gram according to $\mathcal{W}_{\mathcal{A}^p}$. All other state transitions have zero weight and all states in the automaton are considered valid final states. Figure 3 gives a simplified example. The automaton has 41741 states, which is still a considerable improvement in terms of memory, filtering and search guidance compared to an alternative model using $n-3$ SHORTTABLE constraints.

$$\text{COSTREGULAR}(\langle X_1, X_2, \ldots, X_n \rangle, \mathcal{A}^p, \mathcal{W}_{\mathcal{A}^p}, P) \\ P \leq 5$$

5 Weighted Counting Algorithm for the CFG Constraint

In this section, we design a dedicated algorithm for the grammar constraint which computes PMFs for its variables in order to inform branching heuristics for the MiniCPBP solver. The filtering algorithm for the CFG constraint marks triplets (i, j, A) such that there exists at least one string in $L(\mathcal{G}) \cap (D(X_1) \times \cdots \times D(X_n))$ in which nonterminal A generates its substring of size j starting at position i. We take that information as input to our weighted counting algorithm (see Algorithm 1) and denote it as $\mathcal{N}_{ij}$, the set of nonterminal symbols being the root of some parse tree for length-j substrings starting at position i. The other input is a belief (PMF) over the domain of each variable in the scope of the constraint, emanating from the combined beliefs of the other constraints in the CP model. The output of the algorithm are the weighted frequency of variable-value assignments in solutions to the constraint, which we simply call marginals here. We use w_{ijN} to hold the probabilistic weight of the parse trees rooted at nonterminal symbol N for length-j (sub-)words starting at position i, which we compute at Lines 1-15 in a dynamic programming fashion starting from the terminals. We then use f_{ijN} to accumulate the product of weights of branches on either side of a path from the root to the parse tree at N for length-j (sub-)words starting at position i, which we compute at Lines 16-28 again in a dynamic programming fashion but starting from the root. These represent the probabilistic weight of all possible prefix and suffix combinations for that sub-word. Finally Lines 29-34 set the marginals using the f_{i1N} values, corresponding to the weight of supports in solutions. Figure 4 provides an example.

The structure of Algorithm 1 is very similar to that of the original filtering algorithm: it runs in $\Theta(|\mathcal{R}|n^3)$ time using $\Theta(|\mathcal{N}|n^2)$ space. The algorithm is exact for *unambiguous* grammars, i.e. those for which there is a one-to-one correspondence between parse trees and words belonging to the language. Because the counting algorithm proceeds from parse trees, for an ambiguous grammar some words will be counted multiple times and thus overestimate the marginals of the corresponding variable-value (i.e. position-symbol) pairs. The alternative, counting all words directly, is generally intractable.

6 Results

The double purpose of our experiments will be to show how useful the weighted counting algorithm we contributed for the grammar constraint is to guide search and to show that generating potentially useful molecules from a constrained design space modeled in CP using a grammar constraint is effective.

We implemented the CP model described in Section 4 using the MiniCPBP solver, equipped with the CFG constraint. Given this model we compare three solvers:

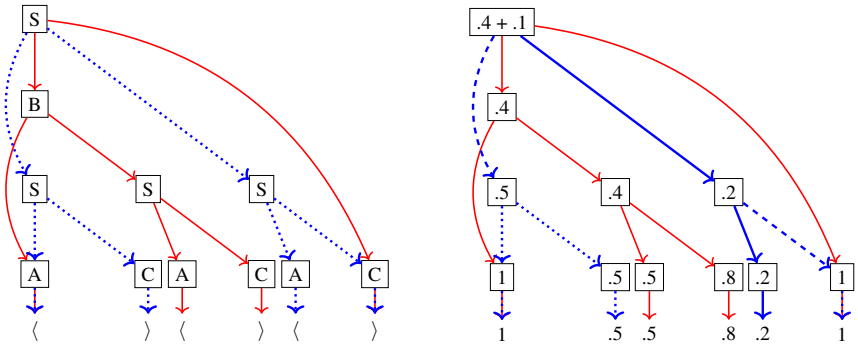

Figure 4: Two parse trees (left) for the two words of length 4 recognized by the example grammar from Section 2.3 and the computed weights w_{ijN} at parse subtrees (right) given some b_{id} values at the bottom. The path in bold from the root to "⟨" in position 3 illustrates the computation of $\theta_{X_3}(\text{“⟨”}) = f_{31A} = w_{12S} \times w_{41C} = .5$ as the product of the weights of the two branches off that path (shown dashed) in the blue dotted parse tree.

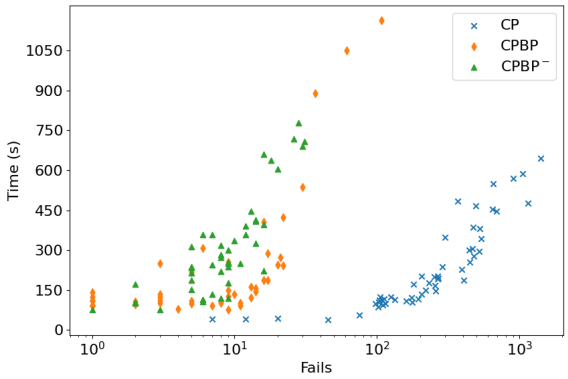

Figure 5: Scatter plot of solved instances for each solver.

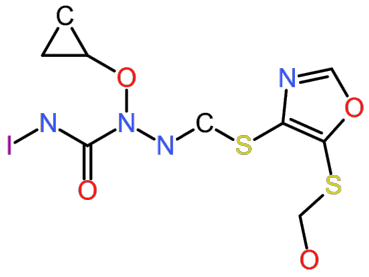

Figure 6: Molecule "OCSc1ocnc1S[C@][NH+]N(OC2[C@]C2)C(=O)NI" of weight $385.1 \in [375, 425]$ Da, logP score $-1.41 \in [-2, -1]$, 3 H-bond donors, 10 H-bond acceptors, 2 cycles, and 2 branches.

Classical CP: uses MiniCPBP with BP disabled and state-of-the-art branching heuristic *domWDeg* (Boussemart et al. 2004). Being a learning-based heuristic, the latter is run with restarts (initially after 100 fails and increased by a 1.5 factor), which is common practice for it. Its default value-selection heuristic, selecting the smallest value in the domain, performed very poorly. Instead we select a domain value uniformly at random and report the median of 11 runs.

CPBP: uses branching heuristic *maxMarginalStrength* (Pesant 2019) based on the marginals computed by MiniCPBP using the weighted counting algorithm of each constraint in the model, including the one we newly designed for CFG. It branches on the variable with the marginal most above a uniform distribution and assigns to it the value exhibiting that marginal. Because it is not learning-based and is deterministic, using restarts would have no effect. Instead we explore the tree using limited-discrepancy search (LDS, with a maximum number of discrepancies starting at 1 and doubled at each iteration, ultimately making the search complete), which is a sensible option for a trusted branching heuristic in order to undo occasional mistakes near the top of the tree (Harvey and Ginsberg 1995).

CPBP⁻: is the same as above but excluding the new weighted counting algorithm for CFG, allowing us to measure its impact.

Our experiments were run on an AMD Rome 7532 processor (2.4GHz, 256M cache L3) with 4 GB of RAM and using a 30-minute timeout on a machine running Rocky Linux 8.10 (Green Obsidian). We set $n = 40$ which is a realistic size encompassing many pharmaceutical molecules.

According to the classification of Du et al. the problem we consider corresponds to *de novo 1D molecule optimization*: we generate molecules from scratch using a string representation and targeting some desired properties. We built 81 constrained problem instances by requiring a mix of structural and physicochemical properties. For structural features we considered every combination of number of cycles and branches in the respective ranges 1..3 and 2..4, yielding 9 combinations. For physicochemical properties we considered those featured in Lipinski's Rule of 5, combining the following restrictions: intervals [175,225], [275,325], [375,425] for molecular weight and [-4,-3], [-2,-1], [1,2] for logP, yielding another 9 combinations. For hydrogen bonds, we keep the original upper bounds of 5 and 10 on the number

Algorithm 1: weightedCount$(\{b_{id}\}, \{\mathcal{N}_{ij}\})$

Input: beliefs from variables: b_{id} for variable X_i and value d (0 whenever $d \notin D(X_i)$); nonterminals appearing in parse trees: $\mathcal{N}_{ij}$ for position i and length j
Output: unnormalized marginals θ_X for each variable X

```
// Clear weights
for i ← 1 to n do
    for j ← 1 to n − i + 1 do
        foreach N ∈ 𝒩 do
            w_ijN ← 0
// Initialize weights for length-one substrings
foreach A → a ∈ 𝒢 do
    for i ← 1 to n do
        if A ∈ 𝒩_i1 then
            w_i1A += b_ia
// Consider substrings of increasing length, accumulating weights
for j ← 2 to n do
    for i ← 1 to n − j + 1 do
        for k ← 1 to j − 1 do
            foreach B ∈ 𝒩_ik do
                foreach A → BC ∈ 𝒢 do
                    if C ∈ 𝒩_{i+k,j−k} then
                        w_ijA += w_ikB × w_{i+k,j−k,C}
// Clear forks
for i ← 1 to n do
    for j ← 1 to n − i do
        foreach N ∈ 𝒩 do
            f_ijN ← 0
// Initialize root of parse trees
f_1nS ← 1
// Consider substrings of decreasing length, accumulating the product of weights that branch off on either side
for j ← n down to 2 do
    for i ← 1 to n − j + 1 do
        foreach A ∈ 𝒩_ij do
            foreach A → BC ∈ 𝒢 do
                for k ← 1 to j − 1 do
                    if B ∈ 𝒩_{i,k} ∧ C ∈ 𝒩_{i+k,j−k} then
                        f_ikB += f_ijA × w_{i+k,j−k,C}
                        f_{i+k,j−k,C} += f_ijA × w_ikB
for i ← 1 to n do
    // Clear marginals
    foreach d ∈ D(X_i) do
        θ_{X_i}(d) ← 0
    // Add forks to marginals
    foreach A ∈ 𝒩_i1 do
        foreach A → a ∈ 𝒢 do
            θ_{X_i}(a) += f_i1A
return θ
```

solver	nb solved	btk-free	fails	time(s)
CP	52	0	333.2	231.2
CPBP	**72**	**25**	**8.1**	**175.3**
CPBP$^-$	49	3	9.9	298.2

Table 1: Comparing solvers on 81 constrained molecule generation instances.

of donors and acceptors respectively.

Table 1 presents the number of instances for which a molecule was successfully generated (out of 81), the number of times that generation was backtrack-free, and the average number of failed search-tree nodes and computation time (over solved instances), using different solvers. Our baseline CP solver performs fairly well but it is clearly dominated by our full CPBP solver taking advantage of superior search guidance through the PMFs over domains obtained by belief propagation. The latter solves 20 more instances, 25 overall without any backtrack, with a much lower average number of failed nodes and a lower average computation time despite the overhead of running BP at each search-tree node. The comparison with CPBP$^-$ shows the significant impact of our Algorithm 1 on search guidance.

Figure 5 plots all solved instances according to failed nodes and time. We see a very clear separation in terms of search guidance (fails) due to BP, at the expense of a slightly steeper slope in terms of computation time. Overall these results show that constrained molecule generation is achievable with such a CP model, although modelling using a CFG constraint with its cubic-time filtering algorithm and with such a large grammar comes at a computational cost: even a backtrack-free generation took more than one minute.

Figure 6 shows one of the molecules we generated. Of course, this in no way guarantees that it would hold any medicinal virtue or be easily synthesized, but it exemplifies how we can concentrate on candidates showing some potential by restricting the design space to desired properties.

7 Conclusion

We presented a promising application of the grammar constraint in CP — constrained molecule generation — and a novel weighted counting algorithm for this constraint which allowed us to solve the problem much more efficiently. Because so few candidate molecules are ultimately retained, our ability to model higher-level properties of molecules as constraints coupled with a CPBP solver gives us a considerable computational advantage over a generate-and-test approach by ruling out whole regions of the design space and by guiding its exploration. Such targeted *in silico* generation offers the promise of increasing the success rate of a lengthy and costly downstream process.

Compared to deep learning models, combinatorial solvers have the advantage of guaranteeing that the generated sequence will exhibit the prescribed properties. But the former also learn from large databases of successful molecules and so neurosymbolic architectures combining the two approaches may offer both property guarantees and proven recipes.

Acknowledgements

This research was made possible through funding from NSERC Discovery grant RGPIN-2023-05705.

References

Akhmetshin, T.; Lin, A. I.; Mazitov, D.; Ziaikin, E.; Madzhidov, T.; and Varnek, A. 2021. ZINC 250K data sets.

Babaki, B.; Omrani, B.; and Pesant, G. 2020. Combinatorial Search in CP-Based Iterated Belief Propagation. In Simonis, H., ed., *Principles and Practice of Constraint Programming - 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7-11, 2020, Proceedings*, volume 12333 of *Lecture Notes in Computer Science*, 21–36. Springer.

Boussemart, F.; Hemery, F.; Lecoutre, C.; and Sais, L. 2004. Boosting Systematic Search by Weighting Constraints. In de Mántaras, R. L.; and Saitta, L., eds., *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, 146–150. IOS Press.

Burlats, A.; and Pesant, G. 2023. Exploiting Entropy in Constraint Programming. In Ciré, A. A., ed., *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 20th International Conference, CPAIOR 2023, Nice, France, May 29 - June 1, 2023, Proceedings*, volume 13884 of *Lecture Notes in Computer Science*, 320–335. Springer.

Carissan, Y.; Hagebaum-Reignier, D.; Prcovic, N.; Terrioux, C.; and Varet, A. 2022. How constraint programming can help chemists to generate Benzenoid structures and assess the local Aromaticity of Benzenoids. *Constraints An Int. J.*, 27(3): 192–248.

Charpentier, A.; Mignon, D.; Barbe, S.; Cortés, J.; Schiex, T.; Simonson, T.; and Allouche, D. 2019. Variable Neighborhood Search with Cost Function Networks To Solve Large Computational Protein Design Problems. *J. Chem. Inf. Model.*, 59(1): 127–136.

Du, Y.; Fu, T.; Sun, J.; and Liu, S. 2022. MolGenSurvey: A Systematic Survey in Machine Learning Models for Molecule Design. arXiv:2203.14500.

Guo, M.; Thost, V.; Li, B.; Das, P.; Chen, J.; and Matusik, W. 2022. Data-Efficient Graph Grammar Learning for Molecular Generation. In *International Conference on Learning Representations*.

Harvey, W. D.; and Ginsberg, M. L. 1995. Limited Discrepancy Search. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*, 607–615. Morgan Kaufmann.

Kraev, E. 2018. Grammars and reinforcement learning for molecule optimization. arXiv:1811.11222.

Krenn, M.; Häse, F.; Nigam, A.; Friederich, P.; and Aspuru-Guzik, A. 2020. Self-referencing embedded strings (SELFIES): A 100% robust molecular string representation. *Mach. Learn. Sci. Technol.*, 1(4): 45024.

Krippahl, L.; and Barahona, P. 1999. Applying Constraint Programming to Protein Structure Determination. In Jaffar, J., ed., *Principles and Practice of Constraint Programming - CP'99, 5th International Conference, Alexandria, Virginia, USA, October 11-14, 1999, Proceedings*, volume 1713 of *Lecture Notes in Computer Science*, 289–302. Springer.

Krippahl, L.; and Barahona, P. 2015. Protein docking with predicted constraints. *Algorithms Mol. Biol.*, 10: 9.

Lafleur, D.; Chandar, S.; and Pesant, G. 2022. Combining Reinforcement Learning and Constraint Programming for Sequence-Generation Tasks with Hard Constraints. In Solnon, C., ed., *28th International Conference on Principles and Practice of Constraint Programming, CP 2022, July 31 to August 8, 2022, Haifa, Israel*, volume 235 of *LIPIcs*, 30:1–30:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Lipinski, C. A.; Lombardo, F.; Dominy, B. W.; and Feeney, P. J. 1997. Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings. *Advanced Drug Delivery Reviews*, 23(1): 3–25. In Vitro Models for Selection of Development Candidates.

Manibod, V.; Saikali, D.; and Pesant, G. 2025. Constrained Sequential Inference in Machine Learning Using Constraint Programming. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2025, 16th-22th August 2025, Montreal, Canada*. ijcai.org.

Omrani, M. A.; and Naanaa, W. 2020. Constraints for generating graphs with imposed and forbidden patterns: an application to molecular graphs. *Constraints An Int. J.*, 25(1-2): 1–22.

Peng, X.; and Solnon, C. 2023. Using Canonical Codes to Efficiently Solve the Benzenoid Generation Problem with Constraint Programming. In Yap, R. H. C., ed., *29th International Conference on Principles and Practice of Constraint Programming, CP 2023, August 27-31, 2023, Toronto, Canada*, volume 280 of *LIPIcs*, 28:1–28:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Pesant, G. 2019. From Support Propagation to Belief Propagation in Constraint Programming. *Journal of Artificial Intelligence Research*.

Polishchuk, P. G.; Madzhidov, T. I.; and Varnek, A. 2013. Estimation of the size of drug-like chemical space based on GDB-17 data. *Journal of Computer-Aided Molecular Design*.

Polykovskiy, D.; et al. 2020. Molecular Sets (MOSES): A Benchmarking Platform for Molecular Generation Models. *Frontiers in Pharmacology*, 11. ISSN: 1663-9812.

Quimper, C.; and Walsh, T. 2006. Global Grammar Constraints. In Benhamou, F., ed., *Principles and Practice of Constraint Programming - CP 2006, 12th International Conference, CP 2006, Nantes, France, September 25-29, 2006, Proceedings*, volume 4204 of *Lecture Notes in Computer Science*, 751–755. Springer.

Sellmann, M. 2006. The Theory of Grammar Constraints. In Benhamou, F., ed., *Principles and Practice of Constraint Programming - CP 2006, 12th International Conference, CP*

2006, Nantes, France, September 25-29, 2006, Proceedings, volume 4204 of *Lecture Notes in Computer Science*, 530–544. Springer.

Varet, A.; Prcovic, N.; Terrioux, C.; Hagebaum-Reignier, D.; and Carissan, Y. 2022. BenzAI: A Program to Design Benzenoids with Defined Properties Using Constraint Programming. *J. Chem. Inf. Model.*, 62(11): 2811–2820.

Vidal, D.; Thormann, M.; and Pons, M. 2005. LINGO, an Efficient Holographic Text Based Method To Calculate Biophysical Properties and Intermolecular Similarities. *Journal of Chemical Information and Modeling*, 45(2): 386–393.

Vucinic, J.; Simoncini, D.; Ruffini, M.; Barbe, S.; and Schiex, T. 2020. Positive multistate protein design. *Bioinform.*, 36(1): 122–130.

Weininger, D. 1988. SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences*, 28(1): 31–36.

Wikimedia. 2007. SMILES.png. Https://commons.wikimedia.org/wiki/File:SMILES.png [Accessed: 2023-07-12].

Yin, C.; Cappart, Q.; and Pesant, G. 2024. An Improved Neuro-Symbolic Architecture to Fine-Tune Generative AI Systems. In Dilkina, B., ed., *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 21st International Conference, CPAIOR 2024, Uppsala, Sweden, May 28-31, 2024, Proceedings, Part II*, volume 14743 of *Lecture Notes in Computer Science*, 279–288. Springer.

Yin, C.; Cappart, Q.; and Pesant, G. 2025. Shaping Reward Signals in Reinforcement Learning Using Constraint Programming. In Tack, G., ed., *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 22nd International Conference, CPAIOR 2025, Melbourne, VIC, Australia, November 10-13, 2025, Proceedings, Part II*, volume 15763 of *Lecture Notes in Computer Science*, 252–269. Springer.