Constrained Molecule Generation Modelled using the Grammar Constraint

David Saikali ✉
Department of Computer and Software Engineering, Polytechnique Montréal, Montreal, Canada

Gilles Pesant ✉
Department of Computer and Software Engineering, Polytechnique Montréal, Montreal, Canada

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 that some low-level target properties such as molecular weight and structural features (cycles, branches) 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 valid molecules.

2012 ACM Subject Classification Mathematics of computing → Solvers; Computing methodologies → Discrete space search; Applied computing → Computational biology

Keywords and phrases grammar constraint, drug discovery, SMILES, weighted counting, belief propagation

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 [9] — 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 [8]. 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. 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 [23, 22], 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 achieve effective search guidance; iii) empirical evidence that some important real-life problems may be solved much more efficiently with it.

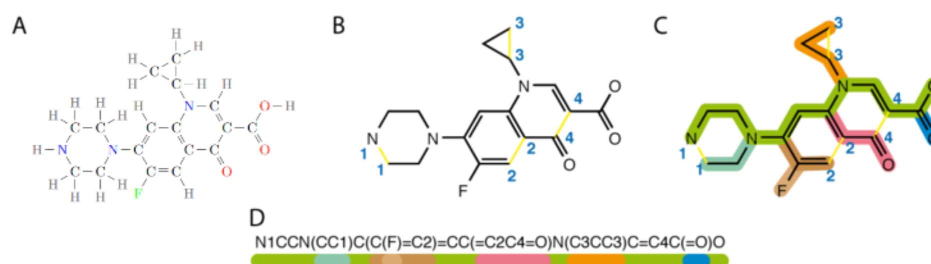

Figure 1 Deriving a SMILES representation for a molecule (reproduced in part from [7]). The structural formula of the molecule (A), its skeletal formula stripped of all hydrogen atoms and with broken cycles (B), the selected main path (shown in green) and branches (C), and the corresponding SMILES notation (D).

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).

2.1 SMILES Representation format

SMILES (Simplified Molecular-Input Line-Entry System) [26] 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 [13] 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 [14]) but their use is not nearly as widespread.

2.2 Lipinski's Rule of 5

Lipinski's rule of 5 [18] describes four physicochemical properties that molecules fit to be orally active drugs in humans tend to respect. 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 bond donors: fewer than 5 hydrogen-bond donors and fewer than 10 hydrogen-bond acceptors. These are bonds between hydrogen atoms and an electronegative atom such as nitrogen, oxygen, or fluorine. Finally, the last rule has to do with how hydrophilic or lipophilic (i.e. hydrophobic) a molecule is. The higher the logP score, the more lipophilic it is. Lipinski's rule of five says that the logP score of a viable molecule should not exceed 5. These rules are heuristic since several exceptions can be found.

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 respectively by a set of nonterminal symbols, a set of terminal symbols (its alphabet), a set of production rules, and a start symbol. We denote $L(\mathcal{G})$ the language recognized by $\mathcal{G}$ i.e. the set of strings that grammar can generate. According to Chomsky's classification, there are many types of grammars, ranging from least to most restrictive: Recursively Enumerable (Type-0), Context-Sensitive (Type-1), Context-Free (Type-2) and Regular (Type-3). For a grammar to qualify as context-free, its production rules must respect two restrictions: 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.

▶ **Example 1.** Context-free grammar $\mathcal{G} = (\{S, A, B, C\}, \{\langle, \rangle\}, \{S \rightarrow SS, S \rightarrow AC, S \rightarrow BC, B \rightarrow AS, A \rightarrow \langle, C \rightarrow \rangle\}, S)$ recognizes correctly bracketed words such as "$\langle\langle\rangle\rangle$", obtained by the successive application of rules: $S \rightarrow BC \rightarrow ASC \rightarrow AS\rangle \rightarrow AAC\rangle \rightarrow A\langle C\rangle \rightarrow A\langle\rangle\rangle \rightarrow \langle\langle\rangle\rangle$. 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 3).

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 [22] 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 $A \rightarrow BC$ or $A \rightarrow a$ 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 [21]. 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 [2, 4] and to train neural networks [17, 27].

3 Related Work

Drug discovery, and molecule design in general, is a vast topic. A recent survey by Du et al. [8] 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 [15]. 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 [16] 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 [25, 6]. 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. [5, 24] 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 [20] improve the enumeration of benzenoid graphs by representing them

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

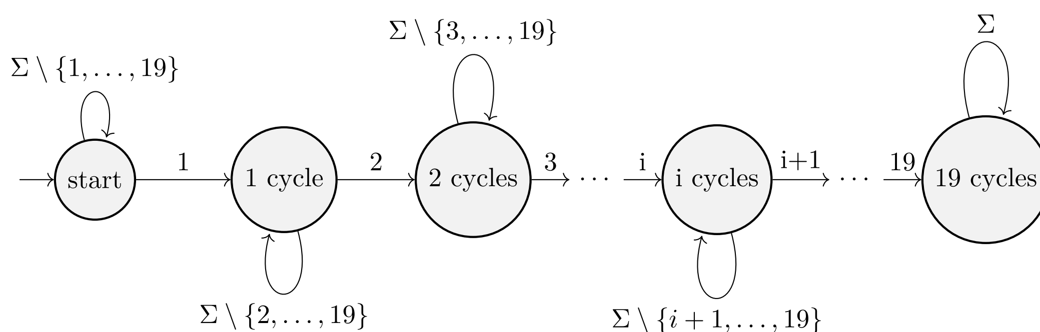

Figure 2 Automaton $\mathcal{A}$ which imposes ordinal order on cycle numbering.

164 using short canonical codes that are invariant to symmetries and rotations, expressed in a
165 CP model. They ensure the presence of a given pattern by completing a suitably prefixed
166 code. The sequential nature of these codes, obtained through graph traversal, makes them
167 similar in spirit to the SMILES notation, though much less general.

168 In the context of their work on constrained graph generation using CP, Omrani and
169 Naanaa [19] consider the generation of molecular graphs corresponding to a given molecular
170 formula.

171 So despite some prior work involving CP, none address the problem we consider and
172 especially the use of the grammar constraint. On a related note we end by mentioning the
173 work of Guo et al. [11] who recently proposed a sample-efficient neural method for molecule
174 generation that is based on learning a graph grammar.

175 4 Model

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

178 4.1 SMILES representation

179 As mentioned earlier, we use a variation of Kraev's grammar [13] to ensure that atom valences
180 are respected in the generated molecules. One of the modifications we made to the grammar
181 was to integrate the cycle length limit directly into the grammar, something Kraev [13] did
182 using a mask. We limit the cycle length to 8: MOSES [10], a data set of about two million
183 molecules, never exceeds length-6 cycles while another, Zinc_250k [1], features some length-8
184 cycles. The resulting context-free grammar features 159 productions, 49 nonterminal symbols
185 and 45 terminal symbols (the SMILES alphabet). Its conversion into Chomsky normal form
186 features the same number of terminals while the number of productions and nonterminals
187 increase to 555 and 172 respectively.

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

$$\textsc{Cfg}(\langle X_1, X_2, \dots, X_n \rangle, \mathcal{G}_{\text{SMILES}})$$

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

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

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:

$$\textsc{Among}(\langle X_1, X_2, \ldots, X_n \rangle, \{j\}, \{0, 2\}) \quad 1 \leq j \leq 8.$$

188 These conventions may be seen as applying static symmetry breaking.
189 In principle we could combine all the constraints of this section into a single Cfg constraint
190 but at the expense of a significant increase in size of an already large grammar.

191 4.2 Targeting Regions of the Design Space

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

To limit the *molecular weight*, which is one of Lipinski's four rules, we first define a table $\mathcal{T}$ 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 $\mathcal{T}$ to compensate for the missing hydrogen atoms. We also associate a negative weight to the tokens representing double bonds, triple bonds and cycle numbers. This is used to counteract the increased weight of atoms since we include the implicit hydrogen atoms. The error on the molecular weight does not exceed 5% when tested on the large datasets. We then index that table with variables X_i,

$$\textsc{Element}(\mathcal{T}, X_i, W_i) \quad 1 \leq i \leq n$$

linking each with an individual weight variable W_i, and sum them to obtain the weight W of the whole molecule:

$$\textsc{Sum}(\langle W_1, W_2, \ldots, W_n \rangle, W).$$

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:

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

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

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

If we want at most n_c, we add

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

194 To require an exact number we use both.

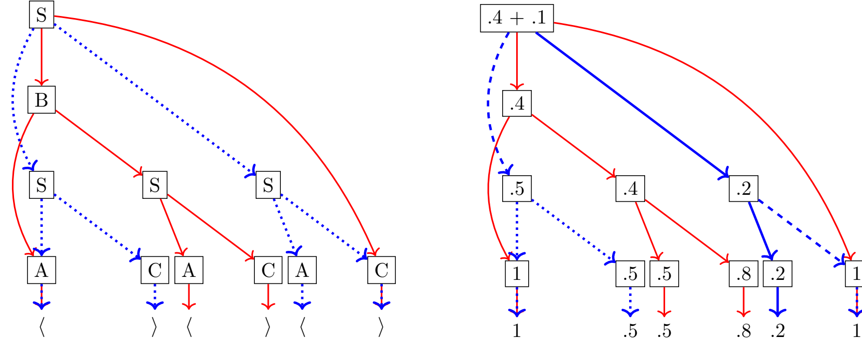

Figure 3 Two parse trees (left) for the two words of length 4 recognized by the grammar of Example 1 and the computed weights w_{ijN} at parse subtrees (right) given some b_{id} values at the bottom. The path in bold from the root to "⟨" in position 3 illustrates the computation of $\theta_{X_3}(\text{``}\langle\text{''}) = f_{31A} = w_{12S} \times w_{41C} = .5$ as the product of the weights of the two branches off that path (shown dashed) in the blue dotted parse tree.

5 Weighted Counting Algorithm for the CFG Constraint

In this section, we design a dedicated algorithm for the grammar constraint which computes marginals for variable-value pairs 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 3 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, but determining whether an arbitrary context-free grammar is ambiguous is undecidable. 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.

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
1  for i ← 1 to n do
2  | for j ← 1 to n − i do
3  | | foreach N ∈ 𝒩 do
4  | | | w_ijN ← 0
   // Initialize weights for length-one substrings
5  foreach A → a ∈ 𝒢 do
6  | for i ← 1 to n do
7  | | if A ∈ 𝒩_i1 then
8  | | | w_i1A ← w_i1A + b_ia
   // Consider substrings of increasing length, accumulating weights
9  for j ← 2 to n do
10 | for i ← 1 to n − j + 1 do
11 | | for k ← 1 to j − 1 do
12 | | | foreach B ∈ 𝒩_ik do
13 | | | | foreach A → BC ∈ 𝒢 do
14 | | | | | if C ∈ 𝒩_{i+k,j−k} then
15 | | | | | | w_ijA ← w_ijA + w_ikB × w_{i+k,j−k,C}
   // Clear forks
16 for i ← 1 to n do
17 | for j ← 1 to n − i do
18 | | foreach N ∈ 𝒩 do
19 | | | f_ijN ← 0
   // Initialize root of all parse trees (start symbol)
20 f_1nS ← 1
   // Consider substrings of decreasing length, accumulating the product
      of weights that branch off on either side
21 for j ← n down to 2 do
22 | for i ← 1 to n − j + 1 do
23 | | foreach A ∈ 𝒩_ij do
24 | | | foreach A → BC ∈ 𝒢 do
25 | | | | for k ← 1 to j − 1 do
26 | | | | | if B ∈ 𝒩_{i,k} ∧ C ∈ 𝒩_{i+k,j−k} then
27 | | | | | | f_ikB ← f_ikB + f_ijA × w_{i+k,j−k,C}
28 | | | | | | f_{i+k,j−k,C} ← f_{i+k,j−k,C} + f_ijA × w_ikB
29 for i ← 1 to n do
   | // Clear marginals
30 | foreach d ∈ D(X_i) do
31 | | θ_{X_i}(d) ← 0
   | // Add accumulated forks to marginals
32 | foreach A ∈ 𝒩_i1 do
33 | | foreach A → a ∈ 𝒢 do
34 | | | θ_{X_i}(a) ← θ_{X_i}(a) + f_i1A
35 return θ
```

instance	marginalStr time(s)	marginalStr fails	marginalStrLDS time(s)	marginalStrLDS fails	domWDeg/minVal time(s)	domWDeg/minVal fails	domWdeg/random time(s)	domWdeg/random fails	dom/random time(s)	dom/random fails
c1b2	6.2	0	6.8	0	123.0	1862	6.1	6	6.0	44
c1b3	4.2	0	5.2	0	–	–	5.9	13	–	–
c1b4	5.9	1	6.3	1	–	–	5.9	17	–	–
c2b2	4.9	0	4.9	0	–	–	23.8	826	231.1	24161
c2b3	4.8	0	5.4	0	–	–	7.7	171	49.7	6065
c2b4	5.9	0	6.0	0	–	–	10.8	569	–	–
c3b2	7.3	0	7.3	0	–	–	–	–	–	–
c3b3	–	–	79.6	93	–	–	–	–	–	–
c3b4	–	–	12.7	17	–	–	–	–	–	–

Table 1 Comparing branching heuristics on some constrained molecule generation instances.

6 Results

The double purpose of our experiment will be to show how useful the weighted counting algorithm we contributed for the grammar constraint is to guide search compared to likely alternative branching heuristics, and to show that generating potentially useful molecules from a constrained design space modeled in CP using a grammar constraint appears to be tractable. Therefore our experiment is restricted to a CP approach using the model we described in Section 4 and we compare runtime and number of search-tree failures. A comparison with other computational approaches on the same basis would only yield a very partial picture anyway. Ultimately a comparative study would involve testing many more properties, some only possible by attempting to synthesize the molecule in a lab, in order to determine how good of a candidate it is. For example, a faster approach may generate mostly useless molecules and hence be slower in confirming a good one. The ability to enforce or at least encourage many properties at the time of *in silico* generation, such as with CP, offers the promise of increasing the success rate of a lengthy and costly downstream process.

According to the classification of Du et al. [8], 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 will seek molecules very close to the recommended limit according to Lipinski's Rule of 5 for molecular weight: between 475 and 500 Da (i.e. we add constraint $475 \leq W \leq 500$ to our model). At the same time, we will ask for specific structural features, considering every combination of number of cycles and branches in the respective ranges 1..3 and 2..4, instance cibj corresponding to i cycles and j branches (in which case we add $N_b == j$ to our model and set n_c to i in the corresponding pair of AMONG constraints). We set $n = 40$.

Table 1 presents the computation time (on an AMD Rome 7532 processor (2.4GHz, 256M cache L3), 1 GB of RAM, and allowing a maximum of one hour) and number of fails to find a first solution to each of our instances using different branching heuristics. The tests were run using the MiniCPBP solver.

Being a learning-based heuristic, `domWDeg` [3] is run with restarts (initially after 100 fails and increased by a 1.5 factor), which is common practice for it. Early experiments on our instances confirmed that it generally performs better with restarts than without. We report on its combination with two value-selection heuristics: `minVal`, which selects the smallest value in the domain, and `random`, which selects a domain value uniformly at random. In the latter case we report the median of 11 runs.

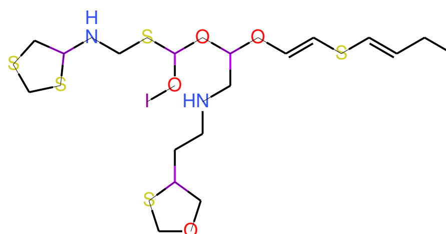

Figure 4 Molecule "IOC(OC(OC=CSC=CCC)CNCCC1SCOC1)SCNC2CSCS2" generated by `maxMarginalStrength` for instance `c2b2`. Regarding Lipinski's Rule of 5, it features 2 hydrogen-bond donors, 11 hydrogen-bond acceptors, and a logP score of about 5.2.

256 `dom/random` selects a variable with the smallest domain and a domain value uniformly at
257 random. Here as well we report the median of 11 runs.

258 `maxMarginalStrength` [21] (identified as `marginalStr` in the table) is a branching heur-
259 istic based on the marginals computed by MiniCPBP using the weighted counting algorithm
260 of each constraint in the model, including the one we newly designed for CFG. Because it is
261 not learning-based and is deterministic, using restarts would not help. We report on its use
262 with standard depth-first search (DFS) and also with limited-discrepancy search (LDS, with
263 a maximum number of discrepancies starting at 1 and doubled at each iteration, ultimately
264 making the search complete), which is a sensible option for a trusted branching heuristic [12].

265 Our baseline branching heuristic `dom/random` is only able to solve three out of nine
266 instances within the one-hour time limit. Even with restarts, `domWDeg` struggles with the
267 usual value selection `minVal` but does better with `random` (solving 6 out of 9), hinting that
268 value selection is quite important for this problem with large domains and that the smallest
269 value may not be a particularly good choice.

270 Branching heuristics based on marginals make an integrated choice of variable and value.
271 The very low number of fails for `maxMarginalStrength` (6 out of 9 instances are solved
272 backtrack-free) is remarkable and shows the usefulness of the weighted counting algorithm we
273 designed for the CFG constraint. It does not manage to solve the last two instances within
274 the time limit, likely because of a bad decision near the top of the tree. Using LDS instead
275 of DFS confirms this as all instances then become solved.

276 Although very convenient and overall effective, modelling using a CFG constraint with a
277 large grammar comes at a computational cost: posting it (including the initial call to its
278 propagator) takes about a second. Running several iterations of belief propagation (including
279 the weighted counting algorithm for CFG) before branching takes three to four times longer
280 than a branching heuristic such as `domWDeg`. However these are offset by the superior search
281 guidance, and thus much smaller search tree, it brings.

282 Out of curiosity, Figure 4 shows one of the molecules we generated, which is not too far
283 from the recommended values according to Lipinski's Rule of 5. Of course, this in no way
284 guarantees that the molecule would satisfy all the other requirements, or that it would hold
285 any medicinal virtue.

286 7 Conclusion

287 We presented a promising application of the grammar constraint — constrained molecule
288 generation — and a novel weighted counting algorithm for this constraint which allowed us
289 to solve the problem more efficiently. Because so few candidate molecules are ultimately

retained, an ongoing challenge is being able to model higher-level properties of molecules as constraints. By actively restricting the design space during generation, it would give us a considerable computational advantage over a generate-and-test approach. Expressing the whole design space in CP allows us to explore little-known regions in that space but we also wish to exploit our knowledge base of successful molecules. Combining CP and machine learning may help us reach a balance between exploration and exploitation and we are currently investigating such a mix.

References

1. Tagir Akhmetshin, Arkadii I. Lin, Daniyar Mazitov, Evgenii Ziaikin, Timur Madzhidov, and Alexandre Varnek. ZINC 250K data sets. 12 2021. URL: `https://figshare.com/articles/dataset/ZINC_250K_data_sets/17122427`, `doi:10.6084/m9.figshare.17122427.v1`.
2. Behrouz Babaki, Bilel Omrani, and Gilles Pesant. Combinatorial search in cp-based iterated belief propagation. In Helmut Simonis, editor, *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*, pages 21–36. Springer, 2020. `doi:10.1007/978-3-030-58475-7_2`.
3. Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. In Ramón López de Mántaras and Lorenza Saitta, editors, *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*, pages 146–150. IOS Press, 2004.
4. Auguste Burlats and Gilles Pesant. Exploiting entropy in constraint programming. In André A. Ciré, editor, *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*, pages 320–335. Springer, 2023. `doi:10.1007/978-3-031-33271-5_21`.
5. Yannick Carissan, Denis Hagebaum-Reignier, Nicolas Prcovic, Cyril Terrioux, and Adrien Varet. 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, 2022. URL: `https://doi.org/10.1007/s10601-022-09328-x`, `doi:10.1007/S10601-022-09328-X`.
6. Antoine Charpentier, David Mignon, Sophie Barbe, Juan Cortés, Thomas Schiex, Thomas Simonson, and David Allouche. Variable neighborhood search with cost function networks to solve large computational protein design problems. *J. Chem. Inf. Model.*, 59(1):127–136, 2019. URL: `https://doi.org/10.1021/acs.jcim.8b00510`, `doi:10.1021/ACS.JCIM.8B00510`.
7. Wikimedia Commons. Smiles.png. Online, accessed July 12, 2023. URL: `https://commons.wikimedia.org/wiki/File:SMILES.png`.
8. Yuanqi Du, Tianfan Fu, Jimeng Sun, and Shengchao Liu. Molgensurvey: A systematic survey in machine learning models for molecule design, 2022. `arXiv:2203.14500`.
9. Polishchuk et al. Estimation of the size of drug-like chemical space based on gdb-17 data. *Journal of Computer-Aided Molecular Design*, 2013. `doi:10.1007/s10822-013-9672-4`.
10. Polykovskiy et al. Molecular sets (moses): A benchmarking platform for molecular generation models. *Frontiers in Pharmacology*, 11, 2020. ISSN: 1663-9812. `arXiv:https://doi.org/10.3389/fphar.2020.565644`, `doi:10.3389/fphar.2020.565644`.
11. Minghao Guo, Veronika Thost, Beichen Li, Payel Das, Jie Chen, and Wojciech Matusik. Data-efficient graph grammar learning for molecular generation. In *International Conference on Learning Representations*, 2022. URL: `https://openreview.net/forum?id=l4IHywGq6a`.
12. William D. Harvey and Matthew L. Ginsberg. 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*, pages 607–615. Morgan Kaufmann, 1995. URL: `http://ijcai.org/Proceedings/95-1/Papers/080.pdf`.

340 **13** Egor Kraev. Grammars and reinforcement learning for molecule optimization, 2018. arXiv:
341 1811.11222.

342 **14** Mario Krenn, Florian Häse, AkshatKumar Nigam, Pascal Friederich, and Alán Aspuru-Guzik.
343 Self-referencing embedded strings (SELFIES): A 100% robust molecular string representation.
344 *Mach. Learn. Sci. Technol.*, 1(4):45024, 2020. URL: https://doi.org/10.1088/2632-2153/
345 aba947, doi:10.1088/2632-2153/ABA947.

346 **15** Ludwig Krippahl and Pedro Barahona. Applying constraint programming to protein structure
347 determination. In Joxan Jaffar, editor, *Principles and Practice of Constraint Programming*
348 *- CP'99, 5th International Conference, Alexandria, Virginia, USA, October 11-14, 1999,*
349 *Proceedings*, volume 1713 of *Lecture Notes in Computer Science*, pages 289–302. Springer,
350 1999. doi:10.1007/978-3-540-48085-3_21.

351 **16** Ludwig Krippahl and Pedro Barahona. Protein docking with predicted constraints. *Algorithms*
352 *Mol. Biol.*, 10:9, 2015. URL: https://doi.org/10.1186/s13015-015-0036-6, doi:10.1186/
353 S13015-015-0036-6.

354 **17** Daphné Lafleur, Sarath Chandar, and Gilles Pesant. Combining reinforcement learn-
355 ing and constraint programming for sequence-generation tasks with hard constraints. In
356 Christine Solnon, editor, *28th International Conference on Principles and Practice of Con-*
357 *straint Programming, CP 2022, July 31 to August 8, 2022, Haifa, Israel*, volume 235 of
358 *LIPIcs*, pages 30:1–30:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. URL:
359 https://doi.org/10.4230/LIPIcs.CP.2022.30, doi:10.4230/LIPICS.CP.2022.30.

360 **18** Christopher A. Lipinski, Franco Lombardo, Beryl W. Dominy, and Paul J. Feeney. Experimental
361 and computational approaches to estimate solubility and permeability in drug discovery and
362 development settings. *Advanced Drug Delivery Reviews*, 23(1):3–25, 1997. In Vitro Models
363 for Selection of Development Candidates. URL: https://www.sciencedirect.com/science/
364 article/pii/S0169409X96004231, doi:10.1016/S0169-409X(96)00423-1.

365 **19** Mohamed Amine Omrani and Wady Naanaa. Constraints for generating graphs with
366 imposed and forbidden patterns: an application to molecular graphs. *Constraints An*
367 *Int. J.*, 25(1-2):1–22, 2020. URL: https://doi.org/10.1007/s10601-019-09305-x, doi:
368 10.1007/S10601-019-09305-X.

369 **20** Xiao Peng and Christine Solnon. Using canonical codes to efficiently solve the benzenoid
370 generation problem with constraint programming. In Roland H. C. Yap, editor, *29th Interna-*
371 *tional Conference on Principles and Practice of Constraint Programming, CP 2023, August*
372 *27-31, 2023, Toronto, Canada*, volume 280 of *LIPIcs*, pages 28:1–28:17. Schloss Dagstuhl -
373 Leibniz-Zentrum für Informatik, 2023. URL: https://doi.org/10.4230/LIPIcs.CP.2023.28,
374 doi:10.4230/LIPICS.CP.2023.28.

375 **21** Gilles Pesant. From support propagation to belief propagation in constraint programming.
376 *Journal of Artificial Intelligence Research*, 2019. doi:10.1613/jair.1.11487.

377 **22** Claude-Guy Quimper and Toby Walsh. Global grammar constraints. In Frédéric Benhamou,
378 editor, *Principles and Practice of Constraint Programming - CP 2006, 12th International*
379 *Conference, CP 2006, Nantes, France, September 25-29, 2006, Proceedings*, volume 4204 of
380 *Lecture Notes in Computer Science*, pages 751–755. Springer, 2006. doi:10.1007/11889205\
381 _64.

382 **23** Meinolf Sellmann. The theory of grammar constraints. In Frédéric Benhamou, editor, *Principles*
383 *and Practice of Constraint Programming - CP 2006, 12th International Conference, CP 2006,*
384 *Nantes, France, September 25-29, 2006, Proceedings*, volume 4204 of *Lecture Notes in Computer*
385 *Science*, pages 530–544. Springer, 2006. doi:10.1007/11889205_38.

386 **24** Adrien Varet, Nicolas Prcovic, Cyril Terrioux, Denis Hagebaum-Reignier, and Yannick Carissan.
387 Benzai: A program to design benzenoids with defined properties using constraint programming.
388 *J. Chem. Inf. Model.*, 62(11):2811–2820, 2022. URL: https://doi.org/10.1021/acs.jcim.
389 2c00353, doi:10.1021/ACS.JCIM.2C00353.

390 **25** Jelena Vucinic, David Simoncini, Manon Ruffini, Sophie Barbe, and Thomas Schiex. Positive
391 multistate protein design. *Bioinform.*, 36(1):122–130, 2020. URL: `https://doi.org/10.1093/`
392 `bioinformatics/btz497`, `doi:10.1093/BIOINFORMATICS/BTZ497`.
393 **26** David Weininger. Smiles, a chemical language and information system. 1. introduction to
394 methodology and encoding rules. *Journal of Chemical Information and Computer Sciences*,
395 28(1):31–36, 1988. `doi:10.1021/ci00057a005`.
396 **27** Chao Yin, Quentin Cappart, and Gilles Pesant. An improved neuro-symbolic architecture to
397 fine-tune generative AI systems. In Bistra Dilkina, editor, *Integration of Constraint Program-*
398 *ming, Artificial Intelligence, and Operations Research - 21st International Conference, CPAIOR*
399 *2024, Uppsala, Sweden, May 28-31, 2024, Proceedings, Part II*, volume 14743 of *Lecture Notes*
400 *in Computer Science*, pages 279–288. Springer, 2024. `doi:10.1007/978-3-031-60599-4_19`.