POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Titre de mon document / Title

DAVID SAIKALI
Département de Génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Juillet 2025

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Titre de mon document / Title

présenté par **David SAIKALI**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Amal ZOUAQ, présidente
Gilles PESANT, membre et directeur de recherche
Louis-Martin ROUSSEAU, membre

DEDICATION

À tous mes amis du labos,
vous me manquerez...

ACKNOWLEDGEMENTS

Texte / Text.

RÉSUMÉ

La recherche de nouvelles molécules est un processus sans fin.

Le résumé est un bref exposé du sujet traité, des objectifs visés, des hypothèses émises, des méthodes expérimentales utilisées et de l'analyse des résultats obtenus. On y présente également les principales conclusions de la recherche ainsi que ses applications éventuelles. En général, un résumé ne dépasse pas trois pages.

Le résumé doit donner une idée exacte du contenu du mémoire ou de la thèse. Ce ne peut pas être une simple énumération des parties du manuscrit. Le but est de présenter de façon précise et concise la nature, l'envergure de la recherche, les sujets traités, les questions de recherche ou les hypothèses soulevées, les méthodes utilisées, les principaux résultats ainsi que les conclusions retenues. Un résumé ne doit jamais comporter de références ou de figures.

ABSTRACT

This thesis goes over our efforts to represent drug-like molecules using Constraint Programming.

David: Ajoute une phrase de motivation, soit avant soit après.

We also attempt to evaluate desirable properties to guide our results towards potentially useful molecules. To try and improve the realism of generated molecules, we combine Machine Learning, specifically Natural Language Processing, and Constraint Programming. We use perplexity and the success rate to evaluate our model's quality. This allows us to weigh the different constraints against the information learned by the model. If our work shows promise, we believe it could be useful in other domains to apply long-term structure in long sequence generation.

Written in English, the abstract is a brief summary similar to the previous section (Résumé). However, this section is not a word for word translation of the abstract in French.

The abstract is a brief statement of the subject matter, objectives, research questions or hypotheses, experimental methods and analysis of results. It also presents the main research conclusions and their possible applications. In general, an abstract should not exceed three pages.

The abstract should provide an exact idea of the thesis or dissertation's contents and it cannot be a simple enumeration of the manuscript's parts. The goal is to precisely and concisely present the nature and scope of the research. An abstract should never include references or figures. If the thesis or the dissertation is in English, the résumé (French-language abstract) should come first followed by the abstract.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

Drug discovery is a very time-consuming and costly endeavor due to its enormous design space — estimated to contain between 10^{23} and 10^{60} different molecules [1] — 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 [2]. It aims to extract from this huge design space the most likely candidates according to some desired properties. 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 (NLP), such as sequential generative neural models. Large Language Models (LLMs) in particular have come to the forefront of popular attention as impressive tools for generating text. However, this generation isn't limited to purely human languages as we can train the model on another text format, such as SMILES, and get a model capable of generating molecules.

SMILES also lends itself very well to Constraint Programming (CP). CP seems like a natural approach to molecule discovery since it allows hard constraints to be placed which could ensure only valid molecules are generated. Using a Context-Free Grammar (CFG) and a few additional constraints could allow us to describe valid SMILES strings in a CP model. We also believe it may be possible to model desirable molecular properties using CP, this would allow our model to restrict its search even further. This allows us to explore the huge design space of possible molecules while adding constraints in order to restrict that space to suitable candidates.

This CP approach, which excels at imposing hard rules and long-term structure while lacking the informed decision making that trained models gain from the dataset used, could allow us to answer one of the issues with sequence models in Machine Learning (ML). Often times, these models struggle to exhibit long-term structure, stemming in part from the token-by-token nature of the prediction process used to generate a sequence. In other words, these models do not explicitly learn the hard rules that determine validity nor desirability and merely mimic what was observed.

David: Ce texte, qui vient de l'article IJCAI, convient moins bien ici. On ne parle pas de propriétés partiellement apprises à l'entraînement mais de n'importe quelle propriété qu'on souhaiterait imposer.

While this problem can occur both during training, where such global structure must be learned, and during inference (generation), where the gradually-outputted sequence must be guided toward that structure, we only consider the latter. Considering that the trained model has already partially learned the structure, we can fine-tune the sequence by using Constraint Programming with Belief Propagation (CPBP).

This guarantees that a generated molecule will respect the desired structure, which is critical if it is mandatory [3, 4].

This combined model is of more interest, however, when we wish to impose constraints that were not featured in the training dataset of the model. This allows the satisfaction of these new constraints while avoiding the retraining of the model, which can be costly with larger models.

This combination of both techniques could lead to valid, realistic and property-constrained molecules. However, there is a balance to maintain as we do not wish to stray too far from what was featured in the training dataset in order to respect the imposed constraints. This is particularly difficult for long-term structure, which requires balancing foresight over many yet-to-be generated tokens and the immediacy of next-token predictions from the sequence model.

1.1 Context-Free Grammar

A Context-Free Grammar is a set of rewrite rules used to generate strings. Formally, grammar $\mathcal{G} = (\mathcal{N}, \Sigma, \mathcal{R}, S)$ is defined, respectively, by a set of nonterminal symbols $\mathcal{N}$, a set of terminal symbols (its alphabet) Σ, 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. 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.

The classic example of a CFG is one where we match opening and closing parentheses. This becomes necessary later to ensure the validity of the generated molecules.

As an example of a CFG, take the grammar defined as follows:

$$\mathcal{N} = \{S, A, B, C\}$$
$$\Sigma = \{\langle, \rangle\}$$
$$\mathcal{R} = \{ \textcircled{1}\ S \to SS,\ \textcircled{2}\ S \to AC,\ \textcircled{3}\ S \to BC,\ \textcircled{4}\ B \to AS,\ \textcircled{5}\ A \to \langle,\ \textcircled{6}\ C \to \rangle\}$$
$$S = S$$

This context-free grammar recognizes correctly bracketed words such as "$\langle\langle\rangle\rangle$", obtained by the successive application of rules: $S \xrightarrow{3} BC \xrightarrow{4} ASC \xrightarrow{6} AS\rangle \xrightarrow{2} AAC\rangle \xrightarrow{5} A\langle C\rangle \xrightarrow{6} A\langle\rangle\rangle \xrightarrow{5} \langle\langle\rangle\rangle$. 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 1.1).

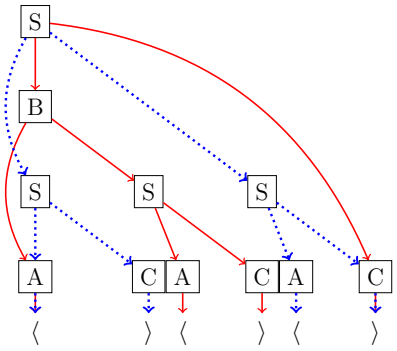

Figure 1.1 Grammar parse tree for the two words of length 4 recognized by the grammar shown in Section 1.1. The first word is in red, the second is in blue.

1.1.1 Chomsky Normal Form

David: Should I move this to when I talk about grammars and explain how the algo works there instead?

A grammar is said to be in Chomsky Normal Form if it follows a few additional rules on top of those of a CFG. No production may contain the null symbol ϵ and the right-hand side of any production must be either: a single terminal or two nonterminals. The following CFG could be converted to be in Chomsky Normal Form by applying four steps.

$$\mathcal{N} = \{S, X, Y, Z\}$$
$$\Sigma = \{a, b\}$$
$$\mathcal{R} = \{S \to XYZ,\ X \to aXb,\ X \to \epsilon,\ Y \to aa,\ Y \to bb,\ Y \to X,\ Z \to abba,\ Z \to XabY\}$$
$$S = S$$

0. Initial grammar

$$\begin{aligned} S &\to XYZ \\ X &\to aXb \mid \epsilon \\ Y &\to X \mid aa \mid bb \\ Z &\to XabY \mid abba \end{aligned}$$

1. Remove null productions

$$\begin{aligned} S &\to XYZ \mid XZ \mid YZ \mid Z \\ X &\to aXb \mid ab \\ Y &\to X \mid aa \mid bb \\ Z &\to XabY \mid Xab \mid abY \mid ab \mid abba \end{aligned}$$

2. Replace unit productions

$$\begin{aligned} S &\to XYZ \mid XZ \mid YZ \mid XabY \mid Xab \mid abY \mid ab \mid abba \\ X &\to aXb \mid ab \\ Y &\to aXb \mid ab \mid aa \mid bb \\ Z &\to XabY \mid Xab \mid abY \mid ab \mid abba \end{aligned}$$

3. Shorten the right-side to two tokens

$$
\begin{aligned}
S &\to XC \mid XZ \mid YZ \mid XD \mid XE \mid EY \mid ab \mid EF \\
X &\to aG \mid ab \\
Y &\to aG \mid ab \mid aa \mid bb \\
Z &\to XD \mid XE \mid EY \mid ab \mid EF \\
C &\to YZ \\
D &\to EY \\
E &\to ab \\
F &\to ba \\
G &\to Xb
\end{aligned}
$$

4. Create unit productions for terminal tokens

$$
\begin{aligned}
S &\to XC \mid XZ \mid YZ \mid XD \mid XE \mid EY \mid AB \mid EF \\
X &\to AG \mid AB \\
Y &\to AG \mid AB \mid AA \mid BB \\
Z &\to XD \mid XE \mid EY \mid AB \mid EF \\
A &\to a \\
B &\to b \\
C &\to YZ \\
D &\to EY \\
E &\to AB \\
F &\to BA \\
G &\to XB
\end{aligned}
$$

1.2 Chemistry

This section will detail different important notions in organic chemistry needed to understand the rest of this work.

1.2.1 Chemical Notation

Atoms are the building blocks of molecules and the bonds they can make are what allows the formation of complex structures. In organic chemistry, the atoms of interest are: Boron (B), Carbon (C), Nitrogen (N), Oxygen (O), Fluorine (F), Phosphorus (P), Sulphur (S), Chlorine (Cl), Bromine (Br) and Iodine (I). 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.

A valence shell is made up of multiple subshells of different energy levels: 1s, 2s, 2p, 3s, 3p, 3d, etc. Each of these subshells can hold a different number of electrons and the valence shell of a given atom is said to be complete when the outermost subshells are full. This often comes back to reaching the configuration of a noble gas, which are the rightmost atoms in the periodic table.

Having a complete valence shell is the stable configuration that most atoms tend towards. To achieve this, atoms will make ionic bonds, a bond where an electron is taken from another atom, or covalent bonds, a bond where an electron is shared by two atoms. In the case of organic molecules, we will usually only consider covalent bonds.

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. The earlier atoms used in organic chemistry have the following number of valence electrons: Boron has 3; Carbon has 4; Nitrogen and Sulfur have 5; Oxygen and Sulphur have 6; Fluorine, Chlorine, Bromine and Iodine have 7. They can do this by making the corresponding number of covalent bonds required to fill their valence shell (commonly represented as line segments between atoms; see e.g. Figure 1.2A).

David: Cite well, I think it is done, but verify

As seen in Figure 1.2B, to reduce the visual clutter of molecular graphs, Carbon and Hydrogen atoms are omitted. Carbon atoms are simply vertices with no letter indicating anything and Hydrogen atoms are implicitly present to complete the valence shell of any atoms that appear to be missing a bond.

1.2.2 Hydrogen Bonds

Hydrogen bonds are inter-molecular bonds caused by polarized molecules. They require a donor and an acceptor. Covalent bonds do not always equally share the shared electron, specifically, the more electronegative an atom is, the more it pulls on the shared electron.

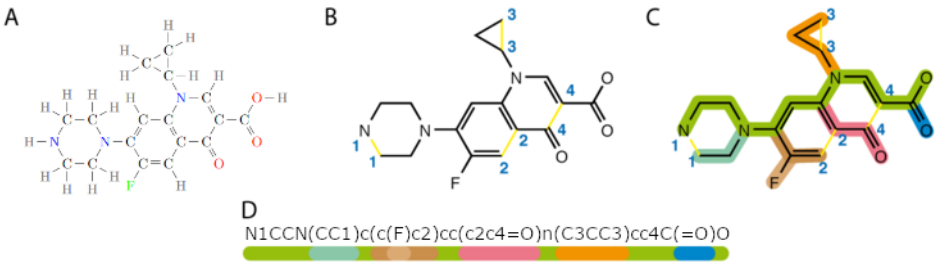

Figure 1.2 Deriving a SMILES representation for a molecule (reproduced in part from [5]). The structural formula of the molecule (A), its skeletal formula stripped of all hydrogen atoms and with broken cycles (B), the selected main path (shown in green) and branches (C), and the corresponding SMILES notation (D).

The electronegative atoms that interest us in the context of organic molecules are: Fluorine (F), Sulphur (S), O (Oxygen) and N (Nitrogen).

The **donor** is an electronegative atom linked to a Hydrogen atom. By pulling on the shared electron more than the Hydrogen atom does, the electronegative gains a partial negative charge. Inversely, the Hydrogen atom gains a partial positive charge.

The **acceptor** is an electronegative atom with a free electron pair on its valence shell. This electron pair, can then attract the partially positively charged Hydrogen from the donor.

This attraction, between two different polarized molecules, is what we call a Hydrogen bond. The most famous example of this is in water and is the reason for many of water's interesting properties (cohesion, high boiling point, high heat capacity, surface tension, expands when frozen, etc). In this case, the Oxygen atom is both the donor and the acceptor. The Oxygen atom, acting as the acceptor, is negatively charged and can attract the positively charged Hydrogen atoms from other water molecules. The same atom will donate its positively charged Hydrogen atoms to other Oxygen atoms.

1.2.3 Molecule Encodings

Molecules can be encoded in many different ways. Two common methods are representing molecules as graphs or as one-dimensional strings. We will be using a one-dimensional encoding in our work to simplify the representation and potentially allow a combined model using NLP models. We will present different one-dimensional encodings used in the molecule discovery field.

David: J'ai add canon SMILES pour montrer d'ou vient deepSMILES, mais maybe not needed? Je trouvais ca plus simple pour comparer

Encoding	Representation
InChI	InChI=1S/C17H18FN3O3/c18-13-7-11-14(8-15(13)20-5-3-19-4-6-20)21(10-1-2-10)9-12(16(11)22)17(23)24/h7-10,19H,1-6H2,(H,23,24)
SMILES	N1CCN(CC1)c(c(F)c2)cc(c2c4=O)n(C3CC3)cc4C(=O)O
canonical SMILES	O=C(O)c1cn(C2CC2)c2cc(N3CCNCC3)c(F)cc2c1=O
DeepSMILES	O=CO)ccnCCC3)))cccNCCNCC6))))))cF)cc6c%10=O
SELFIES	[N][C][C][N][Branch1][Branch1][C][C][Ring1][=Branch1][C][Branch1][=Branch1][C][Branch1][C][F][=C][=C][C][=Branch1][=Branch1][=C][Ring1][Ring2][C][=O][N][Branch1][=Branch1][C][C][C][Ring1][Ring1][C][=C][Ring1][Branch2][C][=Branch1][C][=O][O]

Table 1.1 Different encodings of the molecule shown in Figure 1.2. DeepSMILES could not originally encode the SMILES string, we converted the molecule to canon SMILES for it to encode it correctly.

InChI

International Chemical Identifier (InChI) is a notation standard introduced by the International Union of Pure and Applied Chemistry (IUPAC) [6]. It provides a unique one-dimensional representation of the molecule. The encoding contains information on both the structure and certain properties of the molecule. This representation was not adopted by the automated molecule discovery research community because of its low readability by both humans and machines. Instead, it has become commonly used in indexing and searching tasks (*e.g.* databases).

SMILES

SMILES is a one-dimensional string representation for molecular encoding [7]. This encoding is much simpler than InChI, only maintaining the structural information required to reconstruct the molecule. However, any lost information can be recovered using different techniques. Admittedly, the process can be difficult and time-intensive to guarantee accurate results.

David: Get sources showing which ppl do this

If we picture a molecule as a graph where every vertex is an atom, a SMILES string would be the order in which we explore a tree-representation of the graph using a Depth-First Search (DFS). Since SMILES is a DFS over a tree, any cyles that were in the original graph would be lost. Thankfully, SMILES considers this by designating a specific token to represent broken cycle bonds. This prevents losing the cycles when we convert the graph into a tree. These tokens are represented by number tokens as seen in Figure 1.2B and, later, in D. Similarly,

when we reach a branching path in the tree, one side is chosen as the main branch, which is shown in green in Figure 1.2C, while the other side is written between parentheses to indicate that it is a branch.

A very common concept in organic chemistry is aromatic rings. These are usually 5 or 6 atoms in a ring, the ring bonds alternate between single and double bonds. Due to their frequency, the SMILES language has started using a shorthand for it by writing atoms in aromatic rings using lowercase letters. This allows us to omit the alternating single and double bonds during writing and makes aromatic rings much more visible when reading a molecule. Kekulized SMILES keeps these bonds explicit, but non-kekulized SMILES is preferred.

This encoding has gained a lot of popularity in the automated molecule discovery research community due to its age and ease of readability. Since its introduction, it has become the most popular string representation in automated discovery. However it comes with certain issues that we must address. Unlike InChI, SMILES does not offer a unique encoding for each molecule. It is therefore possible to generate two different strings that describe the same molecule. Another prominent issue is linked to SMILES' special tokens. By requiring an opening token, as is needed to describe cycles, branches and isotopes (which we did not describe), any string that does not have corresponding open and close tokens is syntactically invalid.

David: À déplacer là où tu parleras des limites de la ML

This can be problematic in token-by-token generation if these rules are not hard constraints, which is the case in ML techniques since they lack the ability to impose long-term structure.

SMILES also has no check on the valence shells of the atoms within it. For example a Carbon atom, which wants to make 4 bonds to complete its valence shell, could be placed in such a way that it has 6 bonds.

There are some ways to generate canonical SMILES strings (*i.e.* unique for a given molecule), however no consensus has been reached on which method to use.

This is the encoding that we use during our work, mainly due to its popularity within the automated molecule discovery community, which allowed us to find documentation and tools that helped during the work. We will present two other molecule encodings that were introduced to resolve issues within SMILES, but they were not used due to their relatively new appearance and, consequently, to their smaller research community.

DeepSMILES

DeepSMILES was introduced to answer some of SMILES' shortcomings [8]. It changes how branches and cycles are represented so that only one token is required. Instead of representing cycles using numbers as tags, they instead use numbers to indicate the size of the cycle and place the number at the end of the cycle. Similarly, branches no longer require opening branch tokens, instead they place as many branch closing tokens as there are atoms in the described branch.

Unfortunately, DeepSMILES is not perfect and sometimes fails to encode a molecule correctly. The example molecule used in Figure 1.2 cannot be directly converted into DeepSMILES from its SMILES format. This is a big issue, since the encoding could fail based on which bond in a cycle we choose to break to convert the graph into a tree. However, this can be avoided by first converting the molecule to canonical SMILES. The molecule still has an encoding in DeepSMILES, as shown in Table 1.1.

SELFIES

Similarly to DeepSMILES, SELF-referencIng Embedded Strings (SELFIES) [9] was introduced specifically for ML applications, its language having been designed to minimize syntax invalidity and simplify the structure for ML models.

To resolve some of SMILES' syntax problems (*i.e.* branch and cycle invalidity), it associates each token to a numeric value. It then overloads the tokens following cycle or branch tokens, replacing them by their numeric value. In the case of branches, they place the token at the start of the branch and the overloaded value tells us how many of the future tokens are a part of this branch. For cycles, the token is placed at the end of the cycle and the overloaded value indicates how many atoms back we have to go to find the start of the cycle.

Another important difference is that all tokens are described between square brackets to remove some ambiguity. In SMILES, the square brackets are omitted for common atoms to improve readability.

1.2.4 Lipinski's Rule of Five

Lipinski's Rule of Five is a set of rules describing properties that orally administered drugs tend to respect. While there are only four rules, each rule contains a value that is a multiple of five, which is where the name comes from.

The rules are as follows:

- The molecular weight must not exceed 500 Daltons.
- There must not be more than 10 Hydrogen-bond acceptors.
- There must not be more than 5 Hydrogen-bond donors.
- The logP must not exceed 5.

The molecular weight is the simplest property to understand. By limiting the weight of the molecule, we tend to avoid molecules that are too large. It is important to note that Daltons are on a one-to-one scale with g/mol, which is the more commonly used unit.

Hydrogen-bond acceptors as seen in section 1.2.2, are electronegative atoms (*e.g.* F, S, N, O) with a free electron pair on their valence shell to act as an acceptor for the Hydrogen-bond.

Hydrogen-bond donors , as seen in section 1.2.2, are electronegative atoms linked to a Hydrogen atom. This Hydrogen atom will allow the Hydrogen-bond with an acceptor.

The logP is an evaluation of how lipophilic or hydrophobic a molecule is, *i.e.* how easily the molecule dissolves in fats as opposed to water. This is relevant when trying to control how a drug is absorbed in the human body.

1.3 Constraint Programming

Constraint Programming is a complete, heuristic guided search method which excels at ensuring the respect of constraints while generating a solution. It is complete in that if a solution exists in a given search space, a CP model is guaranteed to find it. By using heuristics as well as constraint propagation (more on that later), it can be much faster than a simple brute force of all possible solutions.

We will first define how a simple CP model functions. We will cover the initial problem declaration, the constraint declaration to describe the problem and finally the solving process and its intricacies (constraint propagation, branching decisions, backtracking). Once that is covered, we can expand on this topic by introducing CPBP

David: cite BP from Gilles' paper

which is an improvement over standard constraint propagation and leads to more informed decisions. We use CPBP in our work since it tends to yield better results and allows for the combination with a ML model as we will describe later.

1.3.1 Constraint Satisfaction Problem

A Constraint Satisfaction Problem (CSP) is defined in three parts:

- The variables making up the problem, defined as the finite set $\mathcal{X}$
- The domains of these variables, defined as a finite set of values D. Each variable can have its own domain
- The constraints, each of which is applied to a subset of the variables, defined as a set of constraints C.

There are a finite number of **variables** defined in the set $\mathcal{X}$. Each of these variables has its own **domain** as is defined in the set D, which contains the possible values that a variable may take on. Finally, we define a finite number of **constraints**, each of which is applied on a subset of the variables. Each variable must then be assigned a value from its domain such that it respects all the applied constraints. If such an assignment is possible for all the variables, that is a solution to the problem.

If we take the Sudoku problem as an example, a classic and very commonly seen problem, we can define it as a CSP as follows. Our **variables** will be each tile in the 9x9 grid. While this gives us the layout of our problem, we must define the possible values for each variable to be able to solve this problem. All the variables can take on the same values and so we can define the **domain** as being the integer values between 1 and 9 inclusively.

We could represent this using a 2-dimensional array of variables like so:

$$tile[i][j] \in \{1, 2, \ldots, 9\} \mid i, j \in \{1, 2, \ldots, 9\} \tag{1.1}$$

All that is missing are the constraints, which are the source of the complexity of the problem.

The **constraints** in a Sudoku are fairly straightforward, lines, columns and all 3x3 sub-grids within the total grid may not contain any repeat values. In the CP community, this type of constraint is very common and is called an `alldifferent` constraint. The Sudoku problem would therefore have the following constraints:

$$\texttt{alldifferent}(tile[1][j], tile[2][j], \ldots, tile[9][j]) \ \forall \ j \mid j \in \{1, 2, \ldots, 9\}$$
$$\texttt{alldifferent}(tile[i][1], tile[i][2], \ldots, tile[i][9]) \ \forall \ i \mid i \in \{1, 2, \ldots, 9\}$$
$$\begin{gathered}\texttt{alldifferent}(\\ tile[3u+1][3v+1], tile[3u+1][3v+2], tile[3u+1][3v+3], \\ tile[3u+2][3v+1], tile[3u+2][3v+2], tile[3u+2][3v+3], \\ tile[3u+3][3v+1], tile[3u+3][3v+2], tile[3u+3][3v+3], \\ ) \ \forall \ u, v \mid u, v \in \{0, 1, 2\}\end{gathered}$$

Overall, we would need 81 variables to define this CSP as well as 27 constraints. Each of our variables could take on any of the 9 possible values in their domain.

1.3.2 Domain Filtering

As mentioned in the previous section, each constraint is applied to a subset of the variables in the problem definition. When a constraint is declared, a filtering algorithm that is specific to that constraint will eliminate values that are inconsistent.

The simple example below illustrates how a constraint can filter a variable's domain after being declared.

$$\begin{gathered}x \in \{2, 3, 4\} \\ y \in \{1, 2, 3\} \\ x \leq y \\ x \in \{2, 3, \cancel{4}\} \\ y \in \{\cancel{1}, 2, 3\}\end{gathered}$$

Both variables initially contained a value that would always breach the constraint if chosen. A value such as that one is said to have no support, *i.e.* there are no solutions to the current constraint that contain this value. A visual representation of this can be seen in Figure **??**, where a constraint is applied to two different variables and both have their domain filtered. While we do not know all the solutions to a problem in all cases, we can use logical processes to determine values that would guarantee a breach of the constraint.

1.3.3 Constraint Propagation

Now that we have declared our constraints, the solver begins propagating the consequences of these constraints. Each constraint in the queue communicates to the variables it affects which values in the domain have to be filtered out. Once a variable's domain has been changed, it notifies the constraints that are affected by the change and those constraints are then added to the queue again.

The solver continues propagating the consequences of the constraints and updating domains until it reaches one of three situations:

1. The queue is empty, but there remain unassigned variables.
2. All variables have been assigned a value, this is a solution to the problem.
3. One of the variables' domain has been completely filtered, there is no solution in the current state of the problem.

In the first case, there is nothing else to deduce with the information currently available and the solver has to make a branching decision from the current state. Any time we reach one of the three cases above, we can consider that state as being a node in the search tree. The solver makes a branching decision from the current node and propagates the consequences of this decision until it reaches another node to handle.

In the second case, the solver has found a solution and can add it it the solution set. Once the solution has been found, we backtrack to the previous node in the search tree and search along the other branches.

Finally, if we reach an unsatisfiable state, the solver backtracks to the previous node and continues its search from there.

Since we have a finite number of values, we know that this process will eventually end and we will either find a value that respects the constraint, or, find that the constraint cannot be satisfied.

1.3.4 Solving

David: Cette section me semble inutile rendu la. Pourquoi pas tout mettre dans constraint propagation?

To continue with the example of a Sudoku, a classic way humans continue solving, once they reach a dead end in their reasoning, is by assigning a value to a tile and seeing if they

reach a contradictory state. If they do, then they know their choice was wrong and they can eliminate that possibility.

1.3.5 Marginals-Augmented Constraint Programming

David: Je mettrais pareil cette section avant la BP, non?
Ça me semble étrange de parler de solution counting dans la section BP, sans avoir montrer les marginales (qui sont du solution counting).

In an ideal world, if we knew every possible solution to a problem, we could use the values within the solution to inform our search and avoid bad branching decisions. This is especially useful when we consider bigger problems that might have a huge combinatorial space to explore.

Marginals-augmented constraint programming is the idea of guiding our branching decisions by counting the number of solutions to a constraint that contain a given value for a given variable as seen in Figure **??**. The difficulty of this task is that it requires an efficient algorithm which can predict the number of solutions without finding and enumerating all possible solutions to the constraint.

David: Redo the figures yourself

One use of these marginals is to change standard constraint propagation to contain more information. Belief Propagation does this by modifying the message that constraints send variables. Instead of sending a message containing a binary representation of which values in the domain have a support, the messages are modified to communicate the probability of a value being contained within a solution as seen in Figure 1.3. This allows the solver to avoid branching on values that have a very small chance of being valid.

When multiple constraints interact on one variable, they each simultaneously communicate to the variable what they estimate the probability distribution to be. The variable then merges these probabilities into the final values as seen in Figure 1.4.

1.4 Neural Networks for Natural Language Processing

This section will give simplified descriptions of different necessary notions for this work.

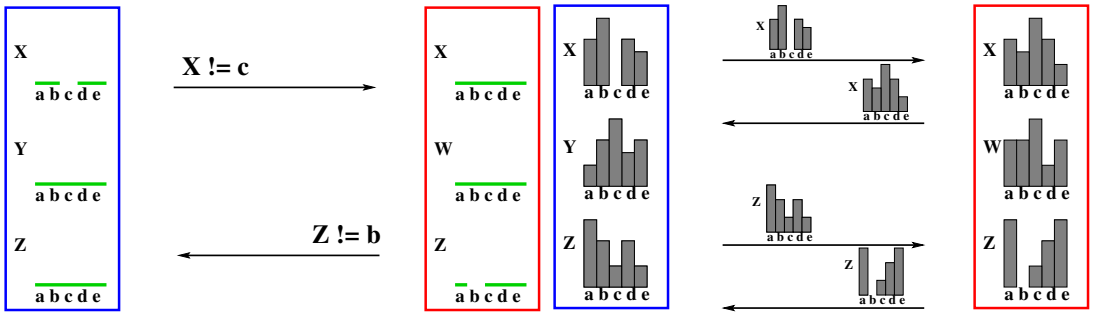

Figure 1.3 Belief Propagation replaces standard constraint messages, which consist of a binary message indicating which values are supported in the domain, with a probabilistic distribution over the domain. As we can see, instead of communicating that the value c for variable X lacks a support, the blue constraint communicates that $X = c$ has a 0% chance of being in a valid solution. This ensures that we can still communicate what values must be filtered out, but we also gain information on the other values in the domain.

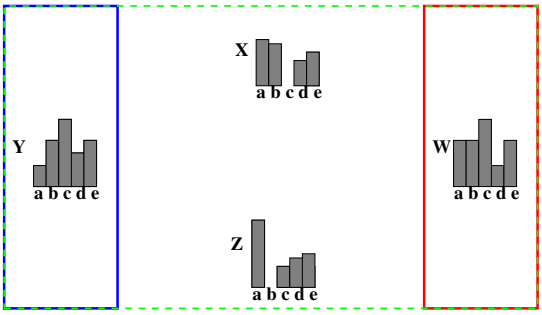

Figure 1.4 The variables which are affected by multiple constraints (X and Z) merge the communicated probabilities that were communicated by the different constraints. X filters out the value c and Z filters out b, as was communicated by the constraints seen in Figure 1.3.

1.4.1 Neural Network

Neural Networks are a Machine Learning architecture that get their name from their resemblance to a brain. Similarly to a brain, a Neural Network (NN) has neurons that communicate with each other to learn how to solve the task at hand. The simplest network we can make is made up of one input layer and one output layer. To improve the learning capabilities of this model, we can add hidden layers, which are neuron layers between the input and output ones. A model which has more than 2 hidden layers is called a Deep Neural Network.

The input layer contains as many nodes as the problem has inputs, each node representing one value. Similarly, the output layer contains as many nodes as the problem has. A model can contain any number of hidden layers, each of which is made up of any number of nodes. Each node in a hidden layer takes its inputs from every node in the previous layer and,

inversely, sends its output to every node in the next layer.

In a standard model, the node sums up the product of all the inputs and their associated weight before applying an activation function to the sum. This result is the node's output and will be passed on to the next layer where it will be used as the input in a similar operation. For the model to learn complex relations, it is critical that the activation function used is non-linear. If the activation function were linear, the entire model would collapse back into a simple linear equation.

To find the right weights, the model must first be trained on a part of the total dataset. During training, the model computes the error between the expected result and the predicted one and then backpropagates this error from layer to layer. Each layer then recalculates the weights of its inputs based on the obtained error before sending a modified message to the previous layer.

From there, the trained model can be given any problem input and will calculate the predicted output based on its internal weights.

1.4.2 Transformers

Transformers [10] are a ML architecture based on encoders and decoders. The model first passes the input through an encoder, that encoded sequence is then used by the decoder to generate an output one token at a time.

The encoder is made up of multiple identical layers, each composed of two sub-layers: a multi-head attention layer and a feed-forward network. The multi-head attention layer is an improvement over standard attention models and allows the model to learn more complex relations. The input embeddings received by the multi-head attention sub-layer maintain more context during training and generation by encoding both the input sequence as well as positional information.

The decoder is also made up of multiple identical layers, each composed of three sub-layers: a masked multi-head attention layer, a standard multi-head attention layer and a feed-forward network. The masked multi-headed attention layer's output is then fed into the next multi-headed attention layer with the encoded input from the encoder. This is finally passed through a feed-forward network. The input received by the masked multi-headed attention is an embedding which encodes both the current output sequence as well as positional information on the tokens.

Once this final output is calculated, we apply a softmax on it to get the probabilities for the next token.

1.4.3 Large Language Model

LLMs were introduced shortly after the proposal of transformers in 2017. Following transformers, Bidirectional Encoder Representations from Transformers (BERT) [11] was introduced as an encoder-only architecture and can be considered the start of LLMs. However, this type of architecture came into the limelight with the Generative Pre-trained Transformer (GPT) models from OpenAI.

The specific GPT model that interests us is the GPT-2 model [12], it is what we use in our architecture. Similarly to what was introduced for GPT-1 [13], the model is a large decoder layer, as seen in the transformers. An important difference is that the second multi-head attention sub-layer is removed from each of the identical layers in the decoder.

These models are usually trained to complete many different tasks, however by training one on a SMILES dataset, we can get a GPT-2 model to generate molecules based on what it has seen.

1.5 Problem Statement

As mentioned previously, drug discovery is both time-consuming and costly and automated drug discovery has been an important field of research to reduce these costs. While ML methods have been gaining a lot of popularity in the field, those techniques suffer from a lack of long-term structure. To address this, CP is a natural answer since it provides the lacking long-term structure.

David: Cite works using CP could help

However, while CP is used in the domain, there isn't much work

David: I found none, but to be investigated further

relating to generating molecule candidates using CP.

We believe that a CP model would be beneficial and would reduce the number of invalid molecules generated.

More importantly however, a CP model that generates valid molecules could then be used to target property-specific molecules using constraints to eliminate undesirable options.

The issue of using CP for this problem is the size of the search space to explore. By using Belief Propagation, we believe that the search will be better guided towards a solution and require less backtracking. However, it remains to be seen if the added cost for the Belief Propagation increases the overall time to solve.

Finally, we believe that by combining a trained token-by-token generating ML model with our CPBP model, we might get molecules similar to what is being used today (molecules in datasets) while still maintaining the long-term structure of the CP model.

1.6 Research Questions

During our research we will answer the following questions:

1. Can we use CP to model molecules in a one-dimensional encoding?
2. Can we use CP to model desirable molecular properties in SMILES molecules?
3. Does combining a CP model with a NLP model improve the realism of generated molecules?

David: Le dernier point me fait encore douter. C'est évident que oui puisqu'on utilise le même modèle durant la génération ET l'évaluation. Donc c'est sûr que ce qu'on génère va être plus réaliste, aka ressembler plus à ce qu'il y a dans le dataset

1.7 Thesis Outline

The rest of this thesis is organized in the following chapters:

- Chapter 2 goes over the necessary concepts to understand the rest of the paper.
- Chapter 3 provides a general overview of the different techniques currently in use.
- Chapter 4 presents our base model as well as the methods used to model valid molecules as per our first research question.
- Chapter 5 expands on the previous section and introduces ways to model molecular properties using CP. This section addresses our second research question.
- Chapter 6 details how we combine our CP model with a NLP model.
- Chapter 7 goes over the paper's contributions, its limitations and potential ways to improve this in future work.

CHAPTER 2 BACKGROUND

2.1 Chemistry

2.1.1 Organic Chemistry Notation

2.1.2 Lipinski's Rule of Five

2.2 Constraint Programming

2.3 Natural Language Processing

CHAPTER 3 LITERATURE REVIEW

3.1 NLP applied to drug discovery

3.2 CP applied to drug discovery

3.2.1 Combining CP with ML

CHAPTER 4 MODELING VALID MOLECULES USING CP

In this chapter, we will put forward a way to model that can represent valid molecules using CP. As mentioned previously, we choose to use SMILES to encode our molecules. This is a simple and easy-to-read one-dimensional molecule representation which can easily be modelled by CP.

4.1 SMILES Representation

David: Should I justify the size of our molecules more? C'était relativement arbitraire comme décision, mais on l'a fait pour garder de la complexité et limiter le temps de recherche dans l'arbre. On avait fait des tests, mais je n'ai plus les valeurs (i don't think so at least)

We chose to limit the size of our molecules to 40 tokens. We made this decision to ensure the problem was difficult enough without making it too long to solve and get results.

Each variable's initial domain contains the entire SMILES alphabet. This allows any combination of SMILES tokens including invalid combinations. To ensure validity, we use three constraints as described in the following subsections.

4.1.1 Grammar

In our work, we use a variation of Kraev's grammar [14] to ensure that atom valences are respected in the generated molecules. In SMILES notation, an ion is written differently than an

Respecting atom valences

increases the potential stability of the generated molecule. However, we cannot guarantee the stability of the molecule even if we do respect valence rules since it is much more complex.

The original work uses masks in addition to this grammar to completely avoid invalid outputs. The first mask handles numerical assignment for cycles, guaranteeing that cycles are numbered correctly. The second mask avoids making cycles that are too small (*i.e.* cycles of 2 atoms) and cycles that are too long. They limit their cycle length to 8 based on what they observe in their database [14].

We address both of these issues by modifying the base grammar and adding new constraints

as will be discussed later.

Chomsky Normal Form

The solver we use, miniCPBP

David: cite miniCPBP

, has an implementation of the grammar constraint that requires a grammar in Chomsky Normal Form.

David: Potentially insert what is in the intro here

We automated the process of converting the CFG into the right form. This allows us to keep working on the more readable CFG format.

The original grammar from Kraev contained 34 terminals, 36 nonterminals and 138 productions. After conversion, the number of nonterminals and productions, respectively, increase to 169 and 411.

Padding

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, allowing for a more versatile model. We chose "_" as our padding token.

An easy way to make this change is to create a new starting token that can be developed into the old start token and any number of padding tokens (including none). This change was not influential on the performance of the algorithm and allows for more options during generation.

Hydrogen tokens

David: Section potentially useless, mais je voulais être explicit avec tous les changements qu'on a fait.

Some Hydrogen tokens can be included in the molecule. These can be followed by a number to indicate the number of Hydrogen atoms present. We change these tokens to directly include the number. Instead of needing two tokens ("H" and "3") we now use one token ("H3") made up of two characters.

This avoids confusing Hydrogen count tokens for cycle tokens and improves our model's understanding of what it is generating.

Cycle-length limit

The final required modification we make to our grammar is to limit the cycle length. This guarantees that the cycle length remains in the desired range (between 3 and 8 inclusively) and that an opened cycle is necessarily closed. Unclosed cycles are syntactically invalid, but long cycles seem to be infrequent because of lower stability. MOSES [15], a data set of about two million molecules, never exceeds length-6 cycles while another, Zinc_250k [16], features some length-8 cycles.

We achieve this by limiting the number of tokens that a cycle production can be developed into. This information must be encoded in nonterminals where a larger cycle nonterminal can be rewritten as an atom and a smaller cycle nonterminal.

This change alone guarantees that any nonterminal "num" will have another nonterminal "num" within an acceptable distance. However, this does not guarantee that the nonterminal "num" will be developped into the same cycle number. Take the unfinished chain "C*num*CCCCC*num*NC*num*CCCCC*num*" as an example. While we would expect the finished chain to be "C1CCCCC1NC2CCCCC2", nothing is stopping the grammar from developing it into "C1CCCCC2NC2CCCCC1" instead.

This was a problem we ran into fairly quickly after applying the cycle size limit changes to the grammar, resulting in one very long cycle and one small one instead of two appropriate cycles. The solution was to integrate into the left-hand side of the production information about which cycle is being developed.

As Kraev mentions in the original paper [14], this change will make the grammar grow very quickly in size based on the maximum number of cycles allowed (not to be confused with the maximum cycle-length). We chose to limit it to 6 cycles for two reasons. First of all, only eight molecules in the ZINC250K dataset [16] have more than 6 cycles. The second reason is that the smallest possible cycle, cycles of length 3, need 5 tokens in our model. Having 6 cycles of that length would take 30 out of the 40 tokens in our molecule and, in our tests, we did not notice any molecule with more than 4 cycles.

David: Is this appropriate as a justification? La deuxième raison

After all these changes, we ensured that cycles had an appropriate length. However this meant that our CFG now had 32 terminals, 194 nonterminals and 538 productions. After conversion to the Chomsky Normal Form, the number of nonterminals and productions,

respectively, explode to 640 and 1996. It is a large grammar.

David: analyze grammar algorithm to determine if it is cubic according to variables or productions

CHAPTER 5 MODELING MOLECULAR PROPERTIES USING CP

David: Determine if we merge it or not

Texte / Text.

CHAPTER 6 COMBINING CP WITH NLP TO IMPROVE GENERATION

CHAPTER 7 CONCLUSION

Texte / Text.

7.1 Synthèse des travaux / Summary of Works

Texte / Text.

7.2 Limitations de la solution proposée / Limitations

7.3 Améliorations futures / Future Research

Texte / Text.

REFERENCES

[1] P. et al., "Estimation of the size of drug-like chemical space based on gdb-17 data," *Journal of Computer-Aided Molecular Design*, 2013.

[2] Y. Du, T. Fu, J. Sun, and S. Liu, "Molgensurvey: A systematic survey in machine learning models for molecule design," 2022.

[3] D. Deutsch, S. Upadhyay, and D. Roth, "A general-purpose algorithm for constrained sequential inference," in *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, 2019, pp. 482–492.

[4] J. Y. Lee, S. V. Mehta, M. Wick, J.-B. Tristan, and J. Carbonell, "Gradient-based inference for networks with output constraints," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 4147–4154.

[5] W. Commons, "Smiles.png," online, accessed July 12, 2023. [Online]. Available: https://commons.wikimedia.org/wiki/File:SMILES.png

[6] S. R. Heller, A. McNaught, I. Pletnev, S. Stein, and D. Tchekhovskoi, "InChI, the IUPAC International Chemical Identifier," *Journal of Cheminformatics*, vol. 7, no. 1, p. 23, Dec. 2015. [Online]. Available: https://jcheminf.biomedcentral.com/articles/10.1186/s13321-015-0068-4

[7] D. Weininger, "SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules," *Journal of Chemical Information and Computer Sciences*, vol. 28, no. 1, pp. 31–36, Feb. 1988. [Online]. Available: https://pubs.acs.org/doi/abs/10.1021/ci00057a005

[8] N. O'Boyle and A. Dalke, "DeepSMILES: An Adaptation of SMILES for Use in Machine-Learning of Chemical Structures," Sep. 2018. [Online]. Available: https://chemrxiv.org/engage/chemrxiv/article-details/60c73ed6567dfe7e5fec388d

[9] M. Krenn, Q. Ai, S. Barthel, N. Carson, A. Frei, N. C. Frey, P. Friederich, T. Gaudin, A. A. Gayle, K. M. Jablonka, R. F. Lameiro, D. Lemm, A. Lo, S. M. Moosavi, J. M. Nápoles-Duarte, A. Nigam, R. Pollice, K. Rajan, U. Schatzschneider, P. Schwaller, M. Skreta, B. Smit, F. Strieth-Kalthoff, C. Sun, G. Tom, G. Falk Von Rudorff, A. Wang, A. D. White, A. Young, R. Yu, and A. Aspuru-Guzik, "SELFIES and the future

of molecular string representations," *Patterns*, vol. 3, no. 10, p. 100588, Oct. 2022. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2666389922002069

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019. [Online]. Available: https://arxiv.org/abs/1810.04805

[12] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.

[13] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

[14] E. Kraev, "Grammars and reinforcement learning for molecule optimization," 2018.

[15] P. et al., "Molecular sets (moses): A benchmarking platform for molecular generation models," *Frontiers in Pharmacology*, vol. 11, 2020, iSSN: 1663-9812. [Online]. Available: https://doi.org/10.3389/fphar.2020.565644

[16] T. Akhmetshin, A. I. Lin, D. Mazitov, E. Ziaikin, T. Madzhidov, and A. Varnek, "ZINC 250K data sets," 12 2021. [Online]. Available: https://figshare.com/articles/dataset/ZINC_250K_data_sets/17122427