Rapport projet RAMI :
Génération de structures moléculaire

Réalisé par par Manon Girard, Romain Durand et Paul Peyssard
Encadré par Nicolas Prcovic

29 janvier 2024

Master 2 Intelligence Artificielle & Apprentissage Automatique

Aix-Marseille Université
Centrale Marseille

Contents

1 Introduction

Ce rapport détaille le projet mené par Manon Girard, Romain Durand et Paul Peyssard, étudiants en Master 2 Intelligence Artificielle et Apprentissage Automatique à Aix-Marseille Université. Ce travail a été réalisé sous la supervision de Nicolas Prcovic, enseignant chercheur au LIS (Laboratoire d'Informatique et des Système) dans l'équipe COALA (COntraintes, ALgorithmes et Applications) [12].

Dans le cadre de sa thèse [13], Adrien Varet, anciennement doctorant au sein de l'équipe COALA, s'est intéressé à la génération des structures de benzénoïdes possédant des propriétés spécifiques. Il a alors developpé le logiciel BenzAI [1] dont le but est de proposer aux chimistes théoriciens une interface intuitive permettant l'application de ses recherches. Ce programme permet la génération automatique de benzénoïdes à partir d'informations données, comme le nombre d'hexagones, le nombre d'atomes de carbone et/ou d'hydrogènes, etc. Dans la continuité des travaux d'Adrien Varet, nous avons pour objectif dans ce projet de créer une application plus générale permettant de générer automatiquement des structures moléculaires à partir d'une liste d'atomes de différents types. Nous nous limitons à la partie programmation par contraintes, sans implémenter d'interface graphique. Pour avoir des rendus visuels, nous utilisons l'application JMol [9] et dans une moindre mesure la bibliothèque GraphViz [7].

L'ambition principale du projet est de fournir aux chimistes un outil leur permettant, à partir de n'importe quelle liste d'atomes, d'accéder à un éventail complet de structures moléculaires potentielles. Pour ce faire, nous avons utilisé une approche par programmation par contraintes [15]. La résolution d'un problème par la programmation par contraintes se fait en deux temps.

Un premier temps est consacré à la *modélisation* du problème à travers un certain formalisme. Dans ce projet, nous avons utilisé le formalisme CSP (*Problème de Satisfaction de Contraintes*) [11]. Ce formalisme consiste à représenter le problème par un triplet (X, D, C) où X est un ensemble de variables, D est l'ensemble des domaines de valeurs de chacune des variables, et C est l'ensemble des contraintes du problème. Ce formalisme nous permet de transformer le problème donné en entrée, en une *instance*, dans notre cas une instance CSP. Cette instance est une traduction du problème dans un format compréhensible par un algorithme de résolution. Dans notre situation, nous possédons une liste d'atome, nous souhaitons faire comprendre à la machine que nous voulons en obtenir une structure moléculaire. Ainsi, à partir de nos données, nous allons créer une instance comprenant toutes les conditions, qu'on appelle *contraintes*, pour former une structure moléculaire.

Après avoir modélisé notre problème, nous pouvons passer à l'étape de résolution. Dans ce cadre là, l'objectif est de trouver une affectations pour chaque variables de l'instance, de telle manière que chacune des contraintes du problème soit respectée.

Que ce soit pour la partie modélisation, ou celle de résolution, nous allons utiliser la bibliothèque `chocosolver` qui utilise le solveur Choco pour la partie résolution. Choco est un solveur de contraintes *open-source* écrit en Java, destiné notamment à la résolution de problèmes de satisfaction de contraintes (CSP). Il permet aux utilisateurs de modéliser leur problème sous forme de contraintes sur des variables et de rechercher ensuite des solutions satisfaisantes ou optimales selon ces contraintes.Ce solveur a l'avantage de prendre en charge plusieurs types de variables, notamment des variables de graphe et des variables réelles, ce qui permet une grande flexibilité dans la modélisation des problèmes, contrairement à d'autres solveurs. De plus, il fournit un large éventail de contraintes prédéfinies, telles que les contraintes arithmétiques, et de logiques.

Dans la suite de ce rapport, nous suivrons le processus du projet étape par étape. Nous débuterons par une analyse approfondie du problème pour examiner comment il peut être modélisé. Nous présenterons notre première approche de modélisation, puis nous explorerons les améliorations apportées à ce modèle initial. Par la suite, nous aborderons certains aspects techniques, notamment les formats de données en entrée et en sortie de nos programmes et la gestion des expériences. Enfin, nous présenterons certains résultats obtenus et conclurons en résumant nos accomplissements, ainsi que les défis rencontrés, et en envisageant les perspectives futures de notre travail.

2 Une première modélisation du problème

Pour commencer, nous devons définir ce que nous appelons une *structure moléculaire*. Vous retrouverez un exemple sur la figure 1 ci-dessous. Cette figure correspond à un graphe où les sommets sont des atomes, comme l'oxygène etc., et les arêtes correspondent à des liaisons entre ces atomes. En chimie moléculaire, il existe différents types de liaisons, comme les liaisons simple, double ou encore triple. Sur la figure 1, les liaisons simples sont les traits simples, et les liaisons doubles sont représentées par un trait doublé. Pour la première modélisation, nous nous sommes limités aux liaisons simples.

De plus, ce graphe est un *graphe étiqueté*. En effet, à chaque sommet on va associer un *type*, qui sera tout simplement le nom de l'atome auquel il correspond, par exemple, atome d'hydrogène.

Il est important de noter que ce graphe possède une propriété importante : il est *connexe*. Cela signifie simplement qu'il existe une chaine reliant chaque atome entre eux. Ainsi, nous voyons apparaitre une première contrainte. De plus, chaque type d'atome possède un degré de liaison, appelé *valence* qui représente le nombre de liaisons qu'il peut former. Par exemple, les atomes d'oxygène ont une valence de 2, ce qui signifie qu'ils doivent, avoir soit deux liaisons simples, soit une liaison double. Il nous faudra donc faire attention à ce que chaque atome possède autant de liaisons que sa valence l'impose.

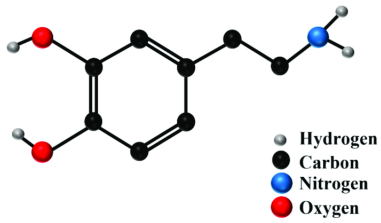


Figure 1: Structure moléculaire de la dopamine

L'objectif de cette première modélisation est donc de construire ce graphe à partir d'une liste d'atomes, en respectant les contraintes imposées. Cette liste représente la *formule chimique* de la molécule en question. Une formule chimique est une expression qui indique quels atomes sont présents dans la molécule, et en quelles quantités. Dans l'exemple de la figure 1, celle-ci est $C_8H_{11}NO_2$, ce qui signifie que que cette molécule est composée de 8 carbones, 11 hydrogènes, un nitrogène, et 2 oxygènes.

En réalité, une même formule chimique peut mener à la construction de plusieurs molécules différentes. C'est ce qu'on appelle, des *isomères*. Les isomères sont donc des molécules partageant la même formule chimique, et qui sont donc composées d'exactement des mêmes atomes. La différence entre eux se trouve dans l'arrangement de ces atomes dans l'espace. Ce qui peut également entraîner des propriétés chimiques différentes.

Un exemple très connu est celui de la formule chimique C_2H_6O qui correspond d'une part à la molécule d'éthanol, qu'on retrouve dans les boissons alcoolisées, et d'autre part à la molécule de méthanol, qui est un gaz mortel. La seule différence entre ces molécules, autre que leur effet sur notre corps, est que l'éthanol possède une chaîne carbonnée plus longue que celle du méthanol 2

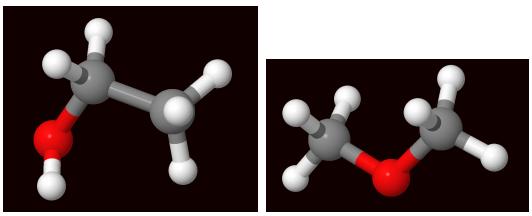


Figure 2: Molécule de formule chimique C_2H_6O

Finalement, notre objectif est de générer **toutes** les structures moléculaires possibles, et donc d'obtenir l'ensemble des isomères issus de la composition chimique donnée en entrée. Pour ce faire, nous allons utiliser le formalisme CSP dont nous avons parlé dans l'introduction. Nous devons donc définir à quoi correspondent les variables et les contraintes de chaque instance. Une fois le formalisme établi, il nous suffit de générer des instances pour chaque formule chimique donnée, puis de donner l'instance correspondante à un solveur, dans notre cas le solveur `choco`. Nous devons simplement lui préciser que nous souhaitons **toutes** les solutions possibles. Ainsi, il comprendra qu'il doit résoudre un problème d'énumération, et nous renverra chaque solution qu'il aura trouvé.

2.1 Modelisation à travers le formalisme CSP

En entrée, nous possédons les informations suivantes. Premièrement, nous connaissons le nombre total d'atome présent dans la molécule, qu'on note n. Chaque atome possède un numéro, allant de 0 à $n-1$. Nous connaissons aussi la liste des types d'atome de chacun d'entre eux, on note $type(i)$ le type de l'atome numéroté i. Et pour finir, on connaît aussi la valence de chaque atome, qu'on note $valence(i)$ pour l'atome numéro i.

Pour chaque instance, nous aurons une variable de graphe. Nous pouvons utiliser directement une variable de ce type puisque la bibliothèque `chocosolver` que nous utilisons pour la modélisation et la résolution de notre problème implémente ce type d'objet [6]. On note cette variable $G_{moc} = (V, E)$ où V est l'ensemble des sommets, correspondant aux atomes dans notre cas, et E l'ensemble des arêtes, qui correspondent à nos liaisons.

Formellement, on note $\mathcal{P} = (X, D, C)$ une instance du problème de génération de structure moléculaire d'une formule chimique donnée, avec :

- $X = \{G_{moc}\}$ où :
  - G_{moc} représente une variable de graphe, composé de n sommets.
- D l'ensemble des graphes qui sont des sous-graphes du graphe complet à n sommets, noté K_n.
- C l'ensemble des contraintes suivantes :
  - **La contrainte induite par la valence des atomes**. Pour chaque sommet, on souhaite que son dégré dans le graphe soit égal à la valence de l'atome auquel il correspond :
    $\forall i \in \{0, 1, .., n-1\},\ degree(i) = valence(i)$, où $degree(i)$ correspond au degré du sommet i dans le graphe.
  - **La contrainte de connexité du graphe**. Pour chaque paire de sommets (i, j), on souhaite qu'il existe une chaîne reliant ces deux sommets.

Pour la partie pratique, la contrainte de connexité est une contrainte qui était déjà implémentée dans la bibliothèque `chocosolver`. Pour ce qui est de la contrainte sur les valences, nous avons une contrainte permettant de définir le degré de chaque sommet.

2.2 Problème d'isomorphisme : plusieurs permutations pour une seule structure

Lorsque nous avons mis en place notre première modélisation, et que nous avons commencé à donner nos instances CSP au solveur, nous avons très vite compris que nous avions un problème. En effet, nos premiers lancements nous donnaient plusieurs centaines de solutions, même pour de petites formules moléculaires. Parmi toutes ces solutions trouvées, beaucoup réprésentaient exactement la même structure moléculaire. En effet, c'est ce qu'on appelle *le problème d'isomorphisme des graphes*. C'est un problème très connu dans le domaine de la théorie des graphes.

Ce problème est particulièrement réputé en raison de son statut indéfini sur le plan de la complexité algorithmique. Jusqu'à présent, il n'a été ni démontré qu'il peut être résolu en temps polynomial, c'est-à-dire qu'il appartient à la classe P, ni confirmé comme appartenant à la catégorie des problèmes NP. Par conséquent, il semble se positionner dans une sorte de zone grise, souvent associée à la classe NP-intermédiaire, qui englobe les problèmes dont le statut exact reste encore à clarifier. [5].

Pour résoudre ce problème, nous retrouvons une multitude de méthodes dans la littérature scientifique. Nos recherches nous ont menés vers l'algorithme VF2 [3] qui permet la détection d'isomorphisme de graphes. Cet algorithme procède de manière incrémentale pour construire une potentielle correspondance entre les graphes en ajoutant un noeud à la fois. De plus, il réduit l'espace de recherche en utilisant des règles de faisabilité qui éliminent les correspondances qui ne peuvent pas mener à un isomorphisme complet. Si la correspondance courante ne peut pas aboutir, l'algorithme utilise la technique de *backtracking* pour tenter une nouvelle approche.

Pour utiliser cet algorithme, nous avons choisi d'utiliser la bibliothèque `JGraphT` [8]. Cette bibliothèque implémente plusieurs algorithmes de la théorie des graphes, dont l'algorithme VF2. Par conséquent, cet algorithme nous permet de vérifier chaque nouvelle structure de graphe que nous découvrons pour s'assurer qu'elle ne correspond pas à un graphe isomorphe à un de ceux déjà identifiés. Si nous confirmons qu'elle est unique, alors nous l'ajoutons à notre collection de graphes. Au final, cette méthode nous permet de compiler une liste exhaustive des différents isomères, chacun représentant une structure moléculaire distincte avec ses propres caractéristiques.

3 L'ajout de la gestion des doubles et triples liaisons

Jusque-là, nous n'avons pris en compte que des molécules contenant des liaisons simples. Or, dans le monde réel, les molécules peuvent avoir des liaisons doubles ou triples. Par ailleurs, rappelons que l'objectif de ce projet est de fournir à des chimistes une application permettant la génération automatique de molécules en partant d'une formule chimique. Se limiter aux liaisons simples empêche la prise en compte d'une multitude de molécules, ce qui limite l'aide que nous pourrions apporter aux recherches des chimistes.

Pour gérer les doubles et triples liaisons, nous prenons les mêmes données que pour la première modélisation, à laquelle on ajoute diverses variables et contraintes qui sont les suivantes.

- Pour les variables :
  Nous ajoutons une matrice comportant les diverses liaisons possibles. On note cette matrice *liaisons*. Elle est de taille $n \times n$ où n est le nombre d'atomes présent dans la formule chimique. On interprète la

matrice *liaisons* comme ceci :
Pour tout atome i et j de notre molécule, $i \neq j$:

$$liaisons(i,j) = \begin{cases} 0 & \text{si } i \text{ et } j \text{ ne sont pas en liaisons} \\ 1 & \text{si } i \text{ et } j \text{ sont reliés par une liaison } \textbf{simple} \\ 2 & \text{si } i \text{ et } j \text{ sont reliés par une liaison } \textbf{double} \\ 3 & \text{si } i \text{ et } j \text{ sont reliés par une liaison } \textbf{triple} \end{cases}$$

- Pour les contraintes :
  - **Contraintes sur le nombre de liaisons de chaque atome** : Pour chaque atome $i \in V$,

$$\sum_{j=0}^{n-1} \text{liaisons}(i,j) = \text{valence}(i)$$

  - **Contraintes sur les arêtes du graphe en fonctions des liaisons** : Pour chaque paires de sommet $i, j \in V$,

$$liaisons(i,j) > 0 \Rightarrow (i,j) \in E$$

Pour définir la matrice *liaisons*, nous avons dû utiliser un tableau *IntVar* de taille $n \times n$ dont chaque élément peut prendre la valeur 0,1,2 ou 3.
Une fois la matrice *liaisons* déterminée par notre modèle, nous avons utilisé les *channeling constraints*(ou contraintes de canaux). Ce type de contrainte nous a permis de synchroniser la présence d'arête dans le graphe G_{mol} avec les valeurs des liaisons. Ce mécanisme est possible grâce à la réification de contrainte. La réification de contraintes consiste à associer à une contrainte une variable booléenne qui représente l'état de cette contrainte. Ce type de contrainte est implémenté dans la bibliothèque `chocosolver`. Nous avons mis le mécanisme en place de la manière suivante. Nous avons une matrice de booléens $aretePresente$ telle que $\forall i,j \ \ aretePresente(i,j) = (liaisons(i,j) > 0)$ et $aretePresente(i,j) \Rightarrow (i,j) \in E$. Cette dernière contrainte est mise en place grâce à l'implémentation de la méthode `edgeChanneling` de `chocosolver`.

4 L'ajout de la gestion automatique du placement dans l'espace

Lors des premières réunions avec M.Prcovic, nous avons établi qu'il n'était pas suffisant de générer la structure de graphe pour obtenir la structure moléculaire. En effet, pour qu'une molécule soit chimiquement correcte, les atomes liés doivent respecter un certain intervalle de distances. De même, pour que deux atomes ne soientt pas liés, leur distance doit être à l'extérieur de l'intervalle de distance entre leur 2 types.

En théorie [10], il existe une distance minimale en dessous de laquelle deux atomes ne sont généralement pas en liaison. En pratique, les atomes non liés peuvent se trouver proches les uns des autres, mais ils ne sont pas aussi proches que les atomes qui forment une liaison chimique. C'est pourquoi, dans notre modélisation nous exigeons que deux atomes non-liés soient à une distance minimale, sans prendre en compte qu'en dessous d'une certaine autre valeur de distance ils puissent être liés. Cela permet de limiter l'aspect combinatoire qui est déjà très grand.

Les distances de liaison diffèrent également selon le type de liaison. Une liaison double entre deux atomes n'aura pas le même intervalle de distance qu'une liaison simple ou triple entre ces deux mêmes atomes.

En conclusion, afin de confirmer que notre structure de graphe représente effectivement une structure moléculaire viable, notre objectif est de déterminer s'il existe un agencement spatial des atomes qui respecte les contraintes de distance préalablement établies. Cela implique de trouver un arrangement où chaque atome est positionné de manière à ce que les distances entre eux soient conformes aux liaisons et à la géométrie moléculaire définies.

4.1 Nouvelle modélisation du problème

Notre première modélisation offre une base solide et ne doit pas être entièrement écartée. En réalité, elle nécessite simplement quelques ajustements, notamment l'ajout de variables supplémentaires et de contraintes. Initialement, nous abordons cette question comme un problème de satisfaction de contraintes (CSP), visant à identifier les structures graphiques moléculaires qui correspondent à une liste donnée d'atomes. Ensuite, l'étape suivante consiste à trouver un moyen de positionner chacune de ces structures dans l'espace de manière cohérente.

Effectivement, pour chaque structure moléculaire, il y a potentiellement une multitude de manières de la positionner dans l'espace, en raison des symétries, rotations, et autres transformations géométriques qui offrent

une grande variété d'options. Toutefois, notre objectif principal n'est pas de trouver un placement quelconque, mais plutôt de déterminer le meilleur placement possible. La définition de ce "meilleur" placement peut varier selon différents critères, que nous allons examiner plus en détail ultérieurement.

En fin de compte, notre tâche se transforme en un problème d'optimisation sous contraintes (COP), où l'objectif est de trouver le placement le plus optimal de la structure dans l'espace, tout en respectant les contraintes spécifiques liées à la structure moléculaire.

Ainsi, nous avons deux modélisations pour ce même problème. Schématiquement, on crée la modélisation via le formalisme CSP, définie dans la section précédente, pour laquelle on va chercher à énumérer toutes les solutions. Pour chaque solution obtenue, on crée une modélisation avec le formalisme COP, correspondant au problème énoncé précédemment, pour laquelle on va chercher une seule solution, qui sera la meilleure, au sens que l'on aura défini.

4.1.1 Modélisation pour le placement dans l'espace

On possède les mêmes données que dans la modélisation précédente. En plus, nous avons accès à la structure de graphe trouvée grâce à la première modélisation, notée $G_{moc} = (V, E)$. Nous avons aussi à disposition les distances de liaison. On note $dist\_max(p, q)$ la distance maximum que peuvent avoir deux atomes de types respectivement p et q pour être en liaison, et $dist\_min(p, q)$ la distance minimum que doivent avoir deux atomes de types p et q pour être en liaisons.

On note $\mathcal{P} = (X, D, C)$ une instance du problème avec :

- $X = \{x_i | i \in V\} \cup \{y_i | i \in V\} \cup \{z_i | i \in V\}$ où :
  - x_i est la coordonnée de la première dimension du sommet i du graphe G_{moc}.
  - y_i est la coordonnée de la deuxième dimension du sommet i du graphe G_{moc}.
  - z_i est la coordonnée de la troisième dimension du sommet i du graphe G_{moc}.
- $D = \{d_{x_i} | i \in V\} \cup \{d_{y_i} | i \in V\} \cup \{d_{z_i} | i \in V\}$ où :
  - $d_{x_i} = [-300; 300],\ \forall i$
  - $d_{y_i} = [-300; 300],\ \forall i$
  - $d_{z_i} = [-300; 300],\ \forall i$
- C : l'ensemble des contraintes suivantes.
  - **Les contraintes sur les distances minimum entre chaque atomes liés :**
    Si $(i, j) \in E$, alors $distance(i, j) \geq dist\_min(type(i))(type(j))$
  - **Les contraintes sur les distances maximum entre chaque atomes liés :**
    Si $(i, j) \in E$, alors $distance(i, j) \leq dist\_max(type(i))(type(j))$
  - **Les contraintes sur les distances minimum entre les atomes non-liés :**
    Si $(i, j) \notin E$, alors $distance(i, j) > dist\_max(type(i))(type(j))$

Pour l'implémentation, nous avons utilisé un nouveau type de variable qui sont des variables réelles, `RealVar` dans la bibliothèque `chocosolver`. C'est la première fois que nous manipulions ce type de variable dans le contexte de la programmation par contraintes. Elles offrent une multitude de contraintes possibles, comme les contraintes d'équations et d'inéquations [4] dont nous nous sommes servis pour modéliser nos contraintes de distances.

4.2 Optimisation

En théorie, avec ce que nous avons dit, il nous suffit de trouver un seul placement possible dans l'espace pour confirmer, ou non, si le graphe étudié correspond à une structure moléculaire correcte.

Cependant, la réalité est plus complexe. En effet, dans une molécule, les atomes s'organisent de manière à atteindre un état de stabilité maximale. Cette organisation implique une minimisation des répulsions entre les atomes tout en respectant les angles de liaison et la géométrie moléculaire appropriés. Chaque atome est entouré d'électrons, et ces électrons se repoussent mutuellement. Afin de réduire cette répulsion, les atomes s'arrangent de telle sorte que leurs électrons soient positionnés aussi loin les uns des autres que possible. Cet agencement contribue à définir la forme tridimensionnelle de la molécule, qui est essentielle pour comprendre ses propriétés et son comportement.

Cependant, le manque de temps, et de connaissances dans le domaine de la chimie moléculaire, nous limite grandement. Pour mettre en place une optimisation représentative de la réalité, il nous faudrait détailler

beaucoup plus la modélisation initiale, en intégrant notamment des réprésentations pour les électrons et autres composants d'un atome. En effet, nous nous sommes limités à la représentation d'un atome comme un sommet d'un graphe, mais la réalité des choses est bien plus complexe.

Il semble donc clair que l'optimisation des coordonnées soit une piste à creuser. Même si nous ne pouvons pas optimiser celles-ci comme on le devrait, nous pouvons choisir d'établir notre propre critère d'optimisation. Cela nous permet d'avoir une première approche d'optimisation, qui pourrait être complétée plus tard avec une description de la modélisation plus bas niveau, et complète, sur les structures des atomes. De plus, il peut être intéressant d'optimiser les placements des molécules de telle manière à ce que l'affichage soit plus agréable. En effet, sur la figure 3, nous voyons bien que le placement de la molécule sur l'image de droite semble plus adéquate.

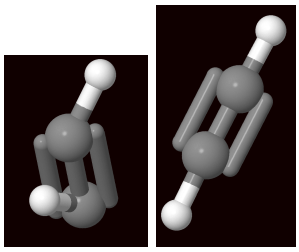


Figure 3: Une même molécule avec deux placement dans l'espace différent

De plus, il est important de noter que dans les contraintes que nous avons énoncées précédemment, nous ne prenons pas en compte les croisements possibles. Cette contrainte est compliquée à établir en pratique. En optimisant le placement de nos molécules, nous pouvons poser cette contrainte indirectement.

Pour commencer, il faut alors établir ce que nous appelons un placement optimisé. Pour ce faire, plusieurs méthodes sont possibles. Nous avons décidé de chercher à maximiser les distances entres les atomes, afin que la molécule prennent le plus d'espace possible.

En pratique, nous avons simplement à ajouter une *fonction objectif*, en précisant que l'on souhaite la maximiser. Cette fonction est ensuite ajoutée au modèle, puis il faut itérer sur toutes les valeurs possibles, et la dernière valeur sera celle maximisant notre fonction. Nous définissons notre fonction de la manière suivante :

- Soit f la fonction objectif à maximiser. Ici, on cherche à maximiser les écarts entres les atomes. Pour ce faire, on va chercher à maximiser la valeur de la plus petite distance :

$$f = \min_{i,j} distance(i,j)$$

La fonction objective que l'on souhaite peut aussi être définie comme la somme des distances des atomes. Ainsi, elle serait définie de la manière suivante :

- Soit f la fonction objectif à maximiser la somme des distances entre atomes.

$$f = \sum_{i<j} distance(i,j)$$

Pour implémenter cette fonction, nous avons dû manipuler une nouvelle bibliothèque, étroitement lié à `chocosolver`, qui est `Ibex` [2].

4.3 Problème de symétrie et de combinatoire

En théorie, pour connaître la meilleure solution, au sens de notre critère d'optimisation, il faut d'abord trouver toutes les solutions, et ensuite constater laquelle elle est la meilleure.

En pratique, il existe des méthodes d'optimisation permettant d'éviter de parcourir l'ensemble des solutions. Choco utilise une technique appelée *Branch and Bound* (BnB). Cette méthode consiste à explorer l'espace des solutions en créant un arbre de recherche. D'une part, on divise l'espace des solutions en sous-ensembles plus petit qui peuvent être évalués et résolus séparément, c'est l'aspect *branch(ing)*. Et d'autre part, à chaque noeud de l'arbre, le solveur évalue la fonction objectif et utilise ces informations pour élaguer les branches de l'arbre qui ne peuvent pas mener à une meilleure solution, c'est l'aspect *bound(ing)*. De plus, pendant la recherche, Choco utilise la propagation des contraintes pour réduire l'espace des solutions possibles. Si une contrainte ne peut pas être satisfaite, le solveur élimine les valeurs qui ne sont plus possibles pour certaines variables.

Néanmoins, ces optimisations ne suffisent pas. En effet, lors de nos expériences, nous avons constaté que le temps d'exécutions pouvaient être extrement long pour obtenir un résultat.

De plus, lorsque nous avons défini nos variables de distance, nous avons pu fournir la précision à laquelle nous voulions les valeurs. Cette valeur joue un rôle important puisque plus la précision est fine, c'est-à-dire que sa valeur est petite, plus l'on va être précisie dans les coordonées, et donc plus on aura de coordonnées possible. Ainsi, l'espace de recherche va croitre lorsque la précision augmente en finesse. Il ne faut évidemment pas choisir une valeur de précision trop grande, sinon les coordonnées vont perdre tout sens logique, et seront loin de représenter des coordonnées plossibles.

Nous avons passé sous silence un problème que nous avons rencontré, et anticipé : le problème des symétries. En effet, il existe une grande quantité de possibilités de placement dans l'espace lorsque les transformations géométriques rentrent en jeu. Pour pallier au problème, nous avons constraint certaines valeurs de variables :

- Le premier atome de la liste est fixé à l'origine du repère. Ainsi, on pose $x(0) = 0,\ y(0) = 0,\ z(0) = 0$.
- Le deuxième atome est forcé à être sur l'axe des z, du côté positif. Autrement dit, $x(1) = 0$ et $y(1) = 0$. Seule la valeur $z(0)$ est à chercher, tout en la forçant à être positive.
- Le troisième atome est forcé à être sur l'axe des y, du côté positif aussi. Ainsi, $x(2) = 0$, et on force $y(2)$ a être positive.
- Pour finir, on fixe le quatrième atome à être dans le demi-espace des x positifs. Seule la valeur de $x(3)$ est contrainte à être positive.

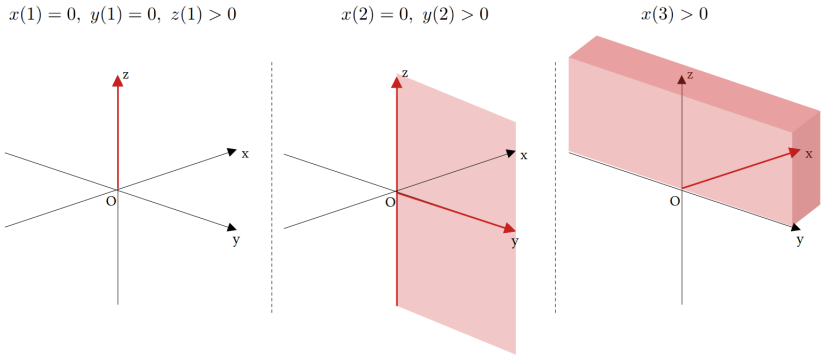


Figure 4: Schéma du mécanisme permettant de casser les symétries

Le premier point permet d'être insensible au translation dans l'espace. Les autres points quant à eux, permettent de casser les symétries axiales et les rotations.

5 Détails d'implémentation

Pour obtenir des informations détaillées sur la structure du code et son organisation, nous vous invitons à visiter le dépôt GitHub de notre projet. Afin de ne pas surcharger le rapport, nous avons décidé de tout mettre dans un fichier README.md exhaustif qui fournit toutes les explications nécessaires sur les différentes classes : cliquez ici.

5.1 Description du format d'entrée

Pour les données d'entrée de notre programme, nous avons opté pour le format JSON [14], en raison de sa simplicité et sa facilité d'utilisation dans les applications de programmation. C'est un format souvent utilisé, et celui qui nous semblait le plus simple à adapter pour une potentielle interface graphique plus tard. De plus, la lecture de ce format est très simple.

Voici comment se composent nos fichiers d'entrée :

```
{
    "types": [type1, ...],
    "quantities": [nb_atome_type1, ...],
    "structure": [[atome1, type_liaison, atome2], ...]
}
```

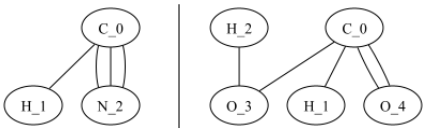


Figure 5: Exemples de visualisation grâce à `GraphViz`

Le premier élement correspond à une liste des types d'atome que l'on souhaite mettre dans notre molécule. Cela correspond finalement au nom de l'atome. Par exemple "O" pour oxygène, "C" pour carbone, etc. Le deuxième élement correspond à la liste des valeurs de quantités de chaque type d'atome.Par exemple, `nb_atom_type1` correspond au nombre d'atome de type `type1`. Et finalement, le dernier élement est `structure` qui est une liste, potentiellement vide, de liaison qu'on souhaite absolument avoir dans notre structure de molécule. Cette liste est composé de une ou plusieurs liste de la forme `[atome1, type_liaison, atome2]` qui traduit le fait que l'on veut que les atomes d'indice `atome1` et `atome2` soient liés par une liaison de type `type_liaison`, dans notre cas, soit par une liaison simple, soit pas une liaison double. On rappelle que les atomes sont indicés de la manière suivante : les atomes de 0 à `nb_atome_type1`-1 correspondent aux atomes de type `type1` etc.

5.2 Description du format de sortie

Comme nous l'avons expliqué dans le rapport, notre travail s'est déroulé en deux grands temps. Premièrement, nous avons uniquement travaillé à la génération de structure de graphe représentant une potentielle structure moléculaire. Dans un second temps, nous avons souhaité placer nos molécules dans l'espace. Ainsi, nous avons travailler sur deux types de visualisation.

5.2.1 Visualisation des structures de graphe

Dans le but de faciliter l'analyse visuelle des graphes générés, nous avons décidé de générer des images correspondantes aux graphes en question. Nous commençons donc par modéliser, puis résoudre, le problème de la génération des graphes de molécules. Pour chaque solution de graphe trouvée, nous générons une représentation visuelle à l'aide de la bibliothèque Graphviz [7]. Cette représentation graphique offre une vue d'ensemble des connexions, simples, doubles, ou triples, entre les atomes, sans se préoccuper des aspects géométriques précis. Vous retrouvez un premier *main* dans la classe `MainViz` permettant l'exécution de ce mécanisme.

5.2.2 Visualisation des molécules dans l'espace

Après avoir réussi à implémenter la modélisation de molécule dans l'espace, nous souhaitons avoir un outil permettant leur visualisation. Pour ce faire, sous les conseils des chimistes, nous avons utilisé le logiciel `JMol` [9]. C'est un visualiseur de molécules *open source* qui est largement utilisé la recherche pour visualiser les structures moléculaires en trois dimensions.

Pour pouvoir l'utiliser, nous devions généré des fichiers au format *Chemical Markup Language* (CML). Le CML est un langage basé sur XML spécifiquement conçu pour la description de structures chimiques. Il permet une représentation détaillée des molécules, incluant les atomes, liaisons, et coordonnées spatiales.

La classe `CMLgenerator` est responsable de la conversion des structures moléculaires générées par notre programme en documents CML. Cette classe crée des éléments XML pour chaque composant de la molécule et les assemble en une structure CML conforme.

Une fois le fichier au format CML généré, il suffit de l'ouvrir dans le logiciel Jmol, et nous obtenons la visualisation de nos molécules. A noter qu'il est possible d'optimiser le placement des atomes directement dans le logiciel, afin d'obtenir la visusalisation la plus représentative de la réalité.

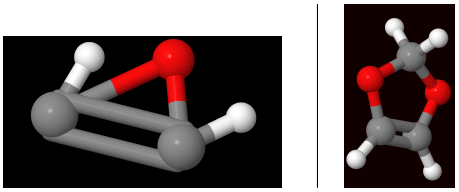


Figure 6: Exemples de visualisation grâce au logiciel `JMol`

5.3 Gestion des experiences

Pour rendre les expériences plus fluide et pratique, nous avons développé deux classes pour automatiser la création de molécules et l'évaluation de performances de notre programme : `InstanceMaker` et `ExperienceMaker`.

La classe `InstanceMaker` est dédiée à la génération automatique de fichiers d'entrée au format JSON présenté précédemment. Elle prend en paramètre une chaîne de caractères représentant les atomes et leurs quantités, et produit un fichier JSON structuré correspondant. Ce fichier sert ensuite d'entrée pour la génération de structures moléculaires. La classe permet une initialisation rapide et efficace de diverses configurations moléculaires pour des tests et analyses ultérieurs.

La classe `ExperienceMaker` permet de réaliser des expériences sur plusieurs configurations moléculaires en automatisant le processus de création d'instances, l'exécution du programme principal,et la collecte des résultats. En entrée, elle reçoit une liste d'atomes, pour lesquels elle génère différentes instances via `InstanceMaker`. Ensuite, elle exécute le Main.java pour chaque instance, collecte les résultats tels que les fichiers CML générés et les statistiques de performance (par exemple, le nombre de molécules trouvées et le temps d'exécution). Un rapport résumant les résultats de l'expérience est produit contenant le nombre de molécule et le temps d'exécution pour chaque atome, ce qui permet d'avoir une vue d'ensemble sur les expériences et d'accélérer le processus.

6 Résultats d'expériences

Pour simplifier, et éviter d'alourdir les tableaux de résultats, nous avons fait le choix de représenter uniquement des molécules composées d'atomes d'hydrogènes (H), d'oxygènes (O), et de carbones (C). Cependant, nos implémentations fonctionnent pour tout type de molécule, à condition que les distances de liaisons des atomes de la molécule d'entrée soient renseignées dans le fichier `./data/distances`.

Les expériences présentées ont été réalisées sur un ordinateur possédant la configuration matérielle suivante : CPU : 11th Gen Intel(R) Core(TM) i9-11950H @ 2.60GHz avec une RAM de 117 Gi.

Dans le tableau ci-dessous, nous présentons une sélection de résultats obtenus lors de nos expériences. Pour chaque molécule, nous affichons son nombre d'isomères trouvés et le nombre de graphes trouvés avant le filtrage des isomorphes. Vous retrouvez aussi le temps d'exécution total du programme, incluant modélisation et résolution, et le temps moyen de la résolution de l'instance permettant la génération des coordonnées des atomes. À noter que les résultats sont triés par temps d'exécution total croissant.

Nous avons fait le choix de ne pas montrer les temps d'exécution pour la partie modélisation et résolutions de graphes, car ils sont insignifiants comparés aux temps d'exécution pour la génération des coordonnées. À titre d'exemple, pour la molécule $C_4O_2H_2$, cela représente 3837 ms pour 522 0071 ms d'exécution totale.

Molécule	Nb isomère(s)	Nb graphe(s)	Temps total	Temps moyen par coordonnées
H_2O	1	1	406 ms	8 ms
CH_4O	2	4	521 ms	18 ms
C_2H_4	2	6	516 ms	14 ms
C_2O_2	4	5	636 ms	7 ms
CH_4O_2	3	20	652 ms	23 ms
C_2H_6O	3	140	878 ms	71 ms
CH_4O_3	4	120	1037 ms	80 ms
$C_2O_2H_2$	10	44	1161 ms	10 ms
C_3O_2	8	42	1335 ms	57 ms
$C_2H_2O_4$	11	274	1712 ms	49 ms
C_3OH_2	10	69	1961 ms	92 ms
H_2O_6	2	720	2187 ms	781 ms
C_4O_2	29	636	11881 ms	331 ms
$C_2O_2H_6$	6	1120	14319 ms	1120 ms
$C_3O_2H_2$	35	543	46567 ms	1252 ms
H_2O_8	2	40320	185232 ms	90482 ms
$C_4O_2H_2$	164	10488	522071 ms	3086 ms

Table 1: Résumé des expériences et des résultats obtenus sans optimisation des coordonnées

Ce premier tableau 1 représente des expériences faites pour la modélisation via le formalisme CSP, et donc sans optimisation. Nous pouvons observer plusieurs phénomènes.

Premièrement, le problème d'isomorphisme est bien visible dans les résultats. En effet, le nombre de graphes avant filtrage des isomorphes peut très vite exploser. Cela montre l'importance du filtrage pour l'optimisation

de la complexité de calcul.

Ensuite, l'augmentation des temps d'exécution est plus influencé par le nombre d'atomes dans la molécule que par le nombre d'isomères. Pour la molécule $C_3O_2H_2$, composé de 7 atomes et correspondant à 35 isomères, le temps d'exécution est plus faible que le temps d'exécution de la molécule H_2O_8 qui possède 10 atomes et correspond à seulement 2 isomères. La différence du nombre d'atomes ne semble pas si grande, puisque égale à 3, alors que la différence de quantités d'isomère est importante. Alors que nous aurions pu penser que le nombre d'isomères ait une grande influence, cela ne semble pas être autant le cas. Cependant, le nombre d'atomes, lui, a un impact. On observe donc que notre modèle est limité par le nombre d'atomes de la molécule. Cela provient certainement du fait que l'on utilise des variables réelles, ce qui complique la recherche à cause de la large plage de valeurs possibles. Nous avions tenté de réduire les intervalles, mais cela n'améliore pas de manière significative les temps d'exécution. De plus, nous pouvons choisir la précision des valeurs, que nous prenons à 1 pour les expériences présentées. Ce paramètre permet de choisir à quel point la valeur de nos variables peut être précis. En réalité, à la sortie du modèle, le solveur nous rend un intervalle de valeur. Ce résultat s'interprète par le fait que notre variable possède une valeur approximativement au centre de l'intervalle. Ainsi, plus la précision est grande, plus la plage de valeur est grande, et donc imprécise. Dans le cadre de ce projet, les valeurs de précision permettant une accélération de la résolution impliquaient des résultats irréalistes. Finalement, le nombre d'atomes complexifie le choix des coordonnées, alors que le nombre d'isomères augmente le nombre d'exécutions de la modélisation pour la génération des coordonnées. Il est donc logique que le nombre d'isomères soit moins impactant dans les temps d'exécution.

Aussi, on remarque que les types d'atomes influencent le nombre d'isomères possibles. En effet, la présence des atomes de carbone et d'oxygène, par exemple, augmente les probabilités d'avoir de nombreux isomères. Cela vient du fait que plus la valence d'un atome est élevée, plus la diversité de liaisons possibles est grande, ainsi, les possibilités de configurations augmentent.

Nos tests ne se sont pas limités à ces molécules. En effet, nous avons tenté des expériences à grande échelle en partant de molécules bien connues telle que la caféine $C_8H_10N_4O_2$. Nous avons tenté de laisser tourner des heures sans avoir de résultats. En réalité, au-delà de 10 molécules, nous n'avons pas obtenu de résultats en dessous de 10 minutes.

Nous avons ensuite voulu tester notre modélisation via le formalisme COP, donc avec optimisation. Les résultats sont représentés dans le tableau suivant 2.

Molécule	**Nb isomère(s)**	**Nb graphe(s)**	**Temps total**	**Temps moyen par coordonnées**
H_2O	1	1	445 ms	31 ms
CO_2	1	1	707 ms	283 ms
CH_2O	1	1	6988 ms	6569 ms
H_2O_2	2	2	8787 ms	4152 ms
C_2H_2O	2	2	11744 ms	5631 ms

Table 2: Résumé des expériences et des résultats obtenus avec optmisation des coordonnées

La taille du tableau représente le problème combinatoire sous-jacent. Optimiser les coordonnées implique un très grand nombre de calculs, ce qui rend les temps d'exécution vite très grands. Ainsi, nombreuses sont les formules moléculaires qui n'ont pas donné de résultats. Pour les expériences, nous avons limité le temps de résolution à 10 minutes, soit 600 000 ms. Même des molécules qui étaient rapides lors de la résolution sans optimisation n'ont pas abouti au delà de ce temps. Nos tests sur cette modélisation ont été très limités par la combinatoire. Pour diminuer la combinatoire, nous pourrions étudier les différentes propriétés chimiques des molécules afin de déduire de nouvelles contraintes afin de limiter la taille des espaces de recherche. Ainsi, les résolutions pourraient-être plus rapide.

7 Conclusion

L'objectif du projet était de générer automatiquement des structures moléculaires. Pour ce faire, nous avons utilisé une approche par programmation par contrainte. Nous avons donc procédé en deux étapes. Premièrement, nous avons dû modéliser le problème. Puis, nous avons dû résoudre les instances en les donnant à un solver, dans notre cas Choco.

Au cours de ce projet, nous avons rencontré diverses difficultés. Tout d'abord, sur les aspects techniques. D'une part, la documentation de la bibliothèque `chocosolver` s'est souvent révélée incomplète, ce qui a compliqué l'appropriation de cet outil.

Un autre problème fut, comme mentionné précédemment, le temps de calcul. La complexité combinatoire du problème a entraîné un grand nombre d'opérations, allongeant ainsi le temps nécessaire pour obtenir des résultats. Or, dans le cadre de ce projet, nous devions pouvoir fournir des solutions assez rapidement puisque

le but était de fournir un outil aux chimistes pour modéliser des molécules automatiquement. Il n'est donc pas imaginable d'attendre plusieurs heures avant d'avoir une réponse à leur requête. L'intégration d'une interface graphique dédiée à notre application pour la visualisation tridimensionnelle des molécules serait également une avancée significative, nous évitant le recours à un logiciel externe.

La complexité théorique de notre sujet, qui se situe à l'intersection de deux disciplines, qui sont la chimie et l'informatique, a posé certains défis. Nous avons été contraints de simplifier le problème initial. Comme nous l'avons mentionné dans le rapport, notre représentation des molécules, et plus particulièrement des atomes, a été réduite à une entité unique. Cependant, un atome est en réalité un ensemble complexe composé d'électrons, de protons et de neutrons. Pour affiner notre modèle et le rapprocher davantage de la réalité, une approche plus détaillée serait de décomposer la structure de l'atome et d'utiliser les propriétés spécifiques de chaque composant pour introduire de nouvelles contraintes, en particulier en ce qui concerne les liaisons interatomiques. De plus, même sans complexifier la notion d'atome, nous pourrions enrichir notre modèle en intégrant davantage les propriétés chimiques des molécules.

Par exemple, l'introduction de contraintes angulaires pour trois atomes liés consécutivement pourrait constituer une amélioration significative du modèle. De même, la modélisation des conditions spécifiques sous lesquelles certains types de liaisons, comme les liaisons double, se forment automatiquement, pourrait offrir une représentation plus fidèle de la réalité chimique. Nos connaissances limitées en chimie nous ont obligés à simplifier notre modèle, réduisant ainsi sa complexité par rapport à la théorie réelle. Approfondir notre compréhension de la chimie serait certainement bénéfique pour le développement et la réussite du projet, en nous permettant d'intégrer des aspects plus nuancés et précis de la chimie moléculaire dans notre modèle.

Ce projet a constitué une opportunité précieuse pour nous initier à l'utilisation du solveur Choco et à la maîtrise de la bibliothèque de modélisation et de résolution 'chocosolver'. De plus, il a permis d'enrichir notre expérience dans le domaine de l'intelligence artificielle, s'étendant au-delà des frontières de l'apprentissage automatique, un domaine que nous avions déjà eu l'opportunité d'explorer en profondeur durant notre parcours de master. Ce projet nous a offert la chance de nous aventurer plus profondément dans le secteur de la programmation par contrainte et de la chimie, en appliquant et renforçant nos connaissances et nos compétences.

References

[1] a.l Adrien Varet. *BenzAI*. https://hal.science/hal-03691364/document.

[2] *Bibliothèque Ibex pour les contraintes dynamiques*. http://ibex-team.github.io/ibex-lib/.

[3] et a.l. Cordella LP Foggia P. *A (sub)graph isomorphism algorithm for matching large graphs*. IEEE Trans Pattern Anal Mach Intell. https://ieeexplore.ieee.org/document/1323804. 2004.

[4] COSLING. *Contraintes sur les variables réelles de chocosolver*. https://choco-solver.org/docs/modeling/realconstraints/.

[5] Martin Grohe et Daniel Neuen. *Recent advances on the graph isomorphism problem*. https://www.cambridge.org/core/books/abs/surveys-in-combinatorics-2021/recent-advances-on-the-graph-isomorphism-problem/2966D84DB2220F6A305472EE73B7973C. 2021.

[6] Javadoc des variables de graphe. *Chocosolver - COSLING*. https://javadoc.io/doc/org.choco-solver/choco-solver/latest/org.chocosolver.solver/org/chocosolver/solver/variables/GraphVar.html.

[7] Graphviz. *Bibliothèque de modélisation de graphe*. https://graphviz.org/.

[8] JGraphT. *Bibliothèque d'algorithmie sur les structures de type graphe*. https://jgrapht.org/.

[9] *Logiciel JMol*. https://jmol.sourceforge.net/.

[10] Georgia Tech Loren Dean Williams Professor Chemistry Biochemistry. *Molecular Interactions*. https://williams.chemistry.gatech.edu/structure/molecular_interactions/mol_int.html.

[11] U. Montanari. *Network of constraints : Fundamental properties and applications to picture processing. Information Science, 7 :95–132*. https://www.sciencedirect.com/science/article/abs/pii/0020025574900085. 1974.

[12] *Site de l'équipe COALA du LIS*. https://coala.lis-lab.fr/presentation/.

[13] Adrien Varet. *Thèse de Doctorat. Programmation par Contraintes et Chimie Théorique : Utilisation du Formalisme CSP pour Résoudre des Problématiques Liées aux Benzénoïdes*. https://www.theses.fr/2022AIXM0508. 13/12/2022.

[14] WIkipédia. *Format JSON*. https://en.wikipedia.org/wiki/JSON.

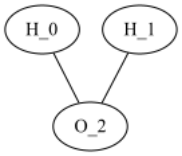
[15] WIkipédia. *Programmation par contraintes*. https://fr.wikipedia.org/wiki/Programmation_par_contraintes.

Annexes

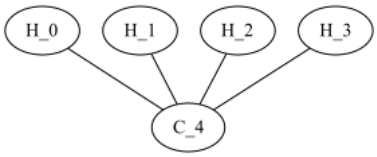
1. Exemples de graphes de molécules générés par le programme :

Ces molécules ne sont pas au format standard de la chimie (angle optimisés etc.) car généré avec graphviz. Cependant, si on met les fichier output dans le logiciel JMOL, nous obtenons des molécules avec un structure réaliste, mais moins visible sur un rapport en 2D.

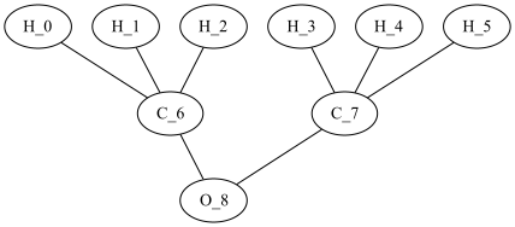
H_2O :

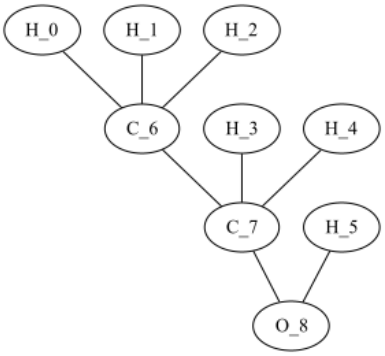


CH_4 :

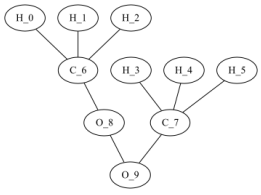


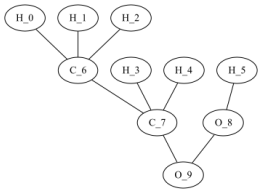
C_2H_6O :

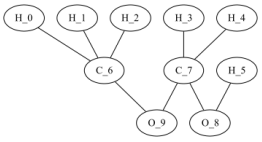


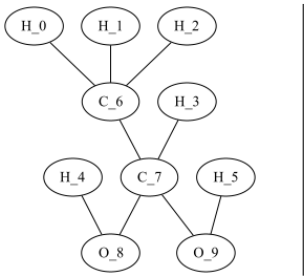


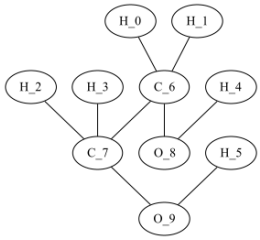
$C_2H_6O_2$:



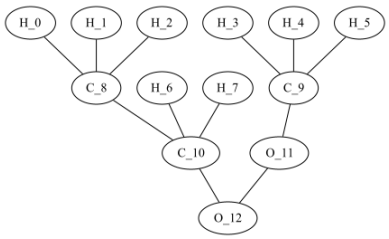


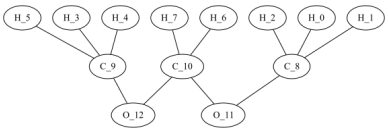


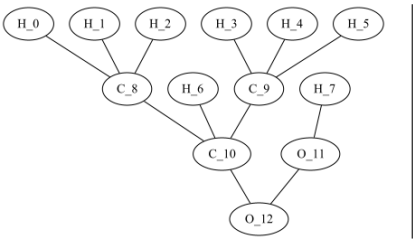


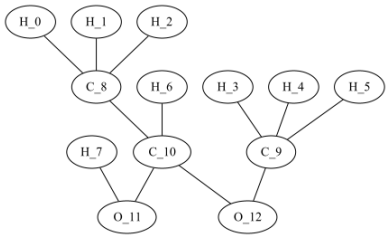


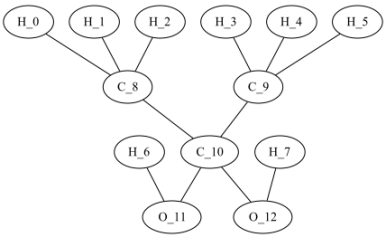
$C_3H_8O_2$:

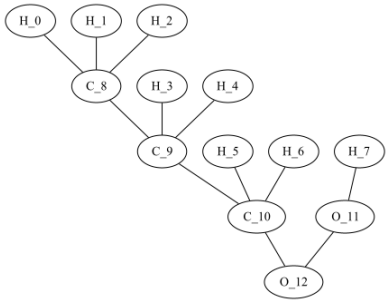

H_0
H_1
H_2
H_3
H_4
H_5
C_8
H_6
H_7
C_9
C_10
O_11
O_12

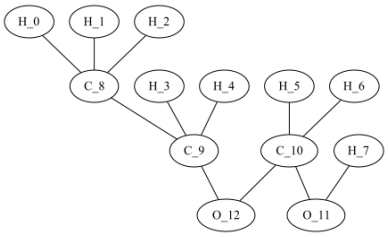

H_5
H_3
H_4
H_7
H_6
H_2
H_0
H_1
C_9
C_10
C_8
O_12
O_11

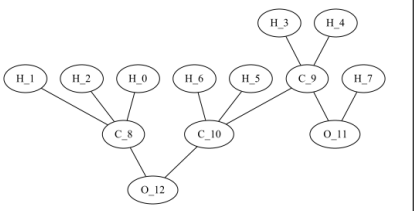

H_0
H_1
H_2
H_3
H_4
H_5
C_8
H_6
C_9
H_7
C_10
O_11
O_12

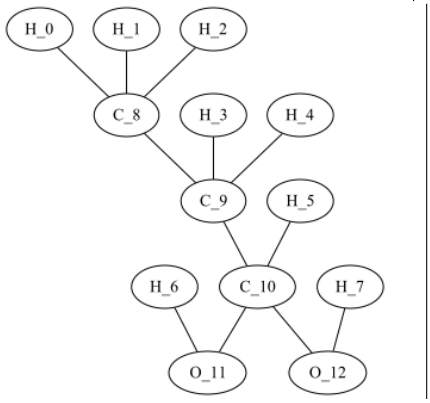

H_0
H_1
H_2
C_8
H_6
H_3
H_4
H_5
H_7
C_10
C_9
O_11
O_12

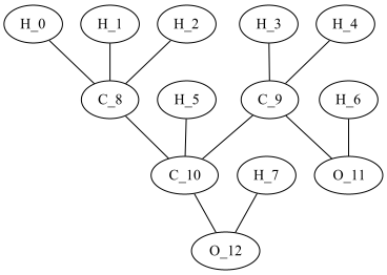

H_0
H_1
H_2
H_3
H_4
H_5
C_8
C_9
H_6
C_10
H_7
O_11
O_12

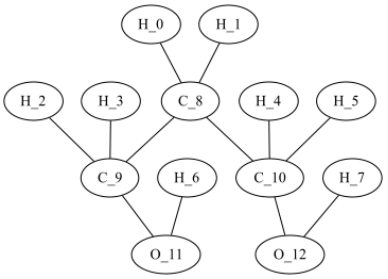

H_0
H_1
H_2
C_8
H_3
H_4
C_9
H_5
H_6
H_7
C_10
O_11
O_12


H_0
H_1
H_2
C_8
H_3
H_4
H_5
H_6
C_9
C_10
H_7
O_12
O_11


H_3
H_4
H_1
H_2
H_0
H_6
H_5
C_9
H_7
C_8
C_10
O_11
O_12

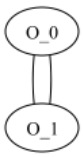

H_0
H_1
H_2
C_8
H_3
H_4
C_9
H_5
H_6
C_10
H_7
O_11
O_12


H_0
H_1
H_2
H_3
H_4
C_8
H_5
C_9
H_6
C_10
H_7
O_11
O_12

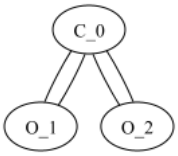

H_0
H_1
H_2
H_3
C_8
H_4
H_5
C_9
H_6
C_10
H_7
O_11
O_12

1. Exemples de graphes à doubles ou triples liaisons générés par le programme :

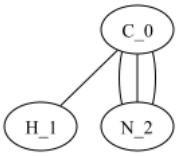
O_2 :



CO_2 :



CHN :



CH_6O_2 :