TRABAJO FIN DE GRADO

INGENIERÍA INFORMÁTICA

Detección de errores en bases de datos químicas

Autor
Jesús Navarro Merino

Directora
Rocío Celeste Romero Zaliz

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, julio de 2023

Detección de errores en bases de datos químicas

Jesús Navarro Merino

Palabras clave: palabra_clave1, palabra_clave2, palabra_clave3, ...

Resumen

Poner aquí el resumen.

Project Title: Project Subtitle

First name, Family name (student)

Keywords: Keyword1, Keyword2, Keyword3, ...

Abstract

Write here the abstract in English.

Yo, **Jesús Navarro Merino**, alumno de la titulación Grado en Ingeniría Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 15429457E, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Jesús Navarro Merino

Granada a X de MES de 201 .

Dña. **Rocío Celeste Romero Zaliz**, Profesora del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Detección de errores en bases de datos químicas***, ha sido realizado bajo su supervisión por **Jesús Navarro Merino**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a X de mes de 201 .

La directora:

Rocío Celeste Romero Zaliz

Agradecimientos

Poner aquí agradecimientos...

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La Química estudia la composición y estructura de la materia, sus propiedades y transformaciones. Estudia las sustancias, la energía y sus cambios durante las reacciones. Desde que se tienen registros, la química ha sido fundamental para el desarrollo de la humanidad, ya que ha permitido la producción de materiales, alimentos, medicamentos y energía, entre otros. Esto ha sido un proceso lento y exhaustivo a través de la experimentación. Por ejemplo, en 1881, Beilstein publica su Manual de Química Orgánica, que recogía 15000 compuestos orgánicos con sus propiedades [4]. Conforme la química se iba expandiendo, también lo hacía el volumen de datos que se generaban, siendo cada vez mas frecuentes preguntas como "¿alguien habrá sintetizado ya este compuesto?" [27]

Eventualmente, hace unas cuantas décadas, se pensó que la cantidad de información que cada químico por su cuenta había acumulado, se podía compartir y hacer accesible a la comunidad científica a través de su almacenamiento en bases de datos [12]. Con el desarrollo de técnicas de manipulación y tratamiento de esos datos surgió el término *chemoinformatics* (en español, quimioinformática).

Las *chemoinformatics* han cobrado gran importancia en los últimos años debido al aumento exponencial de datos experimentales generados en la investigación biomédica y química, y a la necesidad de manejar y analizar esta información de manera eficiente. Esta disciplina ha sido influenciada por diversas áreas, como la química, matemáticas, estadística, biología y ciencias de la computación entre otras. Al parecer, su origen se remonta a la década de 1940, habiendo ya algunas investigaciones en el área, pero el término 'chemoinformatics' se lleva utilizado desde 1998 [14]. Como tal, aun no hay un acuerdo en cuanto a su definición, seguramente por su carácter interdisciplinar, ni si quiera en cómo deletrearlo, pudiendo aparecer también como *cheminformatics*, *chemical informatics*, *chemi-informatics*, y *molecular informatics* entre otras [14, 18]. En la literatura se discuten varias

interpretaciones sobre su definición, unas más precisas y otras más generales: [14, 17, 18, 12]

La mezcla de recursos de información para transformar datos en información, y la información en conocimiento, con el fin de tomar decisiones más rápidas y efectivas en la identificación y optimización de fármacos [Brown 1998]

Chem(o)informatics es un término genérico que abarca el diseño, la creación, la organización, la gestión, la recuperación, el análisis, la difusión, la visualización y el uso de la información química. [G. Paris 1999]

La aplicación de métodos informáticos para resolver problemas de química [J. Gasteiger and T. Engel 2006]

A pesar de ello, son a día de hoy un componente esencial en el descubrimiento de sustancias químicas; sin duda es un área en constante evolución y su importancia solo aumentará en los próximos años, tanto en el descubrimiento de fármacos —que es como originariamente surgió y donde más impacto tiene en la sociedad— como en otros campos de la química.

En este ámbito son de vital importancia los sistemas de representación lineal. Algunos formatos que usan este tipo de representación son Wiswesser Line Notation (WLN), Sybyl Line Notation (SLN), Representation of structure diagram arranged linearly (ROSDAL), Simplified Molecular-Input Line-Entry System (SMILES) o IUPAC Chemical Identifier (InChI). Surgieron a medida que la química y la tecnología computacional avanzaban, y nos permiten codificar moléculas para su análisis y almacenamiento en bases de datos. En el siglo XIX, se desarrollaron varias formas de representación visual de moléculas, como las fórmulas estructurales que permitieron a los químicos dibujar y visualizar moléculas de manera más efectiva. Sin embargo, estas formas de representación no son adecuadas para su uso en la computación, ya que no son fácilmente legibles para los programas informáticos. Nosotros, los humanos, cuando vemos una estructura molecular dibujada la entendemos directamente, obtenemos una visión global de los símbolos que representan los enlaces y la distribución espacial de los átomos que la componen, pero los computadores no tienen esa facilidad. Por ello, se desarrollaron sistemas de notación lineal que permitían describir de manera más precisa y eficiente la estructura molecular, trabajando con tipos de datos sencillos, usando por ejemplo cadenas de caracteres.

Por otra parte, tenemos las herramientas, toolkits o paquetes de software que nos permiten trabajar con las chemoinformatics. Algunos de ellos son RDKit, OpenBabel, Indigo, CDK, Cactvs, OECHEM, o ChemDraw; siendo los más usados RDKit y OpenBabel (se hablará más en profundidad sobre estos dos paquetes en la Sección 2.3).

1.1. Motivación

Los formatos de representación lineal llevan siendo un tema de interés e investigación para los científicos desde mediados del siglo XIX, evolucionando poco a poco y desarrollándose nuevas representaciones en función de las necesidades —principalmente computacionales— del momento y las limitaciones que se iban descubriendo [2]. En la actualidad las más usadas son SMILES, InChI y SELFIES [41]. Como comenté antes, una forma muy potente de representar moléculas y compuestos químicos es mediante cadenas de caracteres, y de esto justamente se encargan las representaciones lineales: traducir una molécula, con sus átomos, enlaces entre ellos, ciclos y otras propiedades, en una cadena de caracteres que la represente, y que la máquina y los propios químicos puedan entender. Sin embargo, hay diferencias notables entre las representaciones, tanto en la sintaxis de las cadenas que se generan como en las aplicaciones que se le puede dar a cada una de ellas.

SMILES, ideada por David Weininger, sale a la luz en 1988 satisfaciendo con creces las necesidades de procesamiento de información química que había, desbancando a la representación estandarizada del momento, Wiswesser Line Notation (WLN). Desde ese entonces SMILES se convirtió —y sigue siendo a día de hoy— en el estándar de representación lineal, ya que permite describir estructuras moleculares de una forma sencilla en un formato fácil de leer, lo que ha hecho que sea una herramienta popular en la química computacional, siendo la más usada entre investigadores y químicos. Pese a esto, SMILES tiene dos grandes inconvenientes: una misma molécula puede escribirse con varias cadenas SMILES distintas válidas, es decir, tiene sinónimos (Figura 1.1); y no es robusto ni sintáctica ni semánticamente. En este sentido se podría generar un string que no represente una molécula válida, como lo es *CC(CCCC*, el cual tiene un paréntesis sin cerrar (lo que implica que no se delimita cuándo acaba la rama). O generar una molécula que no sea químicamente viable como *CO=CC*, que muestra un átomo de oxígeno neutro formando tres enlaces (superando el límite de enlaces covalentes que un oxígeno neutro puede tener) [41].

Esto tiene especial relevancia en el ámbito del Machine Learning (ML) o Aprendizaje Automático. Aunque se sale del alcance de este trabajo, uno de los grandes objetivos de la química computacional es la creación o diseño de nuevas moléculas. Se podrían crear modelos de ML, en particular de redes neuronales, capaces de generar moléculas ficticias válidas, para posteriormente ver sus propiedades, valorarlas energéticamente para ver cuán estables son y estudiar su viabilidad en distintas aplicaciones, entre otras cosas. SMILES dificulta esta tarea, y por ello, aparece en 2020, SELFIES (SELFreferencIng Embedded Strings), una nueva representación lineal 100 % robusta, muy usada actualmente para modelos generativos (ver [41, 35] para

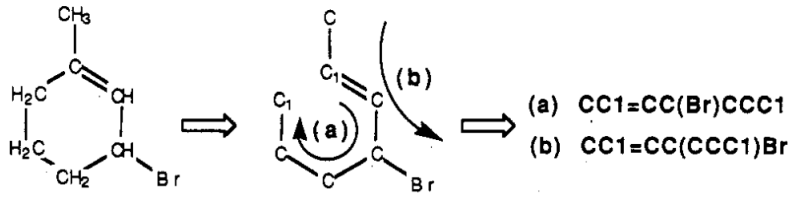


Figura 1.1: Distintas cadenas SMILES válidas para el 1-methyl-3-bromo-ciclohexeno. **(a)** Considera el ciclo como la rama principal y el bromo como ramificación. **(b)** Hace el recorrido que marca la flecha, dejando parte del ciclo como una ramificación. Imagen extraída de [6].

más detalles de cómo soluciona los problemas de robustez y otras características de la representación). SELFIES es relativamente reciente por lo que aun no termina de instaurarse entre la comunidad investigadora, pese a esto, está continuamente ampliando sus funcionalidades, mejorando en simplicidad y facilidad de uso [42]. Por último, InChI es creado en 2013 por la IUPAC (International Union of Pure and Applied Chemistry) como un proyecto para estandarizar el proceso de búsqueda de estructuras moleculares entre distintas bases de datos. Esto es porque InChI (International Chemical Identifier) genera una cadena canónica única para cada molécula, de manera que cada molécula tiene una sola representación, y dicha representación solamente hace referencia a esa molécula. La principal desventaja radica en su sintaxis y su estructura jerárquica, haciéndola complicada de leer y utilizar por los humanos. Por esto mismo, no es la mejor opción para usar en modelos generativos, pues tiene una serie de reglas y normas gramaticales y aritméticas que son complejas de aplicar al generar moléculas a través de modelos de ML[26]. Por lo anteriormente comentado, por su uso tan extendido, por su facilidad de uso y legibilidad frente a las demás representaciones, me centraré en la notación SMILES durante el desarrollo de este trabajo.

Desde la Universidad de Granada, la tutora de este TFG colabora con el grupo de investigación del ICIQ (Instituto Catalán de Investigación Química) liderado por la investigadora Mónica H. Pérez-Temprano[1]. Su foco de investigación gira en torno al entendimiento de transformaciones catalíticas en las que participan compuestos organometálicos, descubriendo y diseñando reacciones más eficientes basadas en catalizadores metálicos (para más detalle sobre el grupo de investigación y sus ámbitos de trabajo, ver su sitio web [49]). En resumen, intentan desarrollar enfoques más sostenibles para la síntesis de moléculas orgánicas usando la química organometálica. Como tal, necesitan codificar correctamente una molécula de organometálica en pos de trabajar con ella adecuadamente y utilizar todas las herramientas, para, entre otras cosas, poder dibujarla y entenderla mejor.

[1]de aquí en adelante se hará referencia a esta colaboración como “los expertos”

Los químicos de este centro comentan que uno de los principales problemas que se detectan en este ámbito es la heterogeneidad en las distintas bases de datos para un mismo compuesto o molécula. Dicho esto, existen diversas bases de datos en química donde se recoge gran cantidad de información acerca de los compuestos. Entiéndase esto como una colección estructurada y organizada que contiene datos sobre compuestos químicos, sus propiedades y relaciones con otros compuestos. Se utilizan para almacenar y recuperar información sobre moléculas, sustancias, reacciones, propiedades fisicoquímicas, e incluso literatura científica relacionada. Mencionaré ahora las más importantes y las que serán objeto de interés:

- *SciFinder*, una herramienta de investigación muy potente que permite explorar las bases de datos de CAS (American Chemical Society) las cuales contienen literatura sobre Química y otras disciplinas afines como Física, Biomedicina, Geología, Ingeniería Química, etc. Incluye referencias bibliográficas y resúmenes de artículos, informes, y libros entre otras cosas. Permite realizar búsquedas por estructura, nombres de sustancias o identificadores, reacciones en la que participa dicha sustancia, artículos y publicaciones que nombren el compuesto en cuestión, e incluso proveedores de compra [57]. Para el uso de esta herramienta es necesario acceder mediante la red de una institución autorizada (en este caso trabajo mediante la VPN de la UGR) y seguir los pasos para registrarte[2].

- *Sigma-Aldrich*, una compañía de ciencia, química y biotecnología que se dedica a la producción y venta de productos químicos, reactivos, equipos y materiales de laboratorio. Ofrece herramientas, servicios, artículos y una gran variedad de productos químicos que se utilizan en investigación, biofarmacéutica, e industria entre otros ámbitos [60]. A través de su página web se enfocan al comercio electrónico pudiendo buscar y comprar productos, compuestos orgánicos e inorgánicos, agentes reactivos, isótopos para síntesis químicas, proteínas, enzimas, etc. De cada producto muestra información relevante como la ficha de datos de seguridad, detalles de las propiedades físicas y químicas así como algunas representaciones lineales del compuesto y la representación molecular en 2D.

- *PubChem*, una base de datos abierta que sirve información a millones de usuarios en todo el mundo, desde investigadores y estudiantes hasta el público general. Recogen para cada compuesto, información sobre su estructura, representaciones 2D y 3D, identificadores, propiedades químicas y físicas, patentes, avisos de toxicidad, etc. [54]

[2]Pasos para el registro en SciFinder`https://bibliotecaugr.libguides.com/scifinder_scholar`

Para ilustrar el problema de la heterogeneidad entre bases de datos, presento las Tablas 1.1 y 1.2. Ambas tablas comparan las mismas moléculas, mostrando el código SMILES y la representación visual que ofrecen las bases de datos Sigma-Aldrich y SciFinder respectivamente. Vemos diferencias claras en el tratamiento de los ciclos aromáticos, la especificación de las cargas de los átomos y la posición de algunas ramificaciones. Utilizo un subconjunto de 5 moléculas pertenecientes a la organometálica, seleccionadas desde un conjunto de datos de 30 moléculas considerados de interés por los químicos del ICIQ (disponible para su consulta en GitHub). En el Apéndice **??**, se puede consultar una tabla comparativa con el set de moléculas al completo.

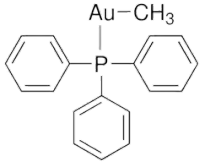
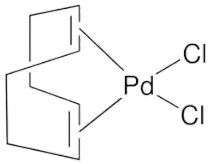
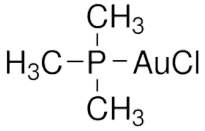
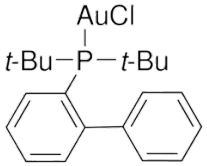
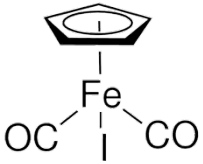
Código SMILES	Representación 2D
C[Au].c1ccc(cc1)P(c2ccccc2)c3ccccc3	
Cl[Pd]Cl.C1CC=CCCC=C1	
Cl[Au].CP(C)C	
Cl[Au].CC(C)(C)P(c1ccccc1-c2ccccc2)C(C)(C)C	
[Fe]I.[C-]#[O+].[C-]#[O+].[CH]1[CH][CH][CH][CH]1	

Tabla 1.1: Códigos SMILES y sus representaciones visuales según Sigma-Aldrich

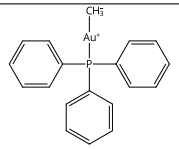
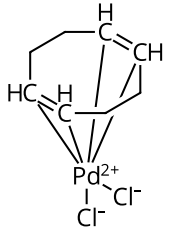
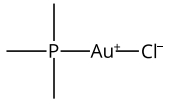
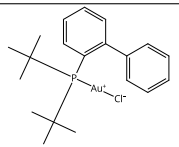
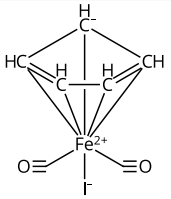
Código SMILES	Representación 2D
[Au+]([CH3-])[P](C=1C=CC=CC1)(C=2C=CC=CC2)C=3C=CC=CC3	
[Cl-][Pd+2]123([Cl-])[CH]=4CC[CH]3=[CH]2CC[CH]41	
[Cl-][Au+][P](C)(C)C	
[Cl-][Au+][P](C=1C=CC=CC1C=2C=CC=CC2)(C(C)(C)C)C(C)(C)C	
O#C[Fe+2]1234([I-])(C#O)[CH]=5[CH]4=[CH]3[CH-]2[CH]51	

Tabla 1.2: Códigos SMILES y sus representaciones visuales según SciFinder

1.2. Objetivos

El objetivo principal de este Trabajo Fin de Grado es mejorar las herramientas existentes para trabajar con chemoinformatics, adaptándolas a moléculas organometálicas y poder suplir los errores encontrados en las bases de datos disponibles. Para ello, se establecen los siguientes subobjetivos:

- Analizar las distintas bases de datos químicas disponibles y evaluar similitudes y diferencias en el almacenado de moléculas organometálicas.
- Diseñar una metodología para representar de forma canónica una molécula organometálica. Esto es, una representación única y estandarizada para una molécula dada, independientemente de la forma en que se haya representado inicialmente.
- Mejorar la visualización de moléculas organometálicas.

1.3. Estructura de la memoria

REVISAR ESTO ACRODE CON LOS CAMBIOS

- **Capítulo 1. Introducción**: se presenta el proyecto indicando la problemática a tratar, los motivos por los que se ha desarrollado y sus objetivos.
- **Capítulo 2. Estado del arte y fundamentos teóricos**: breve revisión bibliográfica del tema del proyecto, estado de las soluciones actuales y conceptos teóricos necesarios para comprender el trabajo.
- **Capítulo 3. Gestión del proyecto y planificación**: descripción de la metodología de desarrollo seguida y planificación temporal. Se detalla también la gestión de recursos, el desglose del presupuesto y se realiza un análisis de riesgos del proyecto.
- **Capítulo 4. Diseño**: se describen las clases y métodos creados, dando una breve explicación de cada una de ellos.
- **Capítulo 5. Implementación y resultados**: definición del sistema canónico alcanzado junto con sus reglas, y las mejoras en el dibujado obtenidas juntos con los resultados.
- **Capítulo 6. Conclusiones y trabajos futuros**: exposición de las conclusiones del proyecto, y posibles ampliaciones para el futuro.

Capítulo 2

Estado del arte y fundamentos teóricos

En este capítulo analizaré el estado actual del tema a tratar, junto con una pequeña revisión bibliográfica y algunos conceptos teóricos necesarios. También profundizaré un poco más en las representaciones lineales comentadas en el Capítulo 1, así como los distintos paquetes software existentes y sus limitaciones.

2.1. Revisión bibliográfica

Para el estudio de los trabajos relacionados y búsqueda de bibliografía se han consultado fuentes como IEEE Xplore, ACS Publications, o Journal of Chemical Information and Computer Sciences, entre otras. En la Figura 2.1 se exponen los resultados de un breve estudio bibliográfico sobre la literatura existente de los temas del proyecto.

Todo comienza en 1988 con la publicación de David Weininger [6], presentando SMILES como un nuevo formato de representación lineal, a partir de ahí y hasta día de hoy, fueron aumentando las publicaciones alrededor de SMILES. Sin embargo, vemos que apenas existe literatura para temas más específicos dentro de este área, como la canonización de las cadenas SMILES o la organometálica (ver Secciones 2.2.2 y 2.2.3). Además, si nos paramos a revisar las publicaciones existentes vemos que apenas ninguna coincide con los objetivos de este proyecto, y mucho menos hacen alguna propuesta de cómo darles solución. Los datos de las publicaciones se han recopilado a través de *Scopus*[1,2] con las siguientes consultas, donde CHEM y COMP

[1] `https://www.elsevier.com/es-es/solutions/scopus`

[2] `https://biblioteca.ugr.es/investigacion/herramientas-apoyo/evaluacion-publicaciones/scopus`

hacen referencia a chemistry y computer science respectivamente:

- *(SUBJAREA(CHEM) AND TITLE-ABS-KEY(smiles))*
- *((SUBJAREA(CHEM) OR SUBJAREA(COMP)) AND TITLE-ABS-KEY(smiles AND canonical))*
- *(SUBJAREA(CHEM) AND TITLE-ABS-KEY(smiles AND organometallic))*

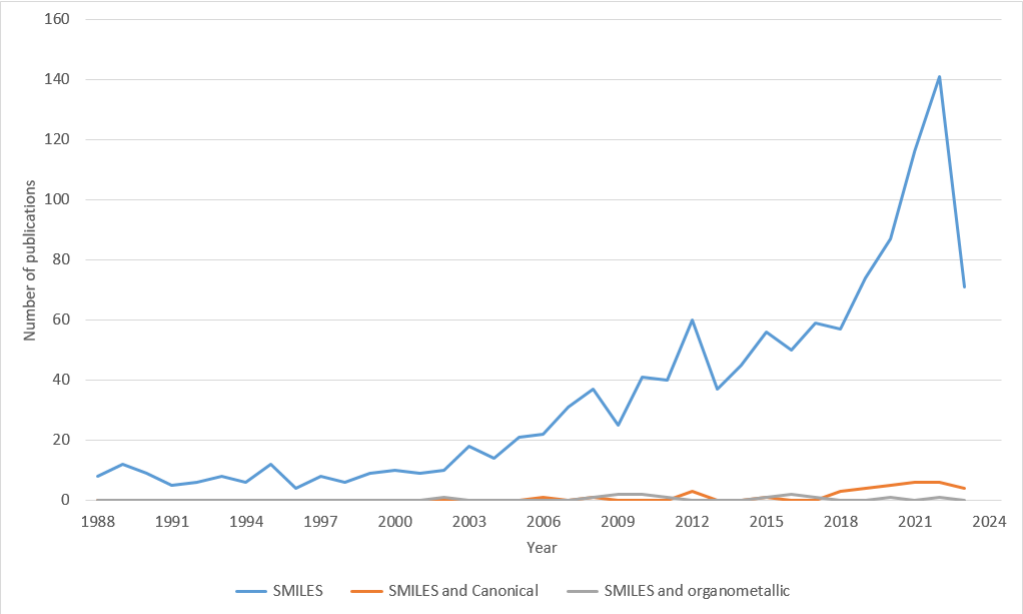


Figura 2.1: Comparativa del número de artículos publicados por año según las palabras clave de la búsqueda. Imagen de elaboración propia. Datos extraídos de *Scopus*.

2.2. Fundamentos teóricos

2.2.1. Moléculas y sus representaciones

Existen numerosas formas de representar una molécula. La más usada es la representación del grafo molecular, siendo capaces de ver rápidamente las conexiones entre átomos. El grafo molecular es un tipo de grafo no dirigido en el que los nodos están etiquetados y las aristas ponderadas. Los nodos se etiquetan según el tipo de átomo que representen, utilizando su símbolo químico: si es un carbono (C), oxígeno (O), nitrógeno (N), cloro (Cl), etc; mientras que a las aristas se le atribuyen pesos según su tipo de enlace: simple, doble, triple o aromático. El concepto de aromaticidad es

particularmente importante en química. El compuesto aromático por excelencia es el benceno, conformado por una serie de electrones deslocalizados de tal manera que no se pueden describir sus enlaces con enlaces simples o dobles, sino con una alternancia de ambos o directamente usando la forma resonante (visualizada con una curva, Figura 2.2).

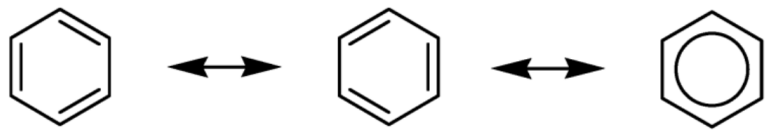
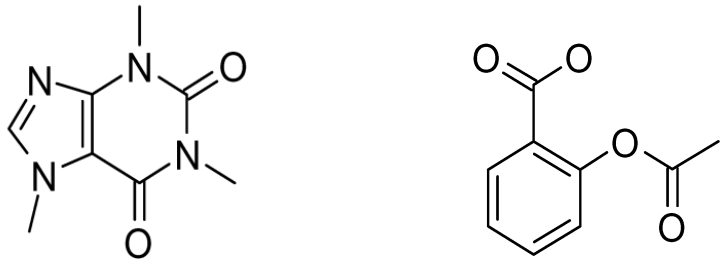


Figura 2.2: Alternativas en la representación del benceno de acuerdo a su aromaticidad. Anillo de 6 carbonos con electrones deslocalizados. Imagen extraída parcialmente de [16].

Hay varias opciones de visualización. Los carbonos por ejemplo, por lo general únicamente se representan si son terminales (último átomo de la cadena), aunque podrían representarse también los carbonos en mitad de una cadena o no hacerlo aun siendo terminales. Los hidrógenos tampoco se representan ya que la mayoría de ellos son implícitos, es decir, se usan para llenar las valencias sin usar de cada uno de los átomos de la molécula. Según la Teoría de Enlaces de Valencia, a cada átomo se le asigna una valencia típica (la más frecuente, sin tener en cuenta los estados de oxidación), así por ejemplo: el carbono tiene una valencia de 4, el oxígeno de 2, el nitrógeno de 3, etc [24, 18]. Se muestra en la Figura 2.3 dos ejemplos de moléculas representadas por su grafo molecular.



(a) Cafeína (b) Aspirina

Figura 2.3: Grafo molecular sin hidrógenos de **(a)** la cafeína y **(b)** la aspirina. Imágenes extraídas de [18].

Como representación alternativa al grafo molecular tenemos la tabla de conexiones —que es en realidad una representación matricial del grafo molecular—, muy utilizada por los formatos de archivo más comunes para almacenar moléculas, SDF (Structural Data File), MOL2 o CML (Chemical

Markup Languaje) entre otros. La tabla de conexiones separa la información de los átomos y los enlaces en dos bloques dentro del fichero. Por último, tenemos del formato de fichero '*.smi*', en donde se pueden almacenar 1 o múltiples cadenas SMILES. representado cada línea del fichero una cadena SMILES distinta.

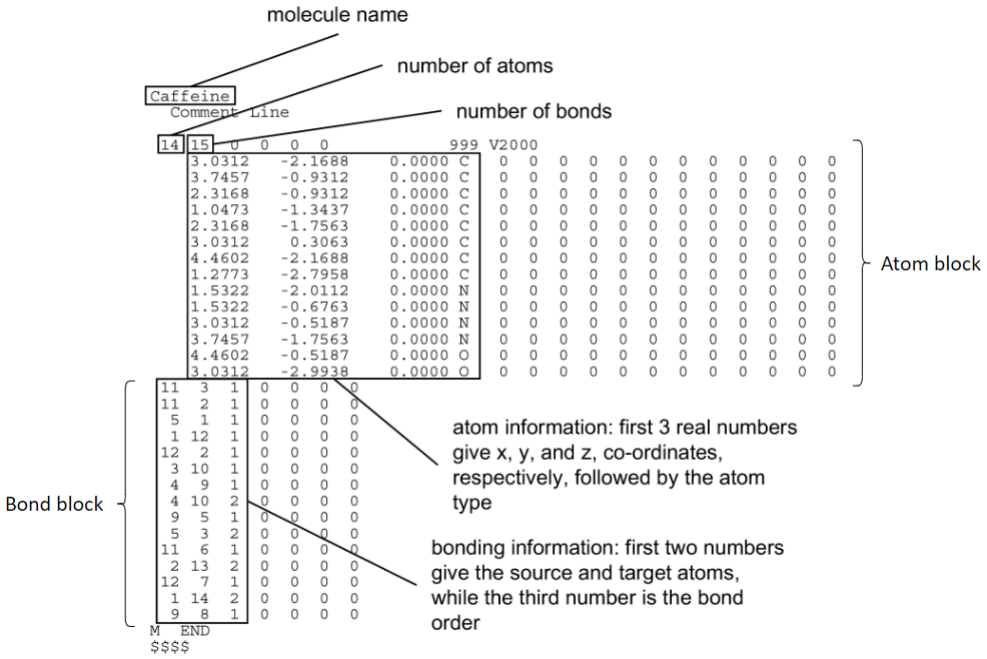


Figura 2.4: Tabla de conexiones para la cafeína en un fichero SDF. Imagen extraída de [18].

2.2.2. Representaciones lineales

En la Sección 1.1 ya se expone la necesidad de las representaciones lineales, y se nombran 3 principalmente: SMILES, InChI, y SELFIES. Para entenderlas mejor, se pasa a describir la sintaxis de cada una de ellas.

SMILES, Simplified Molecular-Input Line-Entry System

SMILES representa una molécula mediante una cadena de caracteres con sus átomos. Las reglas para describir una cadena SMILES son las siguientes:

- Los *átomos* se representan por su símbolo atómico, normalmente en mayúsculas.

- Los *enlaces* se especifican mediante los caracteres "−", "=" y "#" para enlaces simples, dobles y triples respectivamente, aunque los simples no se suelen representar al ser implícitos.
- Las *ramificaciones* se especifican entre paréntesis, pudiendo ser anidados.
- Los *ciclos* se representan mediante un número común que aparece tras los átomos de apertura y cierre del ciclo, indicando los átomos que se conectan entre sí (como si se rompiera el ciclo y los números lo volvieran a unir).
- Las *estructuras aromáticas* se especifican escribiendo los átomos que la componen en minúscula. Por ejemplo, el benceno 'c1ccccc1'. Se suele preferir esta forma aromática sobre la convencional con dobles y simples enlaces 'C1=CC=CC=C1'.

Aquí se han recogido los aspectos básicos sobre la sintaxis de SMILES. Para más información acerca de la especificación de SMILES, ver [61, 6]. Igualmente, se hablará más sobre SMILES y se verán numerosos ejemplos durante el resto del documento.

InChI, International Chemical Identifier

Un identificador InChI es una cadena de símbolos legible por máquina que permite a un ordenador representar el compuesto de forma totalmente inequívoca. Además, la estructura original puede recuperarse a partir de su InChI a través del software adecuado. Como vemos en la Figura 2.5, no es directamente inteligible para el lector humano. InChI representa las diversas características de una estructura química de forma jerárquica, por capas [26, 21]. Cada una de las capas están separadas por '/', y dependiendo de la molécula aparecerán unas capas u otras; cada capa puede tener varias subcapas dedicadas a una propiedad determinada de la molécula (consultar [26, 21]). Las capas son las siguientes:

- Versión de InChI
- Capa principal
  - Fórmula molecular
  - Conectividades entre átomos
- Especificación de cargas
- Información sobre estereoquímica

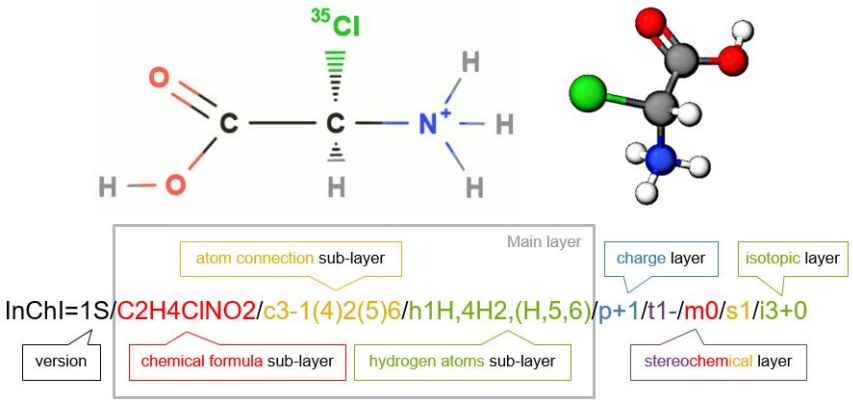


Figura 2.5: Representación del grafo molecular y modelo 3D, junto con su InChI para el *[(R)-carboxy(chloro)methyl]azanium*, obtenida a partir del *2-*(^{35}Cl*)chloro-R-glycine*. Imagen extraída de [30].

- Información sobre isótopos

Aunque InChI es una cadena única que identifica una estructura química, su longitud aumenta con el tamaño de la estructura a representar. Una estructura con más de 100 átomos da como resultado un InChI muy largo. Se vio la necesidad por tanto de desarrollar una versión compacta de InChI para que los motores de búsqueda de las bases de datos fueran efectivos. Esto es InChIKey, una cadena con una longitud fija de 27 caracteres, calculada en base a algoritmos de hashing. El problema de InChIKey es que se pierde la capacidad de recuperar la estructura a la que pertenece, así que para saber la molécula de un determinado InChIKey es necesario hacer una búsqueda en las bases de datos.

SELFIES, SELF-referencIng Embedded Strings

SELFIES es una representación basada en una secuencia de tokens con restriciones semánticas. SELFIES en sí mismo es una gramática formal de Chomsky de tipo 2, reforzada con funciones recursivas autorreferenciadas asegurar la generación de grafos sintáctica y semánticamente válidos [50]. La gramática de SELFIES está diseñada específicamente con el objetivo de eliminar moléculas sintáctica y semánticamente inválidas para su uso en tareas generativas. La secuencia va formándose en varios pasos o estados, utilizando unas reglas de derivación (consultar las reglas en [59, 50]). Lo más importante a saber para entender una secuencia SELFIES son las siguientes pautas:

- Cualquier token de la cadena debe ir entre corchetes.

- Los átomos se especifican con su símbolo atómico, precedido si es necesario del tipo de enlace ('=', '#').
- Para la especificación de ramas y ciclos, en lugar de tener un número de apertura y cierre como SMILES, solamente se usa un token en una posición, y el token siguiente define la longitud de la rama o ciclo. Tanto ramas como ciclos siempre empiezan con '[Branch1]' o '[$< B >$Ring1]' respectivamente, y los tokens que indican la longitud se muestran en la Tabla 2.1.
  - En el caso de las ramas, el token de longitud indica que los $(Q+1)$ tokens siguientes son los que forman parte de la rama (incluyendo otros tokens de rama o ciclo).
  - En el caso de los ciclos, el token de longitud indica que el átomo actual se conecta con un enlace de tipo $< B >$ (' ', '=' o '#') al átomo $(Q + 1)$ posiciones previas (excluyendo tokens de ramas o ciclos).

Indice	Token	Indice	Token
0	[C]	8	[#Branch2]
1	[Ring1]	9	[O]
2	[Ring2]	10	[N]
3	[Branch1]	11	[=N]
4	[=Branch1]	12	[=C]
5	[#Branch1]	13	[#C]
6	[Branch2]	14	[S]
7	[=Branch2]	15	[P]
Cualquier otro token implica índice 0			

Tabla 2.1: Lista de símbolos SELFIES con las longitudes que representan cuando aparecen tras una rama o ciclo. Está basado en un sistema hexadecimal y números mayores se pueden mapear si se utilizan los siguientes n símbolos. Tabla elaborada en base a [41].

En la Figura 2.6 vemos un ejemplo de la derivación de SELFIES.

2.2.3. Organometálica

La organometálica es una rama de la química que se dedica al estudio de compuestos organometálicos, es decir, aquellos que contienen enlaces metal-carbono. Este tipo de compuestos normalmente se clasifican dentro de la química de coordinación, pero con algunas diferencias. Los compuestos de coordinación tienen un metal coordinado (ligado) a cualquier cosa, por lo general 'restos' de alguna reacción previa y que no necesariamente deben ser

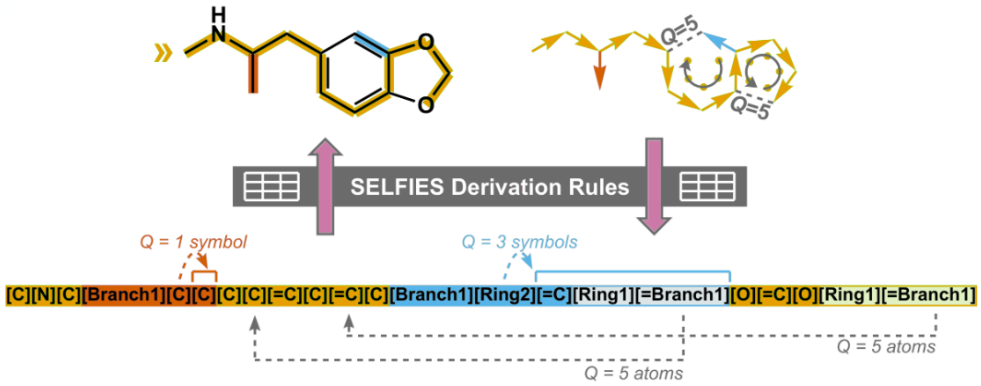
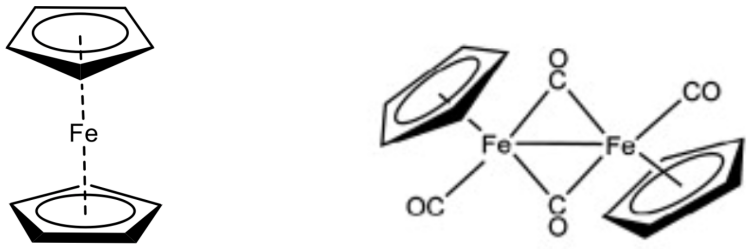


Figura 2.6: SELFIES para la estructura molecular de *3,4-methylenedioxy methamphetamine* (MDMA). Imagen extraída de [41].

orgánicos. Por tanto, no todos los compuestos de coordinación son compuestos organometálicos, pero sí podemos considerar todos los organometálicos como de coordinación. En una reacción por ejemplo, lo importante es que exista una unión metal-carbono en algún punto para considerarla dentro de la organometálica. Cuando hablamos de 'metales', se están considerando los metales de transición, ya que al tener orbitales 'd' disponibles son muy propensos a este tipo de enlaces con ligandos. Estos abarcan desde el grupo 3 hasta el 12 en la tabla periódica, del escandio (Sc) al zinc (Zn) a lo largo de los periodos 4 al 7[3].

Dentro de los compuestos organometálicos, los metalocenos son muy comunes y reconocibles. Estos se caracterizan por un catión metálico central, usualmente de un metal de transición, y dos ligandos orgánicos llamados ciclopentadienilos (Cp[4]). Los ligandos Cp están formados por un anillo de cinco átomos de carbono unidos mediante enlaces covalentes. Estos ligandos Cp se unen al metal central de manera que cada átomo de carbono en el anillo Cp contribuye con un par de electrones pi para formar enlaces con el metal. Debido a la presencia de los enlaces pi, los metalocenos exhiben una gran estabilidad y reactividad química característica, lo que ha llevado a su amplio uso en catálisis y síntesis orgánica. Esta clase de enlaces se denomina *hapticidad*, en donde un grupo de átomos contiguos de un ligando se coordinan a un átomo de metal central. Se utiliza el símbolo griego 'eta', η con el número de átomos contiguos que se coordinan como superíndice. Para el ferroceno mostrado en la Figura 2.7, serían 2 ligandos de η^5-C_5H_5 (para el lector interesado, revisar la literatura sobre química organometálica [9, 10, 38, 65, 23]). En la siguiente Figura 2.7 podemos ver algunos ejemplos:

[3]explicaciones dadas por los expertos
[4]en adelante se hará referencia como Cp



(a) (b)

Figura 2.7: Representaciones 2D. **(a)** compuesto de ferroceno, que consta de un átomo de hierro central enlazado con 2 Cp; **(b)** derivativo del Dicarbonylcyclopentadienyliron. Imágenes extraídas de [39, 15] respectivamente.

2.3. Herramientas o toolkits

Existe una gran variedad de herramientas a la hora de trabajar dentro de la química computacional, tanto open-source como aplicaciones propietarias para las que son necesarias pagar una licencia de uso. La química computacional según comenté en el Capítulo 1, se aplica en muchos ámbitos de la ciencia. Como tal, hay herramientas con propósitos muy distintos [63]:

- Extensiones de navegador que mejoran el acceso a la información de las bases de datos [64].
- Cálculos de propiedades fisicoquímicas (casi cualquier herramienta lo permite).
- Cribado virtual de moléculas como *ChemAxon*, *MOE*, *LigandScout* o *Forge*.
- Modelado y bocetado de moléculas como *Marvin*, *ChemDraw* o *ChemDoodle*.
- Hojas de cálculo con análisis de datos químicos como *Vortex*.
- Toolkits de propósito general con diversas funcionalidades básicas como *OpenBabel* o *RDKit*.

Las herramientas más complejas o relacionadas de alguna manera con la medicina suelen ser de pago. Para los objetivos de este proyecto se han tenido en cuenta los toolkits de propósito general mencionados anteriormente que son capaces de trabajar con SMILES.

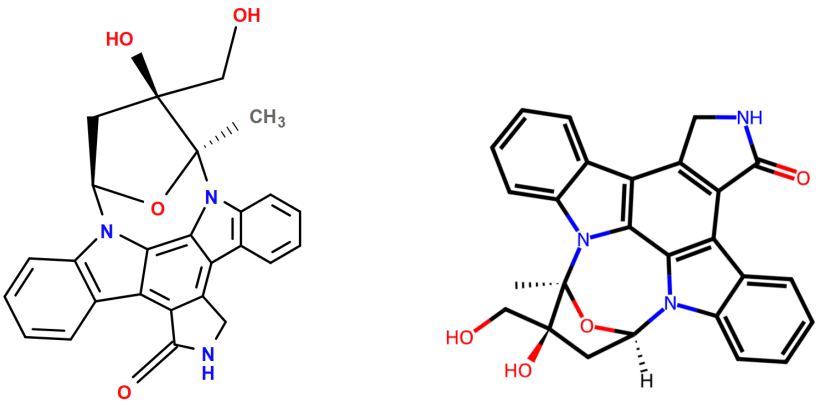
2.3.1. OpenBabel y RDKit

Openbabel y RDKit son bastante parecidos en cuanto a sus funcionalidades, ambos permiten la manipulación de estructuras químicas, cálculos de propiedades moleculares, análisis de similitudes entre moléculas, generación de representaciones 2D y conversión entre distintos tipos de ficheros. Para elegir entre una de estas 2 herramientas, se han hecho pruebas iniciales en un notebook de Google Colab [48] (disponible el en GitHub de este proyecto).

RDKit por su parte, consigue representaciones 2D parecidas o mejores que las de OpenBabel para moléculas pequeñas y cuenta con mayores opciones de personalización del dibujo resultado [55] haciéndolo más visual. Además, es más preciso a la hora de representar la estereoquímica, algo en lo que OpenBabel es bastante pobre. Vemos en la Figura 2.8 un ejemplo con ambos paquetes software. Sin embargo, si se le exige un poco más a RDKit pidiéndole moléculas complejas no funciona del todo bien. Concretamente, cuando introducimos moléculas de organometálica no las lee correctamente aun siendo químicamente válidas. RDKit lleva a cabo un proceso de 'saneamiento' (*sanitization*) [66] por el que, además de calcular algunas propiedades útiles (pertenencia de los átomos a anillos o hibridaciones), comprueba que la molécula de entrada es 'razonable'. Para RDKit, serán razonables las moléculas que cumplan la regla del octeto [28] y puedan representarse mediante las estructuras de Lewis de manera completa [22, 28]. Como hemos visto anteriormente, los compuestos de coordinación se rigen por otro tipo de sistema de valencia, y RDKit no es capaz de trabajar con esta clase de moléculas. De hecho, del set de moléculas del que dispongo para el proyecto, no es capaz de leer ningún código SMILES de los que fueron extraídos de SciFinder. Los SMILES de SigmaAldrich si los puede usar, pero más adelante en esta misma sección se explicará porqué estos no nos sirven.

OpenBabel por otro lado, ofrece más libertad en este sentido siendo capaz de leer todos los SMILES del set del que partimos. Pero en general, algo que hacen mal ambos paquetes es representar moléculas en 2D. Para moléculas convencionales, moléculas orgánicas o inorgánicas sencillas funciona bien, pero las organometálicas les supone un reto, y dada la escasa literatura en el tema (ver Sección 2.1), no dispongo de muchas referencias de las que partir.

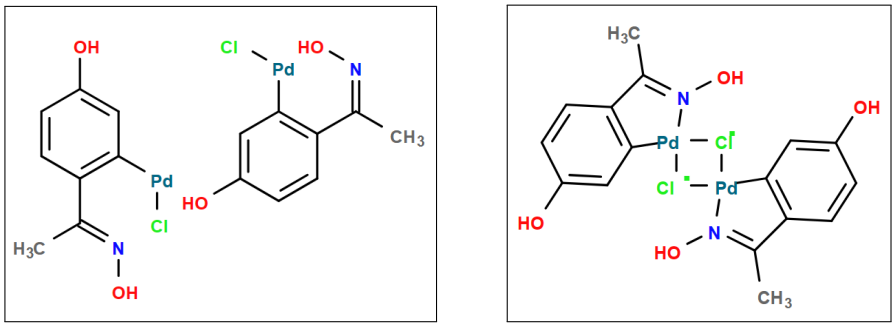
En cuanto a los datos de partida contamos con 2 versiones de SMILES, los provenientes de SigmaAldrich y los de SciFinder. Siguiendo la comparación de la Sección 1.1 ambos SMILES son notablemente distintos, de hecho no se podrían considerar ni siquiera sinónimos ya que al de SigmaAldrich le faltan enlaces, y no llegan a codificar la misma molécula. Tanto es así que las cadenas SMILES que contienen desconexiones (representadas por el punto '.') no nos son para nada útiles. Al fragmentar la molécula estamos perdiendo los enlaces entre los átomos, una información muy valiosa para la



(a) OpenBabel (b) RDKit

Figura 2.8: Representaciones 2D para el *Lestaurtinib*, medicamento en estudio para el tratamiento de las leucemias agudas y algunos otros tipos de cáncer. **(a)** ha sido generado con OpenBabel; y **(b)** ha sido generado con RDKit.

mayoría de operaciones. Como se dijo antes, una molécula se suele almacenar principalmente identificando sus átomos y los enlaces entre sus átomos. Podríamos decir que se está perdiendo casi la mitad de la información acerca de la molécula, y dado que uno mismo no puede inventarse los enlaces, no es un buen SMILES para nuestros objetivos, ni para la canonización ni para el dibujado. En la Figura 2.9 vemos la diferencia entre la representación de un SMILES inconexo frente a uno con buena conectividad.



(a) (b)

Figura 2.9: Representaciones 2D generadas con OpenBabel para el *'Nájera Catalyst II'*. **(a)** SMILES con desconexiones extraído de SigmaAldrich; y **(b)** SMILES conectado extraído de SciFinder.

2.3.2. Herramientas de bocetado

Existen herramientas tipo ChemDraw [45], que son las que los químicos e investigadores utilizan para dibujar manualmente las moléculas que luego añaden a sus publicaciones. Son este tipo de dibujos también los que probablemente podemos ver en algunas bases de datos como SigmaAldrich. En la Figura 2.10 se muestra su interfaz y algunas moléculas bocetadas.

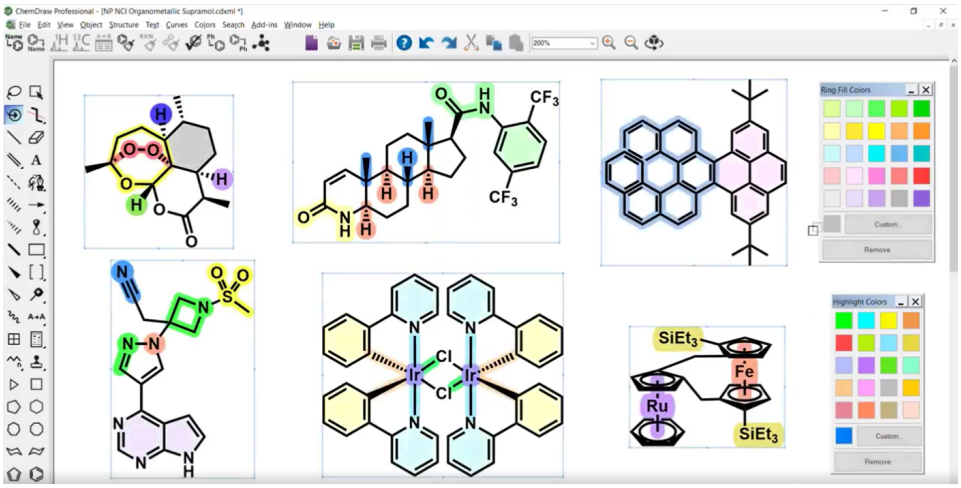


Figura 2.10: Interfaz de ChemDraw con algunas moléculas de prueba bocetadas. Imagen extraída de su página oficial [45]

2.3.3. Nomenclatura canónica

Como ya se ha visto, InChI proporciona una representación canónica para las moléculas, lo que permite una vinculación directa y unívoca entre las bases de datos. SMILES por otro lado complica este proceso. SMILES fue desarrollado en su momento como un software propietario por Daylight Chemical Information Systems (Daylight) [46], y desde su introducción a finales de los 80 se ha extendido como una norma de facto para representar estructuras moleculares. En base a esto y con el tiempo, se han escrito en varios lenguajes de programación muchos paquetes de software independientes que trabajan con SMILES[53].

Bien es cierto que hay cada vez más propuestas de cómo alcanzar una nomenclatura única [8, 20, 62, 44, 67] y ya existen algunas reglas definidas para la química orgánica a la hora de establecer prioridades entre los átomos y ordenar la molécula. Véase por ejemplo las *'Reglas de Cahn-Ingold-Prelog'* aplicadas a enantiómeros [1, 5, 51] para definir órdenes levógiros o dextrógiros (no abordaremos esto, excede el alcance del proyecto). Pero trasladar estas reglas a compuestos organometálicos es complicado, además de haber

multitud de excepciones, en la mayoría de casos un mismo fragmento está enlazado por varios sitios al metal y hay que definir el átomo inicial por el que empezar a recorrer la molécula. Al final, cada paquete software implementa esta decisión de una manera diferente usando un algoritmo propio, obteniendo así lo que cada uno de ellos nombra como 'canonical SMILES', pero siguen siendo distintos entre ellos, por lo que no es canónico.

2.3.4. Conclusiones

Dicho todo lo anterior, durante las pruebas realizadas se ha podido comprobar y extraer las siguientes conclusiones:

- Al representar la misma molécula con herramientas distintas, lo normal es que generen dibujos diferentes, en donde uno será mejor o más correcto que el otro.
- Para una misma molécula es probable que cada base de datos muestre una cadena SMILES diferente y una representación 2D distinta. De hecho, la imagen puede ni siquiera concordar con el SMILES que ofrecen, apareciendo en multitud de casos por ejemplo, un SMILES desconectado y una imagen con la molécula al completo (lo más seguro es que estén hechos a mano)
- Vistos los resultados de los SMILES de SigmaAldrich, para el resto del proyecto se asumirá que se trabaja con los SMILES de SciFinder, ya que dotan de mejor conectividad.
- Debido a lo anterior, y como RDKit no soporta los datos de entrada, se utilizará la librería de OpenBabel para el proyecto.
- Hay diversos tipos de compuestos que no se describen bien mediante la representación de su grafo molecular, como los compuestos de coordinación. Su sistema de enlaces no se ajusta a la teoría de los enlaces de valencia, por lo que es complejo describir sus enlaces mediante relaciones 1 a 1 entre los átomos [34].
- Moléculas complejas con multitud de ciclos o muchos enlaces al mismo átomo (generalmente un metal), OpenBabel genera unas representaciones 2D muy agrupadas, con los enlaces superpuestos y partes del dibujo unas encima de las otras, dificultando su entendimiento.
- Hasta el momento, no hay ningún sistema de canonizado que trabaje de manera eficiente con moléculas organometálicas.

Teniendo esto en cuenta y lo mencionado en la Sección 1.1, en las siguientes secciones se dará pie a la implementación de una propuesta de

nomenclatura canónica para moléculas organometálicas y la mejora del dibujado de las mismas con OpenBabel.

Capítulo 3

Gestión y Planificación del proyecto

3.1. Metodología

En el pasado, el desarrollo de software seguía un enfoque *ad hoc* (software a medida) y poco estructurado, lo que llevaba a problemas como retrasos, presupuestos desbordados, productos finales que no cumplían con las expectativas o proyectos inmanejables y difíciles de mantener. Era frecuente por tanto proyectos fallidos o de mala calidad. Como resultado, surgió la necesidad de establecer un marco de trabajo más formal y disciplinado para el desarrollo de software. Las metodologías de desarrollo proporcionan pues, un marco de trabajo y un conjunto de métodos que guían a los equipos de desarrollo a lo largo del ciclo de vida del proyecto. Es por tanto, a día de hoy, fundamental elegir una metodología que se adapte bien al proyecto. Por lo general, las metodologías de desarrollo se clasifican en dos grandes grupos, tradicionales y ágiles. Se resume en la Tabla 3.1, las diferencias entre ambas metodologías.

Para este proyecto hay varias razones por las cuales elegir una metodología ágil:

- Se trata de un proyecto de investigación, donde se intentarán desarrollar varios algoritmos para la resolución de un problema. Por lo que, a priori no se conoce la calidad de los resultados que se obtendrán. Se deberán realizan experimentos iniciales para detectar los problemas, desarrollar una posible solución y volver a realizar experimentos para evaluar los resultados hasta alcanzar unos que cumplan con los objetivos o sean lo suficientemente satisfactorios.

- Se hará uso de herramientas en las que no se posee experiencia pre-

	Tradicionales	Ágiles
Enfoque	Secuencial	Iterativo e incremental
Planificación	Detallada y exhaustiva	Adaptativa y flexible
Gestión cambios	Difícil de manejar	Fomenta la adaptabilidad
Requisitos	Definidos desde el inicio	Evolucionan con el tiempo
Entrega software	Al final del proyecto	Continua, en incrementos
Colaboración	Menos énfasis	Fomenta la colaboración
Equipos	Mejor en equipos grandes	Mejor en equipos pequeños
Adaptabilidad	Menor flexibilidad	Mayor flexibilidad
Retroalimentación	Al final del proyecto	Constante y temprana

Tabla 3.1: Tabla comparativa de las principales características entre las metodologías tradicionales y ágiles

via, lo que podrá originar retrasos en la planificación o modificaciones frecuentes.

- Las metodologías ágiles se ajustan mejor en equipos pequeños como indico en la Tabla 3.1. En este caso el equipo de trabajo solo tiene una persona.
- La figura del tutor y los expertos en química del ICIQ serán importantes para dar feedback continuo durante el proyecto y opinar sobre la calidad de los resultados.

Por las anteriores razones, una metodolgía clásica no se adaptaría bien al proyecto, prefiriendo una ágil. Permite una mayor flexibilidad con las tareas, y un desarrollo incremental en base a los resultados intermedios que se vayan obteniendo.

Dentro de las ágiles existen varias metodologías como Scrum, eXtreme Programming, Lean Development, Kanban, Crystal, DSDM, etc. Las más comunes hoy día son Scrum y XP, al ser las que mejores resultados obtienen en proyectos de desarrollo software [36, 32, 25, 37]. Hay que tener en cuenta que la mayoría de metodologías imponen una serie de prácticas y principios para guiar al equipo de desarrollo durante el proyecto. A algunos autores como Mike Cohn (fundador de la Scrum Alliance) les gusta tratar las metodologías como marcos de trabajo y no como una serie de métodos y reglas estrictas que haya que seguir al pie de la letra [13].

El perfil del proyecto no terminaba de encajar con ninguna metodología al completo, por lo que se ha aplicado un híbrido entre Scrum y XP, mez-

clando aspectos y técnicas de ambas según se ajustaban al proyecto. Scrum maneja la parte administrativa del proyecto, definiendo cómo se especifica el trabajo y el proceso de entrega de características, mientras se aplican algunas prácticas propias de XP para la parte de codificación[29, 58]

3.1.1. Scrum

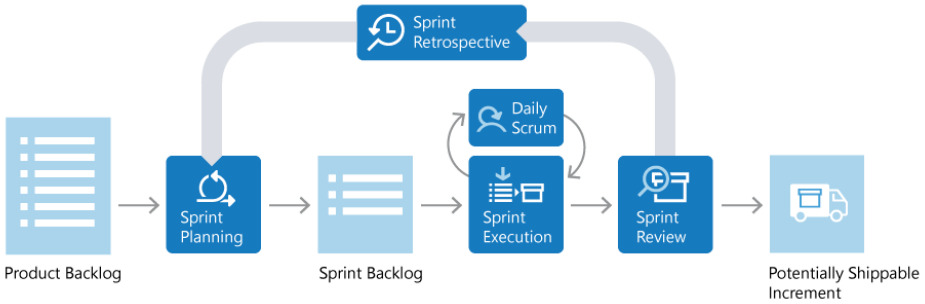


Figura 3.1: Ciclo de vida iterativo de Scrum.

El ciclo de vida de Scrum (Figura 3.1, extraída de Microsoft learn[1]) se basa en un enfoque iterativo e incremental que permite a los equipos adaptarse a los cambios y entregar productos de alta calidad en un tiempo reducido. Los principios de transparencia, inspección y adaptación son fundamentales para alcanzar los valores centrales de Scrum, la calidad, flexibilidad, mejora continua, compromiso, coraje, ritmo y responsabilidad. Una de las características principales de Scrum es su enfoque en ciclos de desarrollo cortos llamados *sprints*. Estos sprints suelen tener una duración de una a cuatro semanas, durante los cuales se planifican, desarrollan, prueban y entregan incrementos de software funcionales. Cada sprint comienza con una reunión de planificación en la que el equipo selecciona un conjunto de tareas para ser completados durante el sprint. Podemos definir Scrum según los elementos que lo componen:

- **Artefactos**
  - **Product Backlog (Pila del producto)**: es una lista ordenada y priorizada de todas las funcionalidades, características y mejoras que podrían ser necesarias para el producto. Es responsabilidad del Product Owner y se actualiza constantemente a medida que se obtiene nueva información o se generan cambios en los requisitos.
  - **Sprint Backlog (Pila del sprint)**: es un conjunto de elementos seleccionado del Product Backlog para el sprint. Se descomponen

[1] `https://www.scrum.org/`

en tareas de desarrollo más pequeñas junto con estimaciones de tiempo, expresando los requisitos en lenguaje técnico.

- **Reuniones**
  - **Sprint Planning**: marca el inicio de cada sprint y se realiza con el propósito de identificar el objetivo principal del sprint y las tareas concretas que se van a desarrollar en él. Como resultado, se genera el Sprint Backlog.
  - **Daily Meetings**: reunión diaria de unos 15 minutos donde participan los miembros del Equipo de Desarrollo y el Scrum Master. Es una manera de estar al tanto del trabajo realizado y cuáles son los siguientes pasos. Es una oportunidad de identificar rápidamente obstáculos o problemas.
  - **Sprint Review**: al final del sprint se pone en común todo el trabajo realizado durante el Sprint. Sirve para recoger información o feedback sobre el estado del proyecto.
  - **Sprint Retrospective**: el equipo dedica tiempo a reflexionar sobre los aspectos positivos y las áreas que requieren mejoras. Como resultado de la retrospectiva, se generan acciones específicas para implementar en el siguiente sprint.
- **Roles**: representan responsabilidades dentro del proyecto.
  - **Stakeholders**: son todos aquellos interesados en el proyecto, tanto personas como organizaciones (gente de marketing, comerciales, usuarios, etc).
  - **Product Owner (PO, propietario)**: debe conocer perfectamente el entorno de negocio del cliente, las necesidades y el objetivo que se persigue con el sistema que se está construyendo. Debe conocer también como funciona Scrum para desempeñar bien su rol. Su responsabilidad principal es la de crear, administrar, y priorizar el P.Backlog, así como validar o rechazar el incremento resultado de cada iteración.
  - **Scrum Master (director del proyecto)**: garantiza el correcto funcionamiento de los procesos y metodologías que se empleen en el equipo. Gestiona el proceso e intenta mejorar la productividad del equipo. Promueve los valores y prácticas de Scrum, elimina impedimentos, facilita la colaboración entre los roles, actúa como escudo ante cosas externas. Se asegura de que el PO sepa cómo ordenar la pila de producto para maximizar el valor generado en cada sprint.
  - **Equipo de desarrollo**: es el que se encarga de desarrollar el producto y hacer los entregables en incrementos. Los miembros del

equipo necesitan ser auto-organizados, multidisciplinares, multifuncionales, con un alto compromiso y sin jerarquías internas. Son los verdaderos responsables de que el producto salga adelante y se completen los incrementos. Se encargan de estimar el tamaño de los ítems del backlog. Es importante que el equipo de desarrollo comprenda bien la visión que tiene el PO acerca del producto. Suelen estar formados de entre 5 a 9 personas.

3.1.2. XP

La programación extrema (XP) es una metodología ágil que se centra en la velocidad y la simplicidad con ciclos de desarrollo muy cortos. En XP se promueven una serie de valores: comunicación, simplicidad, retroalimentación, coraje y respeto [29]. Diseñada para entornos dinámicos con requisitos cambiantes y orientado fuertemente hacia la codificación, reduciendo considerablemente la documentación. En XP, las tareas que se terminan son susceptibles de ser modificadas durante el transcurso del proyecto, incluso después de que funcionen correctamente, por lo que son importantes las siguientes prácticas.

- **Prácticas**: XP propone una serie de prácticas a nivel técnico que se deberían adoptar para el desarrollo del proyecto [47]. Las más importantes a mi parecer son:
  - **Programación a pares**: los programadores trabajan en parejas, mientras uno escribe el código, el otro proporciona comentarios y realiza revisiones en tiempo real. Esto promueve el intercambio de conocimientos, la revisión de código constante y la minimización de errores.
  - **Propiedad colectiva del código**: la propiedad colectiva anima a todos a aportar nuevas ideas sobre todos los segmentos del proyecto. Cualquier desarrollador puede cambiar cualquier línea de código para añadir funcionalidad, corregir errores, mejorar diseños o refactorizar.
  - **Estándares de codificación**: el código ajustarse a las normas de codificación acordadas. Estas hacen que el código sea consistente y fácil de leer y refactorizar para todo el equipo. Un código con el mismo aspecto o que sigue unas normas fomenta la propiedad colectiva.
  - **Marcha sostenible**: encontrar un ritmo de trabajo para el equipo de desarrollo donde todos los miembros se sientan cómodos. Las horas extra acaban con el espíritu y la motivación del equipo. A veces, menos es más.

- **Integración continua**: los equipos de XP no esperan a que se completen las iteraciones, sino que se integran constantemente. Se cuenta con un repositorio de código donde los desarrolladores envían el código cada poco tiempo.
- **Refactorización**: reescribir ciertas partes del código para aumentar su legibilidad y mantenibilidad pero sin modificar su comportamiento. Las pruebas han de garantizar que en la refactorización no se ha introducido ningún fallo. Evita la complejidad innecesaria.
- **Desarrollo orientado a pruebas**: las pruebas son frecuentemente repetidas y automatizadas cada vez que se haga un cambio, por pequeño que sea, antes de desplegar la nueva versión. Se suelen escribir antes que el propio código.

▪ **Roles**

- **Programador**: encargados de escribir y probar el código.
- **Cliente**: representa los intereses del usuario y es responsable de proporcionar las necesidades, requisitos del software y establecer prioridades.
- **Entrenador (Coach)**: es el líder del equipo, actúa como facilitador y promotor de las prácticas y valores de XP, y ayuda al equipo a mejorar y adaptarse.
- **Consultor**: miembro externo al equipo de desarrollo con conocimiento específico en un tema necesario.
- **Rastreador (Tracker)**: se encarga de gestionar la planificación y llevar un seguimiento del proyecto detectando los problemas en él.

3.1.3. Aplicación de Scrum/XP al proyecto

Algunas de las características y prácticas mencionadas de cada una de las metodologías no son aplicables al proyecto dada su naturaleza e integrantes, como por ejemplo, la programación por parejas propia de XP. Se describe ahora, el enfoque que se le ha dado del híbrido Scrum/XP al proyecto.

Se lleva a cabo una planificación por *Sprints* de entre 2 a 4 semanas, dependiendo de las tareas a realizar. Entre medias y al final de cada sprint, se agendará una *reunión con la tutora* en la que se hará retrospectiva del mismo, para comprobar el estado y avance del proyecto. Se revisará qué se ha hecho durante el sprint y cómo se ha hecho, repasando las novedades desde la última iteración y puntos a mejorar o pulir, priorizando las siguientes tareas

a realizar en base a los resultados. Igualmente, se mantendrá el contacto con el tutor de manera constante vía correo electrónico.

Mediante la *integración continua*, los cambios se envían con frecuencia a un repositorio compartido con un sistema de control de versiones. Cada vez que se realiza un cambio o se desarrolla una nueva funcionalidad, se ejecutan *pruebas* en el dataset completo de moléculas, comprobando rápidamente si los resultados son los esperados (parcial o totalmente), y detectar cualquier error antes de que se convierta en un problema más grave. Se sigue por tanto un *desarrollo incremental*, añadiendo pequeñas funcionalidades o porciones de código funcional, centrándome en un aspecto específico en cada commit subido a github.

Se asignarán los roles propios de Scrum. El papel de *Equipo de desarrollo* recae sobre el estudiante, y al ser únicamente una persona, también actuará como *Scrum Master* siendo responsable de la correcta aplicación de la metodología y prácticas al proyecto. La tutora actuará como *Product Owner*, conoce las necesidades del proyecto y el dominio del problema, hace de intermediaria con el cliente y ayuda al equipo de desarrollo a priorizar las tareas. Como posibles clientes o personas interesadas (*stakeholders*) podemos incluir a los expertos, con los que se mantiene el contacto frecuentemente a través del Product Owner.

Al ser OpenBabel una librería existente es importante mantener unos *estándares de codificación* y un estilo consistente. Teniendo eso en mente, y usando el *Refactoring*, se realizan cambios manteniendo el código limpio y legible. Se han usado las guías propias de OpenBabel y la documentación oficial orientada a desarrolladores para añadir nuevas funcionalidades[2,3]. Además, se busca ante todo la simplicidad a la hora de programar, evitando añadir funcionalidades extremadamente complejas o innecesarias y la adopción de soluciones sencillas.

3.2. Planificación

Inicialmente se elaboró una planificación estimada general, dividida en bloques grandes de trabajo, para establecer marcos de tiempo y controlar el ritmo del proyecto. Se puede ver en la Figura 3.2

Conforme se iba profundizando en el dominio del problema y entendiendo más sobre la problemática, se fueron subdividiendo y priorizando las tareas. A continuación, se desglosan las tareas definidas para el proyecto, separados en sprints:

[2] `https://openbabel.org/docs/current/OpenBabel.pdf`
[3] `https://acortar.link/Openbabel_Adding_Plugins`

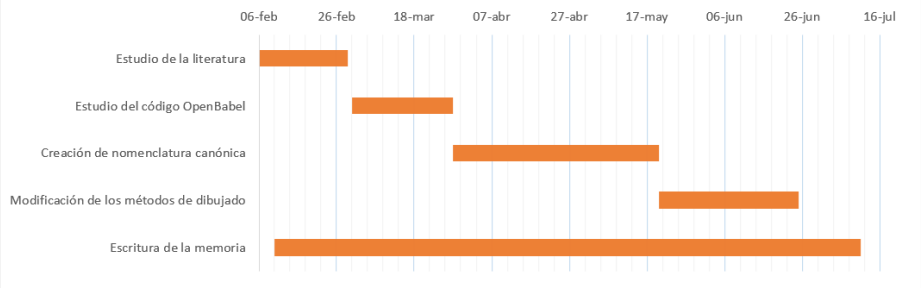


Figura 3.2: Diagrama de Gantt sobre la planificación inicial estimada

- **Bloque 1**: Investigación y aprendizaje previo. Estudio del estado del arte.
  - Lectura artículos y publicaciones sobre Chemoinformatics
  - Repaso de química general y estudio de química organometálica
  - Experimentación y pruebas iniciales con distintas herramientas
- **Bloque 2**: Estudio del paquete OpenBabel
  - Lectura detallada del código OpenBabel
  - Experimentación moléculas usando el dibujado de OpenBabel
- **Bloque 3**: Proceso de mejora del sistema de dibujado
  - Detección de estructuras de ciclopentadienilo (Cp)
  - Modificación dibujado moléculas con Cp individuales
  - Detección de múltiples Cp en la misma molécula usando bloques
- **Bloque 4**: Creación de nomenclatura canónica
  - Detección de bloques en el SMILES y creación del árbol genérico
  - Desarrollo del algoritmo canónico para moléculas con 1 metal
  - Algoritmo canónico para moléculas con 2 o más metales
- **Bloque 5**: Documentación del proyecto
  - Redacción de la memoria

En el siguiente Diagrama de Gantt (Figura 3.3) se presenta la planificación real del proyecto:

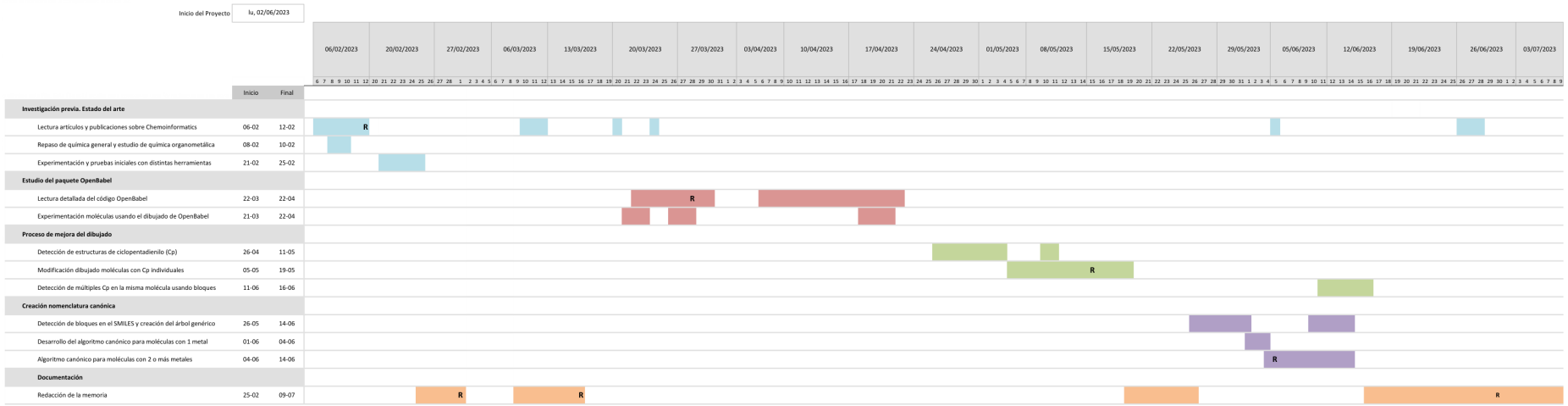


Figura 3.3: Diagrama de Gantt sobre la planificación temporal real

Como describo en la Sección 3.6.1, el proceso de compresión del código base se alargó más de lo esperado, retrasando un poco el resto de tareas. En contraparte, la canonización y dibujado de las moléculas me llevó menos tiempo del estimado. Además, se invirtió el orden entre la modificación del sistema de dibujado y la creación del algoritmo de canonización con respecto a la planificación inicial. A priori era más sencillo alterar esa parte del código, y se podían obtener resultados más visuales.

3.3. Gestión de la configuración

En este apartado se describirá cómo se ha llevado a cabo la gestión de los activos de este proyecto, es decir, el código desarrollado y la documentación generada. Dado que son partes fundamentales del trabajo, atendiendo al análisis de riesgos descrito en la Sección 3.6 y el plan de actuación frente a la pérdida de información, se detalla a continuación la gestión de ambas.

3.3.1. Gestión del código

Para la gestión del código se ha usado un control de versiones a través de Git y Github. Ambas herramientas en conjunto permiten ir creando versiones intermedias del código conforme se va desarrollando y hacer copias de seguridad en la nube. También son muy útiles para proyectos colaborativos, donde varias personas del equipo pueden combinar fácilmente su parte del código desarrollado y revisar el progreso subido hasta el momento. El procedimiento a seguir en la mayoría de casos es bastante similar. Se creará un repositorio remoto en la plataforma de GitHub con el nombre de *TFG* —o el que uno prefiera— donde se irán almacenando los cambios[4]. En la carpeta de trabajo local de nuestro computador, carpeta que contendrá en mi caso todos los elementos de los que quiera llevar un control, se creará un repositorio local usando Git, que habrá que vincular con el remoto para poder ir sincronizando los cambios.

3.3.2. Gestión de la documentación

En lo relativo a la documentación, el proceso de gestión será similar ya que también se ha usado GitHub para su control de versiones. Esta se ha redactado en LaTeX usando el servicio online Overleaf. Overleaf cuenta con una opción para sincronizar el proyecto con un repositorio de GitHub, pero es una opción de pago. En su lugar, descargo el proyecto en el repositorio

[4] `https://github.com/Jesnm01/TFG`

local y desde ahí ya realizo dicha sincronización para subir los cambios en el remoto.

Además, dada la naturaleza del proyecto y de la metodología de desarrollo utilizada, se ha llevado un registro de las reuniones con la tutora que también se ha ido actualizando periódicamente. Este documento pretende recoger los contenidos más relevantes de las reuniones: preguntas, comentarios, anotaciones, tareas que hacer, cosas pendientes de una reunión a otra, revisiones, y puntos a mejorar, entre otras cosas.

3.4. Gestión de recursos

3.4.1. Recursos humanos

- **Dña. Rocío Celeste Romero Zaliz**, profesora del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada en calidad de tutora del proyecto. A cargo de la supervisión y guía del alumno durante su desarrollo del trabajo.

- **Jesús Navarro Merino**, estudiante del grado en Ingeniería Informática en la Escuela Técnica Superior de Ingenierías Informática y Telecomunicación.

3.4.2. Recursos materiales

Para este proyecto no se han necesitado recursos adicionales, habiéndose usado únicamente los siguientes recursos materiales, ya existentes:

- **Portátil personal**: Portátil ACER Aspire A515-51G-8907 con un procesador Intel Core i7 8550U 1.8GHz, 20GB de memoria RAM y una arquitectura de 64 bits. Se ha usado durante todo el proyecto, para labores de programación y redacción de la memoria.

- **Pantalla**. Monitor utilizado de apoyo a la pantalla propia del portátil. De la marca AOC, de 24 pulgadas con una resolución de 1920x1080.

3.4.3. Recursos software

En esta sección describiré todas aquellas herramientas software empleadas durante la realización del proyecto. Todas y cada una de ellas son herramientas de software libre, gratuitas o disponibles a través de licencias de estudiantado. A continuación se lista el software usado:

- **Sistema operativo**: Windows 10 Home. Aunque por lo general Windows no es gratuito, al estar utilizando la típica licencia OEM que trae preinstalada el ordenador al comprarlo, la considero como tal.

- **Visual Studio C++**: es un IDE muy potente de Microsoft orientado a crear aplicaciones .NET y C++ para Windows. Se ha usado su versión gratuita Visual Studio Community 2022. Permite editar, depurar, realizar pruebas de testing, además de tener control de versiones integrado, entre otras cosas. Aquí se ha llevado a cabo todo el desarrollo del código.

- **OpenBabel**: Open Babel es una biblioteca de código abierto multiplataforma utilizada en química computacional y ciencias relacionadas para la conversión y manipulación de estructuras químicas en varios formatos. He trabajado con la versión 3.1.1 disponible en su repositorio de GitHub oficial[5].

- **CMake**: es una herramienta de generación de archivos de compilación que simplifica el proceso de compilación y construcción de proyectos, permitiendo una configuración flexible e independiente de la plataforma. CMake utiliza archivos de configuración llamados CMakeLists.txt para describir la estructura del proyecto y las dependencias necesarias. Se ha usado en su versión 3.25.2 para la compilación y creación de soluciones de OpenBabel.

- **Git**: software de código abierto para el control de versiones de un proyecto.

- **Github**: es una plataforma donde se alojará el código y la documentación del proyecto. Utiliza Git por debajo y es una de las plataformas gratuitas para alojamiento de código mas empleadas a nivel mundial.

- **Google Colab**: es una plataforma en línea gratuita ofrecida por Google que permite a cualquier usuario escribir y ejecutar código en el navegador. Es una herramienta basada en la nube que proporciona un entorno de ejecución como si fueran notebooks de Jupyter, es decir, se puede escribir, editar y ejecutar código en bloques/celdas interactivos. Se ha usado durante las etapas iniciales del proyecto para la experimentación con diversas moléculas.

- **Zotero**: software de gestión de referencias bibliográficas que permite recopilar, organizar, citar y generar fácilmente una bibliografía en varios estilos de formato estándares según los documentos, páginas webs, artículos o archivos PDF guardados. Facilita la creación de referencias

[5] `https://github.com/openbabel/openbabel`

y citas en documentos académicos, ahorrando tiempo y asegurando un uso correcto de las fuentes consultadas.

- **Google Meet**: servicio de videoconferencias de Google. Plataforma utilizada para las reuniones con la tutora.
- **Google Drive Sync**: ahora llamada Google Drive para PC, es una aplicación de Google que permite sincronizar los archivos y carpetas de la computadora con la cuenta de Drive. Usado para realizar copias de seguridad —adicionales a lo almacenado en GitHub— de algunos archivos importantes.
- **Overleaf**: herramienta online para la redacción de documentos en LaTeX usada para la documentación de este proyecto.
- **Scopus**: base de datos de referencias bibliográficas y citas de la empresa Elsevier.
- **Clockify**: herramienta online que te permite registrar las horas dedicadas a un proyecto.
- **Umbrello**: herramienta que combina funciones de modelado y generación de código para el lenguaje unificado de modelado (UML). Usado para la elaboración de los diagramas de clases.
- **Draw.io**: herramienta online para la creación de diagramas variados, esquemas y bocetos.
- **Correo UGR**: servicio de correo electrónico institucional de la UGR.
- **Microsoft Word**: procesador de textos de Microsoft usado para apuntes personales y documentación en sucio. Disponible a través de la cuenta de Microsoft Office 365 que ofrece la universidad.
- **Microsoft Excel**: utilizado para la creación de algunas tablas y gráficos incluidos en la memoria. Disponible a través de la cuenta de Microsoft Office 365 que ofrece la universidad.
- **Visual Studio Code y LaTeX**: VSCode es un editor de texto, que a través de algunas extensiones, permite editar, compilar y visualizar ficheros LaTeX. Es la alternativa a Overleaf según lo descrito en la Sección 3.6.

3.5. Gestión de costes

TERMINAR esto cuando vaya acabando el trabajo

En esta sección se realizará una estimación de los costes asociados al proyecto, atendiendo a los recursos descritos en la Sección 3.4. La elaboración de un presupuesto preciso en muchos casos puede suponer un desafío, ya que los proyectos software a menudo están sujetos a cambios y variables imprevistas, pero es una tarea importante para estudiar su viabilidad.

3.5.1. Coste de recursos humanos

Si actuáramos como una empresa en un proyecto software al uso, existen muchos componentes que tener en cuenta para montar esta sección del presupuesto. Aspectos como el salario de los trabajadores y compensaciones varias, el coste de los procesos de selección y contratación, formación y desarrollo del personal, costes laborales adicionales (seguros, vacaciones, etc), o costes de personal externo (consultores o subcontratistas) entre otras cosas. Dada la naturaleza de este proyecto, en relación a los gastos asociados a recursos humanos contamos con un equipo de desarrollo formado por una persona que tendrá el papel de Ingeniero Informático. Para estimar el costo de su trabajo, se indica el número total de horas dedicadas.

El intervalo de tiempo que está pensado para el proyecto son 5 meses, desde febrero hasta junio.

$$Dias\ totales\ =\ 5\ meses\ *\ 30\ dias/mes\ =\ 150\ dias$$

A este tiempo hay que descontarle los días no laborables de fines de semana más los días de vacaciones correspondientes por mes trabajado:

$$Dias\ no\ laborables = \left(\frac{8 dias}{mes} + \frac{22\ dias\ vacaciones}{12\ meses}\right) * 5\ meses \approx 50\ dias$$

Eso nos deja un total de 100 días aproximadamente de trabajo. Teniendo en cuenta una media semanal de 30 horas trabajadas repartidas en 6 horas diarias, tendríamos el siguiente total de horas trabajadas:

$$Horas\ trabajadas = 100 dias * 6\ horas/dia = 600\ horas$$

Esta cifra duplicaría la cantidad de horas que le corresponderían a un TFG según sus créditos asignados por la universidad. Estos cálculos como tal serían una estimación, pero se ha utilizado durante el desarrollo del proyecto la herramienta Clockify para el registro real de las horas dedicadas. Como se ve en la figura 3.4, han sido **XXXXX** horas en total, por lo que usaré ese dato para calcular el coste.

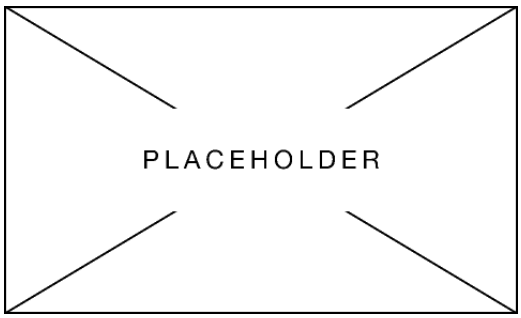


Figura 3.4: Registro de horas dedicadas al proyecto a través de Clockify

Según varias fuentes[6,7,8], el salario medio de un ingeniero informático recién graduado está en torno a 21.000€. Siendo equivalente, unos 1750€ mensuales, que son 13,46€/hora. Con todo eso, tenemos que:

$$Coste\ total\ recursos\ humanos = XXXX horas * 13,46\,€/hora = XXX\,€$$

3.5.2. Costes de recursos materiales

Dado que no se han adquirido expresamente para este proyecto, ya que se poseían con anterioridad, no se valora su precio de compra como tal sino su valor de depreciación. Los productos electrónicos experimentan un proceso llamado depreciación, que conlleva una devaluación gradual a lo largo de su vida útil. Es importante tener esto en cuenta para estimar su valor actual y el coste del recurso. Esta estimación refleja el valor de un activo desde un punto de vista contable.

Al referirnos a la depreciación de un activo a lo largo de su vida útil, no se incluyen situaciones en las que sufra daños debido a accidentes, desastres naturales u otros eventos similares. En cambio, estamos hablando del desgaste del uso cotidiano, así como los impactos derivados de las innovaciones tecnológicas que surgen durante ese período que puedan dejar obsoleto el dispositivo.

Para calcular el valor actual de los activos utilizaré el método de depreciación lineal, que considera un desgaste uniforme durante su uso, mostrando el resultado del gasto anual de depreciación[40]. Necesitamos lo primero, hallar el valor residual del activo, es decir, el valor que se estima que tendrá cuando llegue al final de su vida útil. Usaré para los dispositivos electrónicos una vida útil de 8 años. Haré un ejemplo con el coste del portátil.

[6] `https://www.jobted.es/salario/ingeniero-informtico`
[7] `https://acortar.link/infojobs_salario`
[8] `https://acortar.link/uax_salario`

$$Valor\ residual = \frac{Coste\ inicial}{Vida\ util\ (años)} = \frac{689\,€}{8\ años} = 86,125\,€$$

Con el valor residual estimado, se puede calcular la depreciación lineal anual con la siguiente fórmula.

$$Depreciación = \frac{Coste\ inicial - Valor\ residual}{Vida\ util\ (años)} = \frac{689\,€ - 86,125\,€}{8\ años} = 75,36\,€$$

Teniendo estos 2 datos, se puede calcular el valor actual del activo teniendo en cuenta sus años de antigüedad. Se muestran todos los datos relativos a los costes materiales en la Tabla 3.2.

Recurso	**Valor inicial (€)**	**Depreciación anual (€)**	**Antigüedad (años)**	**Valor actual (€)**
Portátil personal	689	75,36	5	312,2
Pantalla AOC	230	25,15	2	179,69
			Total:	491,89

Tabla 3.2: Tabla de los costes materiales

3.5.3. Costes software

Los costes relacionados con los recursos software son nulos. Como indico en el listado de recursos de la Sección 3.4.3, son herramientas de software libre, utilizadas mediante licencias gratuitas o mediante acceso vía instituciones autorizadas como la UGR, por lo que no suponen coste alguno en el desarrollo de este proyecto.

3.5.4. Otros costes

Aquí se incluyen todos los demás gastos que han sido necesarios para el desarrollo del proyecto y que no pertenecen a los apartados anteriores. Principalmente son los gastos vinculados a las facturas de la luz e Internet. El coste de la tarifa de Internet contratada es de 40€/mes, que a lo largo de los 5 meses el total asciende a 200€. Para la luz, una estimación posible serían 19,58€, teniendo en cuenta el gasto que suponen los recursos materiales en base a una factura trimestral de 11,80€.

Recursos	Importe (€)
Luz	19,58
Internet	200

Tabla 3.3: Tabla de costes adicionales

3.5.5. Presupuesto final

Se presenta por tanto el presupuesto completo asociado al proyecto, dividido en cada una de las secciones tratadas anteriormente. Se ve el desglose en la Tabla 3.4.

Detalle	Importe
Costes de recursos humanos	**X €**
Trabajo autónomo	X €
Costes de recursos materiales	**491,89 €**
Portátil personal	312,2 €
Pantalla de apoyo	179,69 €
Costes de recursos software	**0,00 €**
Windows 10	0,00 €
Visual Studio C++	0,00 €
OpenBabel	0,00 €
CMake	0,00 €
Git	0,00 €
GitHub	0,00 €
Google Colab	0,00 €
Zotero	0,00 €
Google Meet	0,00 €
Google Drive Sync	0,00 €
Overleaf	0,00 €
Scopus	0,00 €
Clockify	0,00 €
Draw.io	0,00 €
Correo UGR	0,00 €
Microsoft Word	0,00 €
Microsoft Excel	0,00 €
Visual Studio Code y LaTeX	0,00 €
Costes adicionales	**219,58 €**
Internet	40€ x 5 meses = 200€
Factura de la luz	19,58 €

Total:	**X €**

Tabla 3.4: Presupuesto total del proyecto

3.6. Análisis de riesgos

El análisis de riesgos desempeña un papel fundamental en la planificación y ejecución exitosa de proyectos. A veces surgen imprevistos que pueden afectar en mayor o menor medida a la correcta evolución de estos. Por ello, la aplicación de este proceso resulta esencial para minimizar la incertidumbre, intentar evitar la aparición de esos riesgos, y en caso de que se materialicen, paliarlos o mitigarlos de manera efectiva mediante unos planes de actuación.

En este aparatado por tanto, se analizarán los riesgos potenciales del proyecto, incluyendo sus causas y el plan de acción para resolverlos o mitigar su impacto al máximo (Figura 3.5). Además, se realizará una evaluación de la probabilidad de ocurrencia y del impacto asociado a cada riesgo, que se puede ver en la Figura 3.6. Esto se basa en una matriz con 2 dimensiones: la probabilidad de ocurrencia de un riesgo y el impacto que tendría en el proyecto si se materializa. Se valorarán los riesgos según aspectos técnicos, de recursos humanos o complejidad y naturaleza del proyecto, y preguntas del tipo, ¿qué tan difícil sería recuperarse del riesgo?, ¿cuál es el resultado más negativo que podría originarse como consecuencia? o ¿ha sucedido este riesgo o alguno similar anteriormente?

3.6.1. Riesgos materializados

Los riesgos materializados han estado relacionados principalmente con aspectos técnicos. Primeramente, el R.5. Tuve problemas para instalar Open-Babel en mi máquina por problemas de versiones, por lo que acabé utilizando Google Colab como entorno virtual e instalar ahí algunas librerías necesarias para las primeras experimentaciones con moléculas. Esto tampoco era muy útil a largo plazo puesto que tenía que poder acceder al código fuente para modificarlo, añadir las funcionalidades y compilarlo manualmente para probar los cambios. Por lo que finalmente con ayuda de las guías, se pudo ejecutar localmente. Instantáneamente después, se materializó el riesgo R.3. La falta de conocimiento ante una librería tan grande ya existente retrasó considerablemente el proceso de modificación del código.

Finalmente, se consiguieron solventar los riesgos manifestados mediante los planes de actuación descritos en cada uno de ellos.

ID	Riesgo	Causa	Plan de acción
R.1	Pérdida de documentación y/o código	Eliminación accidental de ficheros clave	Realizar copias de seguridad y herramientas de control de versiones para su recuperación.
R.2	Fallo en el hardware de trabajo	Fallo irreparable por edad de uso, sobrecalentamiento, golpe o rotura.	Adquisición de nuevo portátil. Instalación y configuración del entorno de trabajo cuanto antes.
R.3	Falta de conocimiento de los paquetes de chemoinformatics	Nula experiencia con toolkits químicos y sus prestaciones	Invertir tiempo en el estudio y compresión del código base.
R.4	Cambios inesperados y tareas nuevas no contempladas	Planificación poco adecuada a las necesidades del proyecto	Replanifiación de las tareas existentes y nuevas, con una adecuada asignación de tiempo
R.5	Problemas en la instalación y configuración de las herramientas	Incompatibilidad de versiones, la máquina en donde se va a realizar la instalación o complejidad del proceso	Consulta de las guías de instalación, búsqueda de herramientas alternativas viables o uso de máquinas virtuales/entornos de desarrollo distintos
R.6	Falta de feedback directo con el tutor	Incompatibilidades horarias, o enfermedad de alguna de las partes	Agendar una reunión en otra fecha, y si fuera necesario o viable, enviar el material a revisar/preguntas por correo para feedback asíncrono.
R.7	Los objetivos del proyecto no son realistas	Se ha infravalorado el alcance del proyecto	Realizar una planificación coherente con el tiempo y recursos disponibles. Si fuera necesario, adaptar el contenido del trabajo original y proponer los cambios
R.8	Los resultados del TFG finalmente no son satisfactorios o los esperados	A la tutora (y los químicos colaboradores) no les gusta el resultado	Recibir feedback periódicamente de los cambios realizados y opiniones sobre los resultados intermedios.
R.9	Dificultades para crear un sistema canónico viable para cualquier molécula de entrada	La tarea de programación para llevar esto a cabo es más compleja de lo que se creía	Proponer, aunque sea un inicio de canonización y detallar formalmente las reglas que lo definen. O, sabiendo cómo funcionan internamente las librerías, buscar un sistema canónico más sencillo en términos de programación.
R.10	Imposibilidad de acceso al servicio de Overleaf para la redacción de la memoria	Caída temporal de los servidores	Si es muy urgente, uso de Visual Studio Code para compilado local de la documentación.
R.11	Finalización del uso gratuito de alguna herramienta utilizada	Conclusión de las licencias gratuitas o periodos de prueba	Extensión de la licencia en caso de ser posible o búsqueda de herramientas alternativas parecidas/compatibles.

Figura 3.5: Riesgos del proyecto, causas, y planes de actuación

	Probabilidad				
Impacto	Muy improbable (0,1)	Poco probable (0,3)	Moderada (0,5)	Probable (0,7)	Casi seguro (0,9)
Muy bajo					
Bajo	R.11	R.12	R.4		
Medio	R.6	R.9	R.8		
Alto	R.1			R.3	
Muy alto	R.2		R.5, R.10		

Figura 3.6: Matriz de probabilidad-impacto de riesgos

Capítulo 4

Diseño e implementación

En este capítulo se describirán las clases y todos los métodos que se han añadido y modificado durante el proceso de implementación a partir del código base de OpenBabel en su versión 3.1.1 (disponible en su repositorio oficial de GitHub[1]). OpenBabel está escrito en C++, por lo que el proceso de desarrollo se ha llevado a cabo exclusivamente en este lenguaje, usando para ello el IDE *Visual Studio C++* para Windows, (en el Anexo A se detalla este proceso).

4.1. Diseño

4.1.1. Estructura de OpenBabel

A partir del código fuente de OpenBabel, al compilar los archivos fuente y generar el proyecto, los archivos de configuración de CMake generan una serie de *soluciones*. Hay soluciones que consisten únicamente en el archivo '*main*', que representa el ejecutable al que llamamos por línea de órdenes desde la terminal. El resto de soluciones forman la propia API de OpenBabel, siendo las clases más importantes '*OBMol*', '*OBAtom*', y '*OBBond*', que permiten almacenar la información de una molécula, un átomo, o un enlace entre átomos respectivamente; y otras clases más orientadas a la conversión entre formatos como '*OBConversion*' u '*OBFormat*'. De entre ellas la clase central es OBMol, que almacena toda la información básica relacionada con una molécula, incluyendo la lista de átomos, la lista completa de enlaces entre átomos, identificadores de cada átomo y enlaces y entre otras variables, el vector de coordenadas 2D de todos los átomos para su representación gráfica.

[1] `https://github.com/openbabel/openbabel/releases`

A continuación se muestra la estructura general de directorios de Open-Babel:

```
openbabel-3-1-1/
├── cmake/ ..... algunos ficheros de configuración para cmake
├── data/ .....
├── doc/ ..... documentación del proyecto y ficheros para su generación
│   automática
├── include/ ..... ficheros .h
│   └── openbabel/
│       ├── depict/ ..... representación de moléculas
│       ├── math/
│       ├── stereo/
│       ├── tree/
│       └── clases principales de Openbabel
│           └── ...
├── src/ ..... ficheros .cpp
│   ├── charges/
│   ├── depict/ ..... representación de moléculas
│   ├── descriptors/
│   ├── fingerprints/
│   ├── formats/ ..... soporte para distintos formatos de conversión
│   ├── math/
│   ├── ops/ ..... plugins de la comunidad
│   ├── stereo/
│   └── clases principales de Openbabel
│       └── ...
├── scripts/ ..... bindings para usar la interfaz en otros lenguajes
├── test/ ..... ficheros de ejecución de tests y datos de prueba
├── tools/ ..... 'mains' para ejecución por línea de órdenes
├── CMakeLists.txt ..... archivo principal de configuración de cmake
├── INSTALL ..... instrucciones breves de instalación de openbabel
└── ficheros propios de .git .....
```

4.1.2. Diagrama de Clases

En el siguiente Diagrama de clases (Figura 4.1) se muestran tanto las clases que se han visto modificadas (en color anaranjado), las creadas desde cero (en color más verdoso) y las demás clases importantes que interactúan con las anteriores pero no se han visto alteradas (en amarillo).

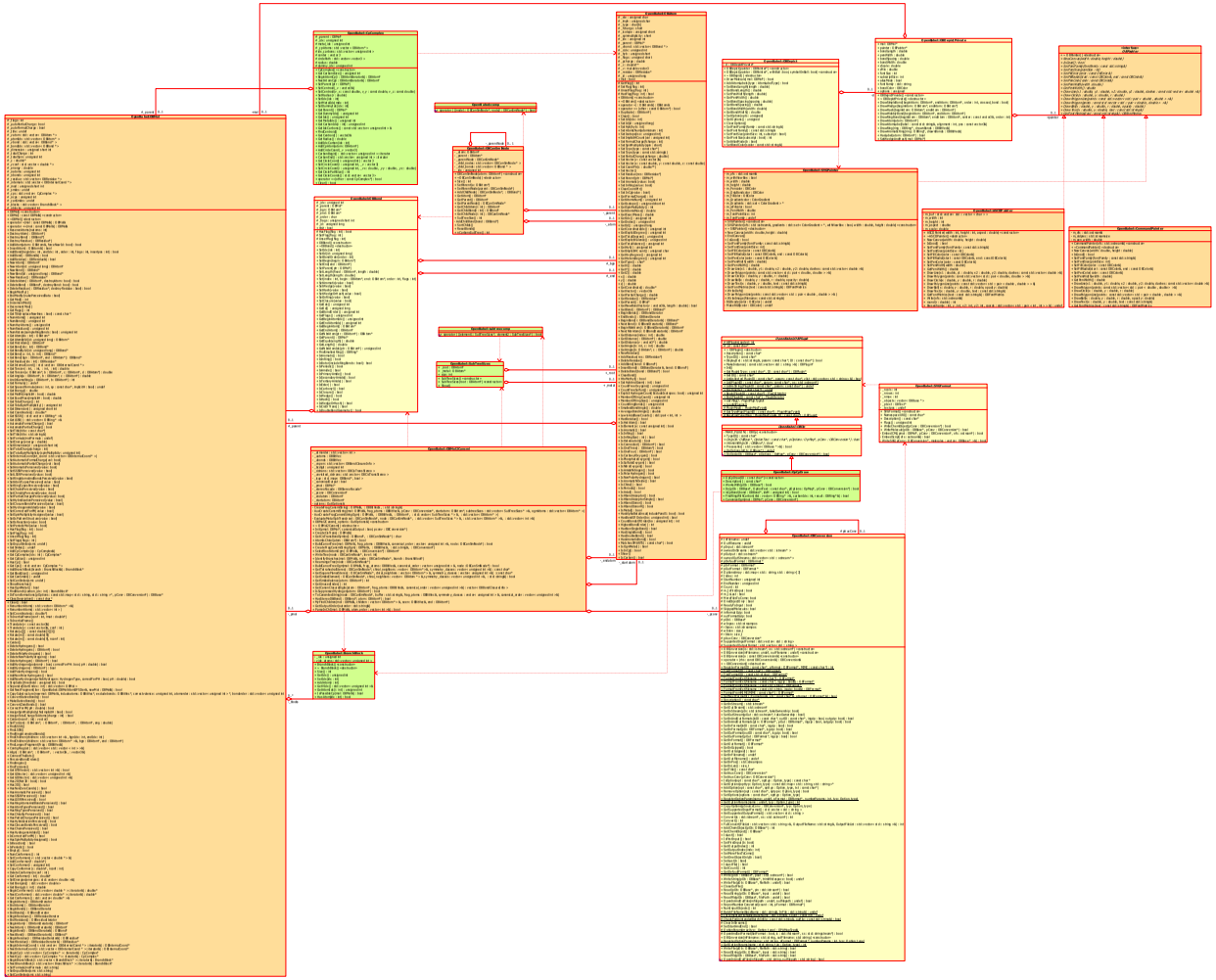


Figura 4.1: Diagrama de clases

Se pasa a detallar ahora cada una de las clases, tanto las modificadas como las nuevas, para qué sirven, y en qué consisten sus métodos. Las clases que no se han alterado, al igual que el resto de clases que no se incluyen en el diagrama se puede consultar su documentación en la página oficial[2]. Puntualizar que existe una enorme cantidad de clases en la librería de Open-Babel, no tendería sentido añadirlas todas en el diagrama. Además, no todas poseen de documentación, por lo que la mayoría no aparecerán en[1].

4.1.3. Clases modificadas

- **OBPainter**: clase base abstracta para las clases de representación gráfica 2D (en */depict/painter.h*). Se ha añadido el siguiente método para poder utilizarlo en las clases que implementan esta interfaz:

```
        public:

virtual void DrawPolygonLine(const std::vector<std::pair<double,
    double> >& points) = 0;
```

- **SVGPainter**: clase que hereda de OBPainter y genera representaciones 2D en el formato de gráficos vectoriales SVG (en */depict/svgpainter.h*).

```
        public:

//Inserts the necessary xml code in the .svg output file to draw
    a polygon according to the vector of points specified by @p
    points
void DrawPolygonLine(const std::vector<std::pair<double, double>
    >& points);
```

- **ASCIIPainter**: clase que hereda de OBPainter (en */depict/asciipainter.h*).

```
        public:

//The method is declared empty to avoid compilation errors due to
     interface implementation. It has no use
void DrawPolygonLine(const std::vector<std::pair<double, double>
    >& points);
```

- **CommandPainter**: clase que hereda de OBPainter (en */depict/commandpainter.h*).

```
        public:

//The method is declared empty to avoid compilation errors due to
     interface implementation. It has no use
```

[1] https://openbabel.github.io/api/3.0/index.shtml

```
void DrawPolygonLine(const std::vector<std::pair<double, double>
    >& points);
```

- **OBAtom**: clase principal, contiene la información relativa a un átomo, guardando su número atómico, cantidad de hidrógenos implícitos, una lista de los enlaces de este átomo con los demás y el vector de coordenadas 2D para su representación, entre otras variables (en */openbabel/atom.h*). Se han añadido los siguientes métodos:

```
class OBAPI OBAtom: public OBBase{
    public:

//\return Is this a metal commonnly present in organometallic
    compounds?
bool IsOgmMetal();

//\return Is atom part of a Cp ring?
bool IsInCp() const;

//\return Is this atom a Carbon (atomic number == 6)?
bool IsCarbon();

//Debug method. Displays on basic output simple data to identify
    the atom
void Show();

//Mark an atom as part of a Cp ring
void SetInCp(bool value = true);
};//class
```

- **OBMol**: clase principal, almacena toda la información básica relacionada con una molécula. Esto incluye la lista de átomos, la lista completa de enlaces entre átomos, identificadores de los átomos y enlaces, y el vector de coordenadas 2D de todos los átomos para su representación entre otras variables (en */openbabel/mol.h*). Se han añadido las siguientes variables y métodos:

```
class OBAPI OBMol: public OBBase {
    private:

std::string _smiles;                //!< Input smiles string for
    the molecule
std::vector<CpComplex*> _cps;       //!< Cp information
unsigned int _ncps;                 //!< Number of cps complexes
    detected
std::string  _canSmiles;            //!< Canonical smiles based
    on Ogm canonicalization
std::vector<BranchBlock*> _blocks;  //!< Branches information
unsigned int _nblocks;              //!< Number of blocks


    public:

//! Set the input smiles string of this molecule to @p smi
void SetInputSmiles(std::string smi);
```

```
//! \return the input smiles string of this molecule
std::string GetSmiles();

//! Add a new CpComplex specified by @p cp
void AddCpComplex(CpComplex& cp);

//! \return the cp at index @p idx or NULL if none exists.
CpComplex* GetCpComplex(int idx);

//! \return number of cp in the molecule
unsigned int GetCpSize();

//! \return whether the molecule has cps or not
bool HasCp();

//! \return the whole container of cps of this molecule
std::vector<CpComplex*> GetCps();

//! Add a new block to the molecule, specified by @p branch
BranchBlock* AddBranchBlock(BranchBlock& branch);

//! \return the number of blocks in the molecule
unsigned int GetBlockSize();

//! \return the canonical smiles string generated by the ogm
    canonicalization methods
std::string GetCanSmiles();

//! Set the canonical smiles string of this molecule to @p smi
void   SetCanSmiles(std::string smi);

//! Debug method. Displays on basic output all molecule blocks
    with basic information of the atoms.
void ShowBranches();

//! \return If this molecule has any Ogm metal or not
bool HasOgmMetal();

//! \return the block of which the carbon at index @p carbon_idx
    is part, or NULL if no such block exists
BranchBlock* FindBranch(int carbon_idx);

//! Set the iterator to the beginning of the Cp list
//! \return the first Cp structure, or NULL if none exist
CpComplex* BeginCp(std::vector<CpComplex*>::iterator & i);

//! Advance the iterator to the next Cp record
//! \return the next first Cp record, or NULL if none exist
CpComplex* NextCp(std::vector<CpComplex*>::iterator& i);

//! Set the iterator to the beginning of the BranchBlock list
//! \return the first BranchBlock structure, or NULL if none
    exist
BranchBlock* BeginBranchBlock(std::vector<BranchBlock*>::iterator
    & i);

//! Advance the iterator to the next BranchBlock record
//! \return the next first BranchBlock record, or NULL if none
    exist
BranchBlock* NextBranchBlock(std::vector<BranchBlock*>::iterator&
     i);
};//class
```

- **OBMol2Cansmi**: clase que maneja la conversión del smiles de entrada a un smiles canónico (en *src/formats/smilesformat.h*). Se han añadido los siguientes métodos:

```
class OBMol2Cansmi{
        private:

//Only changed visibility to private, since
    CreateFragCansmiStringOgm was created. Selects the "root"
    atom, which will be first in the SMILES, then builds a tree
    in canonical order, and finally generates the SMILES.
void CreateFragCansmiString(OBMol&, OBBitVec&, std::string&);


//Auxiliary private methods for SelectRootAtomOgm

//Shortened version of the CreateCansmiString method. Create the
    necessary variables to call AuxCreateFragCansmiStringOgm.
void AuxCreateCansmiString(OBMol& mol, OBBitVec& frag_atoms,
    OBConversion* pConv, OBAtom* startatom, std::vector<
    SubTreeSizes*>& subtreeSizes, std::vector<OBAtom*> ogmAtoms);

//Shortened version of the CreateFragCansmiStringOgm method.
    Create the necessary variables to build a new canonical tree
    using as root @p startAtom
void AuxCreateFragCansmiStringOgm(OBMol&, OBBitVec&, OBAtom*, std
    ::vector<SubTreeSizes*>&, std::vector<OBAtom*>);

//Once the tree is built, this method runs through it in DFS
    evaluating the subtrees hanging from the other ogm metals.
    Use the auxiliary struct SubTreeSizes for this.
void EvaluateMetalSubTrees(OBCanSmiNode* root, OBCanSmiNode* node
    , std::vector<SubTreeSizes*>&, std::vector<OBAtom*>&, std::
    vector<int>&);

        public:
//Method based on CreateFragCansmiString. Share much of the code,
     with some additional methods specifically for my own
    canonical form designed for organometallic molecules.
void CreateFragCansmiStringOgm(OBMol&, OBBitVec&, std::string&,
    OBConversion*);

//If more than 1 Ogm metal is present in the molecule, this
    method chooses one of them, based on some rules and the
    conectivity of the metal within the molecule and the rest of
    the atoms
OBAtom* SelectRootAtomOgm(OBMol&, OBConversion*);

//Debug method for writing in basic output the tree with
    hierarchy formating
void WriteTree(OBCanSmiNode* node, int level = 0);

//Adds information to the molecule of the blocks that form it.
    Being a block, each set of atoms that, due to their bonds,
    are within the same parenthesis in the original input Smiles.
     Or, according to the OBMol2Cansmi::BuildCanonTree method,
    the parent-child relationship between atoms.
void IdentifyBranches(OBMol& mol,OBCanSmiNode* node, BranchBlock*
     branch = nullptr);

//Modifies the tree built by BuildCanonTree based on the length
```

```
    of the branches identified in IdentifyBranches. This is a
    canonical rule designed for a little more consistency in the
    output canon smiles.
void RearrangeTree(OBCanSmiNode* node);

//Builds the SMILES tree, in canonical order, for the specified
    molecular fragment. Based on the BuildCanonTree method.
    Shares much of the code, with some changes in the neighbour
    selection algorithm.
bool BuildCanonTreeOgm(OBMol& mol, OBBitVec& frag_atoms, vector<
    unsigned int>& canonical_order, OBCanSmiNode* node);
};//class
```

- **OBCanSmiNode**: clase que representa un nodo. En conjunto se forma una estructura de árbol, cada nodo es un átomo del árbol para luego escribir el SMILES canónico (en *src/formats/smilesformat.h*). Se han añadido los elementos:

```
class OBMol2Cansmi{
        private:

OBCanSmiNode* _parentNode;      //!< Pointer to the parent node

//! Add a child bond to the node, specified by @p bond. Should
    only be used in the ResetBonds method as a part of the
    OBMol2Cansmi::RearrangeTree algorithm.
//! Otherwise, use addChildnode to add both the child node and
    its respective bonds
void AddChildBond(OBBond* bond);

        public:

//! Set the parent node to @p parent
void SetParentNode(OBCanSmiNode* parent);

//! \return the parent node
OBCanSmiNode* GetParentNode();

//! Traverses the tree in dfs from the node calling the method
//! \return the number of total children (counting himself)
int SubTreeSize();

//! Sort a node's child_nodes using a std::sort operation an a
    custom comparator 'mycomp'
void SortChilds();

//! When added at the same time in the addchildnode method, the
    child with its bond have a 1 to 1 index correspondence. When
    reordering the children, in OBMol2Cansmi::RearrangeTree, the
    indices of the bonds are lost. This method clears and adds
    the bonds back in order.
void ResetBonds();

//! \return the total number of carbons in this node subtree
int nCarbonsSubTree();
};//class
```

4.1.4. Clases nuevas

- **CpComplex**: nueva clase principal que maneja y permite almacenar estructuras de ciclopentadienilo (en */openbabel/cpcomplex.h*). Se han creado las siguientes variables y métodos:

```
class CpComplex {
        protected:

OBMol* _parent;                          //!< Parent molecule
unsigned int _idx;                       //!< Cp identifier within
     the molecule
unsigned int metal_idx;                  //!< Atom idx of central
    metal
std::vector<OBAtom*> _cpAtoms;           //!< Atoms for the
    carbons of the Cp structure
std::vector<unsigned int> idx_carbons;   //!< Atom indexes for the
     carbons of the Cp structure
vector3 center;                          //!< Cp center, for
    normal bond connection with metal atom, and aromatic circle
    position
std::vector<vector3> circlePath;         //!< Coordinates for the
    cp circle (needed to achieve a perspective circunference)
double radius;                           //!< Cp's aromatic circle
     radius
unsigned int dummy_idx;                  //!< Dummy central atom
    idx


        public:

//! Default constructor
CpComplex();

//! \name Methods to modify internal information
//@{
//! Attach an OBMol @p ptr as the parent container for this Cp
void SetParent(OBMol* ptr);
//! Set the center point of the Cp, sprecified by @p _v. It is
    equidistant to every carbon in th Cp, as they are disposed in
     a regular polygon
void SetCentroid(vector3& _v);
//! Set the center point of the Cp, sprecified by @p v_x, v_y,
    v_z. It is equidistant to every carbon in th Cp, as they are
    disposed in a regular polygon
void SetCentroid(const double v_x, const double v_y, const double
     v_z);
//! Set the radius of the Cp circle
void SetRadius(double r);
//! Set the Cp identifier
void SetIdx(int idx);
//! Set the idx of the central metal to which this Cp is attached
void SetMetalIdx(int midx);
//! Dummy atom is created to make a perpendicular bond between
    the metal and the Cp drawing
//! Set the atom idx of the dummy atom created for this Cp
void SetDummyIdx(int idx);
//! Set the point of the Cp circle at index @p i to the
    coordinates specified by @p _v
void SetCircleCoord(unsigned int i, vector3 _v);
```

```
//! Set the point of the Cp circle at index @p i to the
    coordinates specified by @p _vx, _vy, _vz
void SetCircleCoord(unsigned int i, double _vx, double _vy,
    double _vz = 0.0);
//@}


//! \name Methods to retrieve information
//@{
//! \return number of carbon atoms in the cp
unsigned int GetCarbonsSize();
//! \return the molecule which contains this Cp, or NULL if none
    exists
OBMol* GetParent();
//! \return dummy atom idx for this Cp structure, or 0 if none
    exists
unsigned int GetDummyIdx() const;
//! \return Cp identifier
unsigned int GetIdx() const;
//! \return Central metal identifier
unsigned int GetMetalIdx() const;
//! \return carbon idx at position @p i in tha container. Zero
    based access method to vector
unsigned int GetCarbonIdx(int i) const;
//! \return the whole contanier of carbon idx
const std::vector<unsigned int>& GetIdxCarbons();
//! \return the centroid of this Cp in a coordinate vector
vector3& GetCentroid();
//! \return the radius of the Cp circle
double GetRadius();
//! \return the coordinate vector for the Cp circle point at
    position @p i in the container. Zero based access method to
    vector
vector3 GetCircleCoord(unsigned int i);
//! \return the number of points of the Cp circle
int GetCirclePathSize() const;
//! \return the whole container of coordinates of the Cp circle
std::vector<vector3> GetCircleCoords() const;
//@}


//! \name Addition of data for a Cp
//@{
//! Adds a new atom idx to this Cp
void AddIdxCarbon(int idx);
//! Adds a new atom to this Cp
void AddCpAtom(OBAtom* atom);
//! Adds a new point to the coordinate vector that forms the Cp
    circle
void AddCircleCoord(vector3 _v);
//@}


//! \name Iteration methods
//@{
//! Set the iterator to the beginning of the Cp atom list
//! \return the first atom, or NULL if none exist
OBAtom* CpComplex::BeginAtomCp(OBAtomIterator& i);
//! Advance the iterator to the next atom in the Cp
//! \return the next first atom record, or NULL if none exist
OBAtom* CpComplex::NextAtomCp(OBAtomIterator& i);
//@}
```

```
//! \name Other operations
//@{
//! Calculate and set the centroid of this Cp, taking into
    consideration all atoms stored in _cpAtoms
void FindCentroid();
//! Equivalence operator
bool operator==(const CpComplex* other) const;
//@}
};//class
```

- **BranchBlock**: clase que representa un grupo funcional aislado dentro de la molécula, p.ej. un ciclo de benceno, un Cp, o toda una rama de un átomo (en */openbabel/cpcomplex.h*). Se han añadido las siguientes variables y métodos:

```
class BranchBlock {
        private:

unsigned int _idx;                          //!< Block identifier
std::vector<unsigned int> vidx_atoms;       //!< Vector idx of
    the atoms that are part of the block.

        public:

//! Default constructor
BranchBlock();

//! Destructor
~BranchBlock();

//! \return the size of the block (number of atoms in the block)
int Size();

//! \return the block identifier
unsigned int GetIdx();

//! Set the block identifier
void SetIdx(int idx);

//! Add an atom's idx to the block
void AddAtom(int i);

//! \return the idx of the atom at position @p i. Zero based
    access.
unsigned int GetAtomIdx(int i);

//! \return Whether the @p idx exists within the atoms already
    inserted in the block
bool HasAtom(int idx);

//! Cp will be possible if all the elements in the block are
    carbons up to that point and have a bond with an ogm metal.
//! \return whether or not it appears to be a Cp block
bool IsPossibleCp(OBMol &mol)
}; //class
```

- **OpCpDraw**: clase plugin que hereda de OBOp (en */ops/cpdraw.cpp*).

Contiene el algoritmo de detección, identificación, y almacenamiento en la molécula de estructuras tipo Cp.

```
class class OpCpDraw : public OBOp {
//! Default constructor
OpCpDraw(const char* ID);

//! Inherited method.
//! Display through the output stream a brief description of the
    plugin.
const char* Description();

//! Inherited method.
//! \return true if this op (plugin operation) is designed to
    work with the class of @p pOb, e.g. OBMol
virtual bool WorksWith(OBBase* pOb) const;

//! Inherited method. Required function that does the work.
    Normally return true, unless object is not to be output.
virtual bool Do(OBBase* pOb, const char* OptionText = nullptr,
    OpMap* pOptions = nullptr, OBConversion* pConv = nullptr);

//! \return If @p bond is likely to be a cp-bond like
bool isCpBond(OBBond* bond, unsigned int idxM);

//! Finds the ring of which the carbon with idx @p carbonIdx is a
     part of, among the rings of @p rlist (obtained from a SSSR
    perspective), and stores it in @p result.
//! \returns whether it was found or not
bool FindRingWithCarbon(vector<OBRing*>& rlist, int carbonIdx,
    OBRing*& result);

//! Canonize the input SMILES and identify blocks
void CanonizeOgm(OBMol* mol, OBConversion* pConv);
}; //class
```

- **SubTreeSizes**: struct auxiliar creado para la selección del primer metal durante la canonización (se profundiza sobre esto en la Sección 4.2.1). Contiene las siguientes variables y métodos (en *src/formats/smilesformat.h*):

```
struct SubTreeSizes {

OBAtom* _root;      //!< Tree root
OBAtom* _metal;     //!< Metal to evaluate
int size;           //!< Size of the subtree for the _metal to
    evaluate
int nCarbons;       //!< Number of carbon atoms

//! Default constructor
SubTreeSizes();

//! Parameter constructor. Creates a new object with _root as @p
    root
SubTreeSizes(OBAtom* root);

//! Debug method. Displays through basic output the struct
    information.
void Show();
```

```
};//struct
```

- **subtreecomp**: objeto comparador que prioriza unos metales sobre otros en el proceso de selección del átomo raíz para el árbol (en *src/formats/smilesformat.h*).

```
struct subtreecomp {
bool operator() (SubTreeSizes* element1, SubTreeSizes* element2)
    const;
}subtreecomp;
```

- **mycomp**: objeto comparador que prioriza las ramas del árbol canónico durante el proceso de reordenación (en *src/formats/smilesformat.h*).

```
struct comp{
bool operator() (OBCanSmiNode* node1, OBCanSmiNode* node2);
}mycomp;
```

En resumen, se han modificado 8 clases existentes, sufriendo los cambios más importantes las clases OBMol, OBAtom, OBCanSmiNode y OBMol2Cansmi; y se han añadido 6 clases nuevas. En total, se han agregado 93 métodos y 22 variables nuevas. Con todo esto, en la siguiente sección se describen los algoritmos implementados.

4.2. Implementación

4.2.1. Nomenclatura canónica

La idea detrás de una canonización, como ya se ha comentado, es obtener una representación única para una molécula independientemente de su forma inicial. Para ello se suelen emplear estructuras de tipo árbol y realizar un recorrido de los nodos de una forma concreta. La información de la que disponemos para una molécula tras un proceso de parsing de la cadena SMILES de entrada, es una lista de átomos y una lista de enlaces entre átomos. Estas estructuras de datos lineales se utilizan para generar el árbol, donde los nodos serán los átomos y las aristas los enlaces. OpenBabel almacena esta estructura de datos a través de la clase *OBCanSmiNode*, que representa cada nodo individual y contiene: el átomo del nodo, el átomo del nodo padre, un vector con los nodos hijos y un vector con los enlaces a los hijos.

El SMILES de salida dependerá por tanto de 2 factores. Primero, el orden en el que se recorra el árbol, que dado nuestro objetivo de obtener un canónico, el recorrido será siempre el mismo. En este caso OpenBabel utiliza un recorrido en profundidad de preorden (DFS, Depth First Search en

inglés). Y segundo, aunque el recorrido sea el mismo, si el árbol cambia, el resultado será distinto. Por lo que se debe ser capaz de generar consistentemente el mismo árbol si la molécula es la misma. Con ese propósito se le asigna a cada nodo una etiqueta (label) única, independientemente de su orden de entrada, para luego a la hora de generar el árbol saber qué átomos tienen preferencia sobre otros. Este algoritmo de asignación de etiquetas es lo que suele diferenciar unos toolkits de otros.

En el estado actual, OpenBabel cuenta con su propio algoritmo de canonización en 2 versiones. Ambas por defecto realizan un tratamiento de ciclos, colocándolos en su forma aromática y reutilizando los números de apertura y cierre, pero se diferencian en la asignación de las etiquetas. Hay una versión canónica simple (*standard labels*) en la que se asignan las etiquetas en orden ascendente de llegada, del 0 al n siendo n el número total de átomos en la molécula. Por tanto, el SMILES canónico resultado no varía en nada respecto al SMILES de entrada excepto por el tratamiento de los ciclos.

La segunda versión es una versión canónica completa (*canonical labels*) basada en el algoritmo de Morgan y en el uso de los invariantes de un átomo [68, 31]. Estos invariantes son características únicas que clasifican a cada átomo en base a: su topología con respecto a los demás átomos en el grafo molecular, el número de conexiones, cantidad de hidrógenos enlazados, número atómico, carga eléctrica, etc. El algoritmo en sí mismo es bastante complejo, por lo que no se tratará más en profundidad. Para más detalles de su implementación y trasfondo ver [8, 52, 3, 68].

Para el objetivo del proyecto nos sirve con saber que dicho algoritmo proporciona buenos resultados y el uso de las *canonical labels* funciona adecuadamente para todas las moléculas, en el sentido de que las hace canónicas. Es efectivo también para moléculas organometálicas, pero no le da la suficiente importancia al metal entre otras cosas. El sistema de canonización propuesto en este proyecto está basado en las peticiones de los expertos. Siguiendo sus recomendaciones, les sería útil que en compuestos organometálicos el SMILES comenzara por el metal. En la Figura 4.2, se muestran algunos ejemplos de moléculas organometálicas canonizadas usando los *canonical labels*. Vemos que el metal se encuentra siempre enmarañado por medio del SMILES rodeado de otros enlaces que pueden ser o no ser suyos, entorpeciendo el ver qué conectividad tiene dicho metal.

Reglas canonizado

Se busca por tanto, lo primero de todo colocar el metal al inicio del SMILES y en base a esto ir colocando el resto de átomos vecinos. Esto también se cree que sería beneficioso a la hora de trabajar con modelos generativos. El tener una estructura legible y estricta en donde se fije el

Id	Original
8	[Br-][Ni+2]1([Br-])O(C)CCO1C
5	OC=1C=CC=2C(=[N](O)[Pd+2]3([Cl-][Pd+2]4([Cl-]3)[C-]=5C=C(O)C=CC5C(=[N]4O)C)[C-]2C1)C
25	Cl[Au]=C1N(C=CN1C=2C(=CC=CC2C(C)C)C(C)C)C=3C(=CC=CC3C(C)C)C(C)C

Id	Canónico
8	C[O]1CC[O]([Ni+2]1([Br-])[Br-])C
5	OC1=C[C-]2=C(C=C1)C(=[N]([Pd+2]12[Cl-][Pd+2]2([Cl-]1)[C-]1=C(C(=[N]2O)C)C=CC(=C1)O)O)C
25	CC(c1cccc(c1n1ccn(c1=[Au]Cl)c1c(cccc1C(C)C)C(C)C)C(C)C)C

Figura 4.2: SMILES canónicos pertenecientes a la organometálica según el algoritmo propio de OpenBabel. Los metales están marcados en rojo. Son 3 moléculas del dataset mostrado en el Anexo **??**, identificadas por su Id.

metal al principio de la cadena favorece la robustez de la representación [41, 35]. En base a la premisa de colocar el metal lo primero, se ha desarrollado un sistema canónico con las siguientes reglas:

- Tiene que haber un metal de interés para la organometálica (los metales de transición mencionados en la Sección 2.2.3) en el SMILES de entrada para aplicar el algoritmo de canonizado propio. En caso negativo, se aplicaría el original de OpenBabel.
- Si el SMILES está desconectado (por fragmentos) se aplicará el algoritmo original. El SMILES resultado es la concatenación de cada fragmento canonizado por separado.
- Para escoger el metal que da inicio al SMILES se siguen las siguientes prioridades:
    1. Si solamente hay un metal, se escoge dicho átomo directamente.
    2. Si hay más de 1 metal, se selecciona el que tiene mayor cantidad de subhijos (mayor tamaño de su subárbol) con respecto a los demás metales. Se considera que el metal más importante es del cuál dependen más átomos.
    3. Si la condición anterior resulta en empate, se toma el metal con mayor número atómico.
    4. Si la condición anterior resulta en empate, es decir, es el mismo elemento con la misma cantidad de subhijos, se toma el metal que de entre sus subhijos aparezcan menos carbonos. Ya que los carbonos son esenciales, son muy frecuentes. Por lo que se da prioridad a cualquier otro elemento antes.
    5. Si lo anterior también resulta en empate, todo indica que se trata de una molécula simétrica, por lo que no importa qué metal se escoja.

Se esboza a continuación un pseudo código en lenguaje natural que muestra el flujo de ejecución del método principal, *CreateFragCanSmiStringOgm*, durante la canonización.

Algoritmo 1: CreateFragCanSmiStringOgm

if *no tiene metal* || *está fragmentado* **then**
  aplicar algoritmo original
end
startAtom ⟵ Seleccionar el metal inicial
canonicalLabels ⟵ Calcular los *canonical labels*
root ⟵ Generar el nodo raíz(starAtom)
BuildCanonTree(*root*, *canonicalLabels*)
RearrangeTree(*root*)
IdentifyBranches(*root*)
ToCansmilesString(*root*, *buffer*, *canonicalLabels*)
delete root

Mencionar del código anterior el método *RearrangeTree*. Una vez se monta el árbol a través del método *BuildCanonTree* usando el orden canónico calculado por OpenBabel, se realiza un post-procesado a dicho árbol para reordenar cada uno de los hijos en función de la longitud de sus subárboles, de menor a mayor. Esto se ejemplifica con la Figura 4.3. Partimos del SMILES

O#C[Fe+2]1234([I-])(C#O)[CH]=5[CH]4=[CH]3[CH-]2[CH]51

que contiene un átomo de hierro central y 4 ramas: dos enlaces CO, un yodo, y un Cp. El primer árbol que se genera es el de la imagen izquierda, en donde las ramas hijas parecen no seguir ningún orden. El árbol de la derecha es el resultado de reordenar los hijos, colocando las ramificaciones de menor tamaño las primeras. De esta manera cuando el método *ToCansmilesString* recorra el árbol para generar el SMILES canónico, aparecerán antes. Podríamos decir que *BuildCanonTree* se encarga de colocar los hijos verticalmente, asegurando una correcta relación padre-hijo entre los átomos; mientras que *RearrangeTree* los ordena horizontalmente.

En la Figura 4.4 se muestra el caso para el SMILES

[Cl-][Au+][P](C=1C=CC=CC1)(C=2C=CC=CC2)
[C-]34[CH]5=[CH]6[CH]7=[CH]3
[Fe+2]6789%10%1154[CH]=%12[CH]%11=[CH]%10
[C-]9([CH]%128)[P](C)C

el cual tiene 2 metales: un átomo de hierro (Fe) y un átomo de oro (Au). Vemos los 2 árboles generados durante el proceso de selección: en el

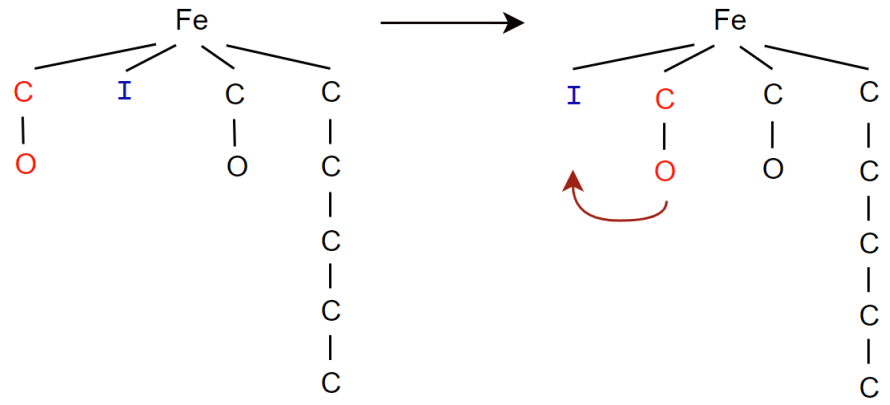
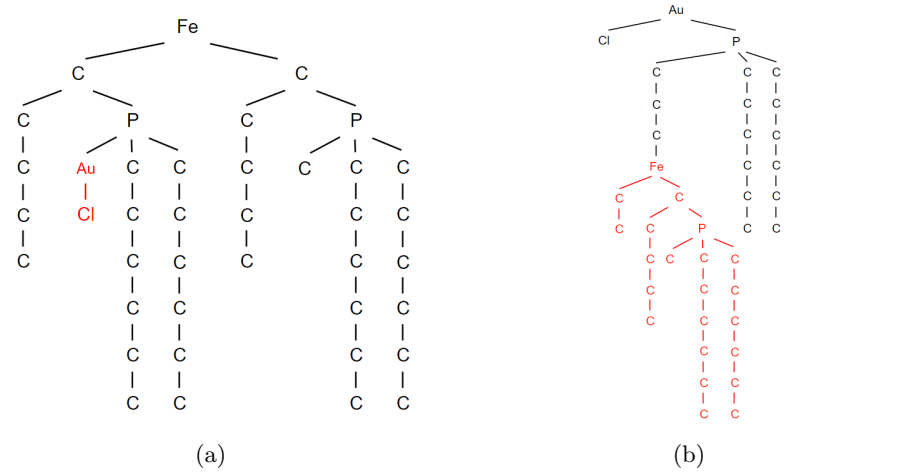


Figura 4.3: Reordenación de los hijos en función de la longitud de sus ramas. Imagen de elaboración propia. Molécula 28 del Anexo **??**.

de la izquierda el hierro se coloca como nodo raíz y se evalúa el subárbol del oro, obteniendo 2 nodos (marcados en rojo); en el de la derecha el oro se coloca como nodo raíz y se evalúa el subárbol del hierro, obteniendo 22 nodos. Se escogerá por tanto el hierro como nodo raíz al tener más nodos que dependen de él. Notar que estos no son los árboles canónicos finales, sino árboles auxiliares cuyo único propósito es seleccionar el primer átomo. Observar también que a ninguno de los dos árboles se le ha aplicado el reordenamiento comentado antes.



(a) (b)

Figura 4.4: Esquemas de los árboles generados para la selección del metal. Imagen de elaboración propia. **(a)** árbol con el hierro como raíz, el subárbol del oro solamente tiene 2 nodos; **(b)** árbol con el oro como raíz, el subárbol del hierro tiene 22 nodos. Imagen de elaboración propia.

4.2.2. Sistema de representación 2D

Al generar representaciones 2D de una molécula pueden surgir muchas dificultades relacionadas con la disposición de los átomos como su orientación o intentar evitar el solapamiento; y relacionadas con el texto, tipo y tamaño de letra, uso o no de abreviaturas, alineación y posición relativa de las etiquetas con respecto a los demás átomos, etc. Estos problemas se pueden superan de mejor o peor manera mediante una serie de algoritmos, pero por ahora, ninguno es tan versátil como para ajustarse a cualquier tipo de estructura química [34]. Para más detalles y ejemplos de algoritmos de representación 2D y sus limitaciones en diferentes toolkits (RDKit, OpenBabel, CDK, Avalon e Indigo), se recomienda ver la presentación de 2016 de John Mayfield [56].

En lo que respecta a este proyecto, se han llevado a cabo 2 modificaciones en el sistema de visualización de OpenBabel para representar moléculas:

1. Se ha aumentado la longitud de los enlaces entre átomos para una mayor claridad y separación. Para moléculas organometálicas, en donde un mismo átomo normalmente tiene varios enlaces y existen muchos ciclos, el mero hecho de espaciar un poco los enlaces ya ayuda a que no se vea todo tan solapado.

2. Se han centrado los esfuerzos en la detección y mejora de visualización de estructuras tipo Cp, muy comunes en organometálica (ver Sección 2.2.3).

Para el segundo punto, se han visualizado una gran cantidad de moléculas que contienen Cp para averiguar las maneras más frecuentes en las que se suelen describir y visualizar. Exsiten varias maneras de representar compuestos de coordinación con ligandos Cp usando enlaces convencionales (Figura 4.5), pero ninguna opción refleja correctamente las propiedades de este tipo de uniones.

Se han valorado otras opciones ya existentes en la literatura, como lo que propone Alex M. Clark en la publicación *"Accurate Specification of Molecular Structures: The Case for Zero-Order Bonds and Explicit Hydrogen Counting"* [19]. Se puede ver en la Figura 4.6 el resultado de su propuesta usando lo que él llama enlaces de orden cero (zero-order bonds).

Aun así, no se diferencia mucho de la Figura 4.5(c) y tampoco termina de ajustarse a cómo los químicos suelen representarlos. Se ha intentado seguir por tanto el manual de la IUPAC *"Graphical representation standards for chemical structure diagrams (IUPAC Recommendations 2008)"* [16], que alberga una gran cantidad de casuísticas con todos los aspectos relacionados a la representación gráfica de la estructura molecular, explicadas de manera

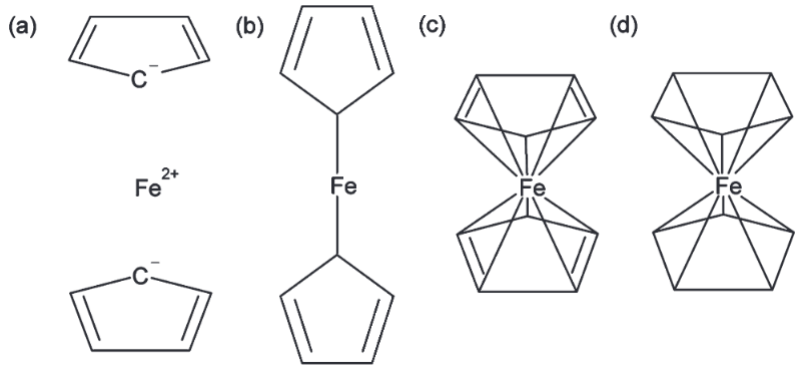


Figura 4.5: Intentos para representar el ferroceno usando enlaces convencionales. **(a)** ni si quiera representa los enlaces, dibujando varios fragmentos desconectados; **(b)** mantiene intactos los recuentos de valencia orgánica, desconectando los carbono de enlaces dobles; **(c)** Ignora las restricciones de valencia normales e incluye todos los enlaces átomo-átomo posibles; **(d)** representa todos los enlaces significativos a costa de los dobles enlaces. Imagen y descripción extraída y traducida de [19]

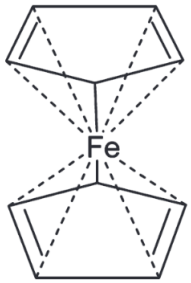


Figura 4.6: Ferroceno utilizando enlaces de orden cero según Alex M. Clark. Imagen extraída de [19].

detallada y rigurosa. Concretamente en las secciones GR-1.7.1 y GR-6 del manual habla sobre los enlaces de coordinación y enlaces de ciclos aromáticos y electrones deslocalizados. Con lo anterior y atendiendo a las recomendaciones de los expertos, el objetivo es implementar la forma mostrada en la Figura 2.7(a) ya que capta la aromaticidad y geometría de los enlaces del ligando.

A todo lo anterior se nos suma el concepto de percepción de anillos, una problemática para nada trivial y que ha estado siempre presente en en mundo de las chemoinformatics. Para entender esto, imaginemos que tenemos una molécula con un anillo de *decalina* (10 carbonos). A la hora de identificar y almacenar la pertenencia a ciclos de cada uno de los átomos no podemos simplemente utilizar todas las combinaciones de ciclos existentes (Figura

4.7), ya que haría que un conjunto de átomos pertenezcan simultáneamente a varios anillos (en este caso los átomos 4 y 5). Ejemplo extraído de [33]. Esto según las necesidades de cada aplicación puede suponer un problema o no.

Figura 4.7: Decalina y todo su subset de ciclos.

Por lo general, los algoritmos en los toolkits de chemoinformatics requieren un set de ciclos filtrado, únicamente con los ciclos más pequeños y relevantes. Para el caso anterior, bastaría si nos quedáramos solamente con cualquier combinación de 2 de los 3 ciclos, teniendo así todo el espacio de ciclos cubierto. Esto se puede alcanzar con el algoritmo de SSSR (Smallest Set of Smallest Rings), siendo el que más se usa hoy día para la percepción de anillos [33, 43, 11, 7].

La percepción SSSR es la que utiliza OpenBabel y por lo general funciona bien para cualquier tipo de molécula. Pero justamente a la hora de representar estructuras Cp, la forma en la que los átomos de carbono se unen con el metal hace que se genere un conjunto de ciclos que dificulta los algoritmos de detección. Se ilustra este problema con la Figura 4.8. Supongamos que un hierro (Fe) se une con un ligando Cp. Vemos que cada par de carbonos forma un ciclo de tamaño 3 junto con el hierro, desapareciendo el ciclo original entre los propios carbonos, de manera que internamente tenemos los ciclos almacenados como "3-1-2, 4-1-3, 5-1-6, 6-1-2 y 5-1-4". Por defecto OpenBabel trata los enlaces como en la Figura 4.5(c) y teniendo en cuenta el tratamiento de los ciclos, en la mayoría de casos obtenemos resultados como los de la Figura 4.9, ya que a la hora de colocar las líneas de los enlaces, los toma como anillos independientes.

Reglas de detección

Además, hay que tener en cuenta otras situaciones a la hora de detectar Cps completos. Si en la misma molécula hay más de un Cp, cómo se detectan y diferencian cada uno de las estructuras o cómo sabemos qué carbonos forman parte de un Cp o de otro. A través de la cadena SMILES no se puede obtener esa información ya que todos los enlaces son iguales (un metal con un carbono) y la percepción SSSR complicaba más aun la identificación de pertenencia de los carbonos al mismo anillo. En base a esto, surge la identificación por bloques. Se utiliza para esto el árbol canónico

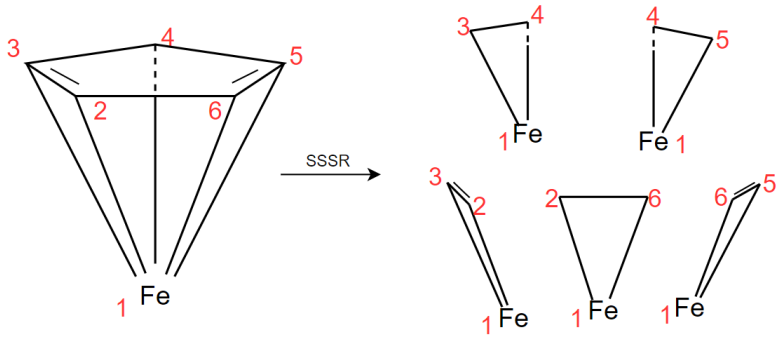


Figura 4.8: Conjunto de ciclos según la percepción SSSR para un hierro con un ligando Cp. Imagen de elaboración propia.

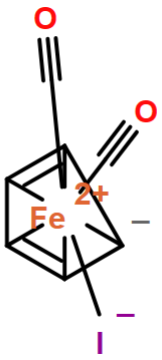


Figura 4.9: Representación 2D de la molécula *Dicarbonylcyclopentadienyliodoiron(II)* generada con OpenBabel. Molécula 28 del Anexo **??**.

explicado previamente con el objetivo identificar relaciones padre-hijo o relaciones de dependencia entre átomos, indicando la pertenencia a una misma subestructura, en este caso a un mismo Cp.

Por tanto, las reglas de detección para una estructura Cp se pueden resumir en:

- Que exista un enlace Metal-Carbono (M-C)
- Que el carbono pertenezca a un bloque y que los demás carbonos del mismo bloque tengan también un enlace M-C con el mismo metal que el resto del bloque.

Una vez se identifiquen correctamente, se activa un flag especial para marcar qué átomos pertenecen o no a un Cp y darles un tratamiento distinto a la hora de dibujarlos. En concreto:

1. Se crea un único átomo nuevo a modo de señuelo (átomo especial en

OpenBabel con número atómico 0, *dummy*) para que enlace con el metal.

2. En base a ese nuevo átomo dummy, se modifican las coordenadas 2D de los carbonos de interés y se disponen en forma de polígono.

3. Se calcula el círculo central mediante una serie de coordenadas y tanto al círculo como al polígono se le da una perspectiva rotándolos sobre el eje X.

4. Finalmente se eliminan todos los enlaces M-C para que OpenBabel no los dibuje automáticamente. El hecho de eliminar los enlaces no supone ningún problema ya que no se realizan más operaciones a posteriori que puedan necesitarlos.

Todo lo relacionado con cada Cp individual se almacena en la clase CpComplex, guardando información como la lista de carbonos que forman el Cp, el metal al que están asociados, el átomo dummy que lo posiciona y el centroide del polígono entre otras cosas.

En el siguiente capítulo se muestran los resultados alcanzados tras la implementación.

Capítulo 5

Resultados y pruebas

En las siguientes secciones se expondrán los resultados que se han obtenido tras usar la nueva nomenclatura implementada y los cambios en el sistema de representación. Se describirán también las pruebas llevadas a cabo para validar todo este proceso.

5.1. Nomenclatura canónica

- Poner una tabla con las moléculas: smiles original, dibujo original, smiles canonico, dibujo canonico (a ser posible mejorado)

Para los resultados de la canonizacion tendré que hacer una tabla comparando ambas cadenas o algo asi

Con esta solución en donde se organiza el SMILES según sus ramificaciones, se ha intentado también dar un enfoque al canonizado orientado a la representación 2D. De esta manera, podemos visualizar ambos e identificar claramente la correspondencia entre una porción del SMILES con una sección de la representación. Así, se recorren primero todos los vecinos de un átomo hasta cerrar esa rama antes de pasar a la siguiente, obteniendo un SMILES más limpio. (ilustrar esto con una imagen que me invente yo). Se ilustra esto en las siguientes Figuras 5.1. auqi puedo usar la mol 24, la 23, o la 5 simetrica, o alguna de las ultimas de iridio.

5.2. Sistema de representación 2D

PH(aqui tengo que mostrar 100 % la evolución de 2 o 3 para el ironII, para ilustrar este proceso de pensado que explico. Luego ya en resultados muestro todas en sus versiones finales).

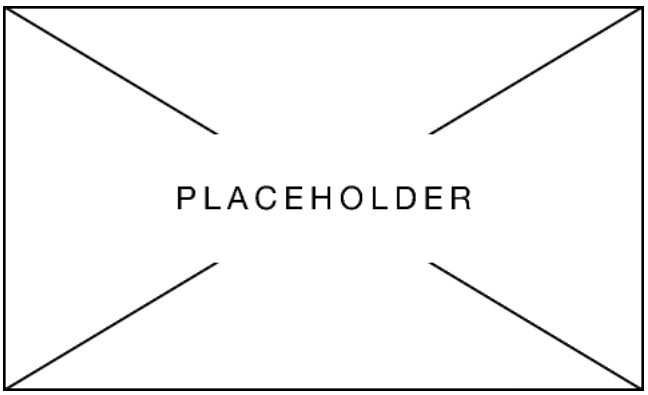


Figura 5.1: Correspondencia entre el SMILES canónico y la representación 2D de la molécula.

Los ficheros svg completos con el antes y el después para el conjunto de datos que he usado están disponibles para su consulta en GitHub.

Meter una evolución de 3 fotos del progreso del dibujado. Y poner algo tipo "Se han alcanzado varios modos de dibujado pero el experto finalmente ha preferido el X, por ser el más comodo y entendible entre químicos y ser más utilizado en la literatura" (siguiendo además el manual mencioando en la seccion 4.2.2 anteriormente). Puedo tener un par de moléculas de juguete/referencia en donde voy mostrando los cambios gordos que voy aplicando; y luego la tabla grande con todo el dataset (cuando ejecuto todas juntas) como para mostrar el "antes mal -¿ahora bien". Aquellas que tanto antes como después las dibuje bien o igual, puedo poner una o un par y listo (no todas).

(siguiendo el este estándar de la iupac mencionado en la Sección 4.2.2, las líneas gordas son preferibles. Pero para moléculas de este tipo donde todo sale muy pegado, y según las recomendaciones de los expertos es mejor que salga todo un poco más espaciado. Si bien esto no arregla el solapamiento (esto requiere de un cambio sustancial en el algoritmo de generación de coordenadas), mejora minimante la visibilidad de los enlaces)

Poner tambien las versiones sin alias y con alias. "version sin alias vs version con alias, puede quedar más legible", "la visualización se puede mejorar aun, utilizando la opción de generación de abreviaturas de OpenBabel", con los parámetros –genalias -xA en la línea de ordenes, podemos invocar esta funcionalidad. Esto consiste en "... blah blah, fichero de base de datos, equivalencias, no se que"

Mostrar tb el dubujo del rutenio ultimo, para mostrar que se detectan no solo Cp de 5, sino de más carbonos tambien.

5.3. Pruebas

El propio proyecto de OpenBabel ya trae consigo un entorno de pruebas, que se gestiona a través del ejecutable 'test_runner.exe'. Según los parámetros que le pases al programa, se ejecutará un test u otro. Dichos tests pueden incluso albergar otros tests más específicos dentro. La ejecución de un test concreto sigue este esquema:

test_runner <nombreTest><numeroTest>

Un ejemplo de llamada al test "conversion", que es un test único sería: *test_runner.exe conversion*. Con la llamada *test_runner.exe regressiontest 225* se estaría ejecutando el test número 225 especificado en regressiontest.cpp. A la hora de añadir nuevos ficheros de testing hay que seguir una serie de indicaciones y utilizar una sintaxis especial para determinar si el test es correcto o no. Esto se describe en la guía para desarrolladores de la página oficial de OpenBabel[1,2] (en el Anexo A se describe como añadir nuevos tests).

Dicho esto, la mayoría de tests que se han realizado son tests funcionales completos. Hacer un test de unidad de según que métodos es complicado ya que necesitan una serie de parámetros y variables que se van creando sobre la marcha en multitud de métodos previos. El fichero de pruebas creado específicamente para el proyecto se llama "canonogmtest.cpp" y tiene la siguiente estructura: **METER un salto de pagina por si queda mal del dirtree al acabar todo**

- `test_runner.cpp` ..... fichero main gestor de pruebas
  - `canonogmtest.cpp` ..... fichero main de pruebas para el proyecto
    - `SelectCanonMetal` ..... prueba 1
    - `RandomCanonStandardLabels` ..... prueba 2
    - `RandomCanonCanonicalLabels` ..... prueba 3
    - `CanonOverCanon` ..... prueba 4
    - `DrawDoubleCpTest` ..... prueba 5
  - `Otros .cpp de pruebas creados por OpenBabel`
    - `regressiontest.cpp`
    - `ringtest.cpp`
    - `invalidsmiles.cpp`
    - `conversion.cpp`
    - ...
    - ...

En el GitHub del proyecto se pueden consultar los ficheros de salida

[1] `https://openbabel.org/docs/current/Contributing/Testing.html`
[2] `https://open-babel.readthedocs.io/en/latest/Contributing/Testing.html`

que resultaron de la ejecución de las pruebas (las que tengan alguna salida).**AÑADIR ESTO A GITHUB** Este proceso de testing, tal como indico en la Sección 3.1.3, se realizaba de manera frecuente para comprobar si los resultados iban siendo los esperados. A continuación se describen las pruebas llevadas a cabo:

- **Test unitario**
  - **SelectCanonMetal**: comprueba que el método *OBMol2Cansmi::SelectRootAtomOgm* funciona correctamente eligiendo el átomo adecuado de entre todos los átomos para moléculas que contengan algún metal. Los SMILES a comprobar se cargan de un fichero, junto con el identificador del átomo que se debería seleccionar. El test es válido si el identificador cargado y el que devuelve el método coinciden.
- **Tests funcionales**
  - **RandomCanonStandardLabels**: comprueba la consistencia de la nomenclatura canónica para la forma estándar. Se toman los SMILES de prueba desde un fichero y se obtienen unos SMILES sinónimos de manera aleatoria para la misma molécula. Tanto el SMILES original como el randomizado se canonizan a su forma estándar. El test es válido si ambos canónicos obtenidos coinciden. Por lo general, teniendo en cuenta lo discutido en la Sección 4.2.1 es normal que el test falle.
  - **RandomCanonCanonicalLabels**: comprueba la consistencia de la nomenclatura canónica para la forma canónica. Se toman los SMILES de prueba desde un fichero y se obtienen unos SMILES sinónimos de manera aleatoria para la misma molécula. Tanto el SMILES original como el randomizado se canonizan a su forma canónica. El test es válido si ambos canónicos obtenidos coinciden.
  - **CanonOverCanon**: comprueba la consistencia de la nomenclatura canónica. Se toman los SMILES de prueba desde un fichero y se canonizan. A esos SMILES canonizados, se les vuelve a aplicar el algoritmo. Con este test se quiere verificar que sucesivas aplicaciones del algoritmo no degeneren el resultado o haya una vaivén de los átomos, como sería el caso de la Figura 5.2. El test es válido si ambos resultados coinciden.
  - **DrawDoubleCpTest**: comprueba si los algoritmos de dibujado para moléculas con múltiples Cp se aplican correctamente. Se introduce una molécula sabida de antemano que contiene varias de estas estructuras y se realiza la conversión. El test en sí tiene es

Original	[Cl-][Pd+2]1([C-]=2C=CC=CC2C=3C=CC=CC3[N]1(C)C)[PH](C4CC5CCC4C5)C6CC7CCC6C7
Canonizado	[Pd+2]1([Cl-])([N](C)(C)c2ccccc2C2=[C-]1C=CC=C2)[PH](C1C2CCC(C1)C2)C1C2CCC(C1)C2
Canonizado x2	[Pd+2]1([Cl-])([N](C)(C)c2c(cccc2)C2=[C-]1C=CC=C2)[PH](C1CC2CCC1C2)C1CC2CCC1C2
Canonizado x3	[Pd+2]1([Cl-])([N](C)(C)c2ccccc2C2=[C-]1C=CC=C2)[PH](C1C2CCC(C1)C2)C1C2CCC(C1)C2

Figura 5.2: Ejemplo de una canonización imperfecta. Se muestra el SMILES original, y los SMILES canónicos tras haber aplicado 1, 2 y 3 veces el algoritmo sobre el anterior resultado respectivamente. Esto no ocurre con ninguna de las moléculas probadas.

válido si el proceso de conversión tiene éxito. Dado que analizar de manera automática el resultado de una representación gráfica en formato .svg no es tarea fácil, hay que verificar si el resultado es el esperado de acuerdo a lo descrito en la Sección 4.2.2 de manera manual.

Se han ejecutado también otros tests útiles como el *invalidsmiles* para comprobar que OpenBabel rechazaba SMILES incorrectos. Decir que para los tests *RandomCanonStandardLabels*, *RandomCanonCanonicalLabels* y *CanonOverCanon* se han usado datasets relativamente grandes, de aproximadamente 1000 moléculas incluyendo química orgánica general y las moléculas nuestras propias de organometálica. Para todas ellas, salvo 6 excepciones de estereoquímica con compuestos cis/trans, el test se cumple. Las otras dos pruebas son mucho más selectivas y específicas, por lo que se han aplicado a nuestro dataset simplemente.

Capítulo 6

Conclusiones y trabajos futuros

Blah blah **introduccion**

6.1. Conclusiones

blah blah **comparar los objetivos iniciales con los resultados obtenidos y cómo he alcanzado dichos objetivos**

En la actualidad, como exponía **en la Sección 2, cada software** Se pretende por tanto con este trabajo, hacer una propuesta con el objetivo de resolver los conflictos entre las distintas bases de datos, usando una nueva nomenclatura canónica para moléculas organometálicas y mejoras en su dibujado. Se aspira que en algún momento los cambios planteados se agreguen al software oficial, se desplieguen en un futuro release, y con el tiempo se extienda su uso y sea útil para los que utilizan esta herramienta. **esto no se si quitarlo de aqui y simplemente decirlo en la presentacion** De hecho, ya se ha contactado con los administradores del repositorio oficial de GitHub para una posible contribución, quedando a la espera de la revisión del código y una respuesta por su parte.

En química, hay una serie de reglas establecidas y la mayoría de moléculas convencionales se ajustan a ellas. Existen también muchas excepciones y particularidades, macromoléculas, proteínas, o compuestos de coordinación y organometálicos, que se rigen por sus propias normas. En general, creemos se han alcanzado resultados satisfactorios para el conjunto de datos con el que se ha trabajado. Es un conjunto de datos relativamente pequeño, pero se ajusta bien al alcance de un proyecto de estas características.

Realmente el problema que estamos tratando aqui no son los algoritmos

en sí, si no que todo el mundo utilice el mismo algoritmo de generacion canonico. OpenSmiles: Çanonical SMILES should not be considered a universal, global identifier (such as a permanent name that spans the WWW). Two systems that produces a canonical SMILES may use different rules in their code, or the same system may be improved or have bugs fixed as time passes, thus changing the SMILES it produces. A Canonical SMILES is primarily useful in a single database, or a system of related databases or information, in which all molecules were created using a single canonicalizer."

6.2. Trabajos futuros

Extender el estudio y realizar pruebas con un conjunto de moléculas mayor, adaptando poco a poco el sistema de dibujado para que se ajuste a las excepciones.

un punto a mejorar muy bueno sería el tema de la estereoquímica en la representación de las moléculas. A priori puede no parecer muy importante que se dibuje una línea plana o con cierta geometría (triangular, tetraédrica, cuadrada plana, octaédrica, etc), pero en ciertas áreas de la medicina y la bioquímica que trabajan con encimas y pequeñas proteínas, un determinado fármaco según su geometría o isomería puede tener efectos completamente distintos. blah balh

para la parte de canonizacion, ahora mismo, está pensado para organometálica simplemente, la cosa sería ampliar la canonización para abarcar otras áreas de la química (poner alguna palabras complicada de estas, estereoquimica, etc)

Bibliografía

[1] R. S. Cahn, Christopher Ingold y V. Prelog. "Specification of Molecular Chirality". en. En: *Angewandte Chemie International Edition in English* 5.4 (1966), págs. 385-415. ISSN: 1521-3773. DOI: 10.1002/anie.196603851. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/anie.196603851.

[2] William J. Wiswesser. "107 Years of Line-Formula Notations (1861-1968)". en. En: *Journal of Chemical Documentation* 8.3 (ago. de 1968), págs. 146-150. ISSN: 0021-9576, 1541-5732. DOI: 10.1021/c160030a007. URL: https://pubs.acs.org/doi/abs/10.1021/c160030a007.

[3] Clemens Jochum y Johann Gasteiger. "Canonical Numbering and Constitutional Symmetry". en. En: *Journal of Chemical Information and Computer Sciences* 17.2 (mayo de 1977), págs. 113-117. ISSN: 0095-2338, 1520-5142. DOI: 10.1021/ci60010a014. URL: https://pubs.acs.org/doi/abs/10.1021/ci60010a014.

[4] Reiner Luckenbach. "The Beilstein Handbook of Organic Chemistry: the first hundred years". en. En: *Journal of Chemical Information and Computer Sciences* 21.2 (mayo de 1981), págs. 82-83. ISSN: 0095-2338. DOI: 10.1021/ci00030a006. URL: https://pubs.acs.org/doi/abs/10.1021/ci00030a006.

[5] Vladlmir Prelog y Günter Helmchen. "Basic Principles of the CIP-System and Proposals for a Revision". en. En: *Angewandte Chemie International Edition in English* 21.8 (1982), págs. 567-583. ISSN: 1521-3773. DOI: 10.1002/anie.198205671. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/anie.198205671.

[6] David Weininger. "SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules". en. En: *Journal of Chemical Information and Modeling* 28.1 (feb. de 1988), págs. 31-36. ISSN: 1549-9596. DOI: 10.1021/ci00057a005. URL: https://pubs.acs.org/doi/abs/10.1021/ci00057a005 (visitado 20-11-2022).

[7] Geoffrey M. Downs et al. "Review of ring perception algorithms for chemical graphs". en. En: *Journal of Chemical Information and Com-*

puter Sciences 29.3 (ago. de 1989), págs. 172-187. ISSN: 0095-2338, 1520-5142. DOI: 10.1021/ci00063a007. URL: https://pubs.acs.org/doi/abs/10.1021/ci00063a007.

[8] David Weininger, Arthur Weininger y Joseph L. Weininger. "SMILES. 2. Algorithm for generation of unique SMILES notation". en. En: *Journal of Chemical Information and Computer Sciences* 29.2 (mayo de 1989), págs. 97-101. ISSN: 0095-2338. DOI: 10.1021/ci00062a008. URL: https://pubs.acs.org/doi/abs/10.1021/ci00062a008 (visitado 20-11-2022).

[9] Gabino A. Carriedo Ule José Daniel Miguel San y Daniel Miguel San José. *Curso de iniciación a la química organometálica*. es. Google-Books-ID: 1YH85bieBt0C. Universidad de Oviedo, 1995. ISBN: 978-84-7468-873-3.

[10] Eduardo Peris Fajarnés. *Química organometálica de los metales de transición*. es. Google-Books-ID: AhSihif_h_kC. Publicacions de la Universitat Jaume I, 1997. ISBN: 978-84-8021-134-5.

[11] Franziska Berger et al. "Counterexamples in Chemical Ring Perception". En: *Journal of Chemical Information and Computer Sciences* 44.2 (mar. de 2004). Publisher: American Chemical Society, págs. 323-331. ISSN: 0095-2338. DOI: 10.1021/ci030405d. URL: https://doi.org/10.1021/ci030405d.

[12] Johann Gasteiger y Thomas Engel. *Chemoinformatics: A Textbook*. en. John Wiley & Sons, dic. de 2006. ISBN: 978-3-527-60650-4.

[13] Mike Cohn. *Differences Between Scrum and Extreme Programming*. en. Abr. de 2007. URL: https://www.mountaingoatsoftware.com/blog/differences-between-scrum-and-extreme-programming (visitado 14-05-2023).

[14] Andrew R. Leach y V. J. Gillet. *An Introduction to Chemoinformatics*. en. Google-Books-ID: 4z7Q87HgBdwC. Springer, sep. de 2007. ISBN: 978-1-4020-6291-9.

[15] Vincent Artero y Marc Fontecave. "Hydrogen evolution catalyzed by {CpFe(CO)2}-based complexes". en. En: *Comptes Rendus Chimie* 11.8 (ago. de 2008), págs. 926-931. ISSN: 1631-0748. DOI: 10.1016/j.crci.2008.03.006. URL: https://www.sciencedirect.com/science/article/pii/S1631074808000635.

[16] Jonathan Brecher. "Graphical representation standards for chemical structure diagrams (IUPAC Recommendations 2008)". en. En: *Pure and Applied Chemistry* 80.2 (ene. de 2008), págs. 277-410. ISSN: 1365-3075, 0033-4545. DOI: 10.1351/pac200880020277. URL: https://www.degruyter.com/document/doi/10.1351/pac200880020277/html.

[17] Johann Gasteiger. "The Scope of Chemoinformatics". en. En: *Handbook of Chemoinformatics*. Ed. por Johann Gasteiger. Weinheim, Germany: Wiley-VCH Verlag GmbH, mayo de 2008, págs. 3-5. ISBN: 978-3-527-61827-9 978-3-527-30680-0. DOI: 10.1002/9783527618279.ch1. URL: https://onlinelibrary.wiley.com/doi/10.1002/9783527618279.ch1.

[18] Nathan Brown. "Chemoinformatics—an introduction for computer scientists". en. En: *ACM Computing Surveys* 41.2 (feb. de 2009), págs. 1-38. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/1459352.1459353. URL: https://dl.acm.org/doi/10.1145/1459352.1459353.

[19] Alex M. Clark. "Accurate Specification of Molecular Structures: The Case for Zero-Order Bonds and Explicit Hydrogen Counting". En: *Journal of Chemical Information and Modeling* 51.12 (dic. de 2011). Publisher: American Chemical Society, págs. 3149-3157. ISSN: 1549-9596. DOI: 10.1021/ci200488k. URL: https://doi.org/10.1021/ci200488k.

[20] Noel M O'Boyle. "Towards a Universal SMILES representation - A standard method to generate canonical SMILES based on the InChI". En: *Journal of Cheminformatics* 4.1 (dic. de 2012), pág. 22. ISSN: 1758-2946. DOI: 10.1186/1758-2946-4-22. URL: https://jcheminf.biomedcentral.com/articles/10.1186/1758-2946-4-22.

[21] Stephen Heller et al. "InChI - the worldwide chemical structure identifier standard". En: *Journal of Cheminformatics* 5.1 (ene. de 2013), pág. 7. ISSN: 1758-2946. DOI: 10.1186/1758-2946-5-7. URL: https://doi.org/10.1186/1758-2946-5-7.

[22] *Lewis Structures*. en. Oct. de 2013. URL: https://chem.libretexts.org/Bookshelves/Physical_and_Theoretical_Chemistry_Textbook_Maps/Supplemental_Modules_(Physical_and_Theoretical_Chemistry)/Physical_Properties_of_Matter/Atomic_and_Molecular_Properties/Lewis_Structures (visitado 08-03-2023).

[23] *Organometallic Chemistry (Evans)*. en. Oct. de 2013. URL: https://chem.libretexts.org/Bookshelves/Inorganic_Chemistry/Organometallic_Chemistry_(Evans) (visitado 13-03-2023).

[24] *Valence Bond Theory*. en. Oct. de 2013. URL: https://chem.libretexts.org/Bookshelves/Physical_and_Theoretical_Chemistry_Textbook_Maps/Supplemental_Modules_(Physical_and_Theoretical_Chemistry)/Chemical_Bonding/Valence_Bond_Theory (visitado 26-02-2023).

[25] M. Despa. "Comparative study on software development methodologies". En: *Database Systems Journal* (dic. de 2014). URL: https://www.semanticscholar.org/paper/Comparative-study-on-software-development-Despa/28d32b9375e9125624ec614a2cec85c9f0716b13.

[26] Stephen R. Heller et al. "InChI, the IUPAC International Chemical Identifier". En: *Journal of Cheminformatics* 7.1 (mayo de 2015), pág. 23. ISSN: 1758-2946. DOI: 10.1186/s13321-015-0068-4. URL: https://doi.org/10.1186/s13321-015-0068-4.

[27] Thomas Engel y Johann Gasteiger. *Applied Chemoinformatics. Achievements and Future Opportunities*. en. Wiley-VCH, 2018. ISBN: 978-3-527-34201-3.

[28] *Lewis Structures and the Octet Rule*. en. Jul. de 2018. URL: https://chem.libretexts.org/Courses/Bellarmine_University/BU%5C%3A_Chem_103_(Christianson)/Phase_3%5C%3A_Atoms_and_Molecules_-_the_Underlying_Reality/10%5C%3A_Molecular_Structure_and_Geometry/10.1%5C%3A_Lewis_Structures_and_the_Octet_Rule (visitado 08-03-2023).

[29] Juan Camilo Salazar et al. "Scrum versus XP: similitudes y diferencias". es. En: *Tecnología Investigación y Academia* 6.2 (dic. de 2018), págs. 29-37. ISSN: 2344-8288. URL: https://revistas.udistrital.edu.co/index.php/tia/article/view/10496.

[30] *2.4: Line Notation*. en. Jun. de 2019. URL: https://chem.libretexts.org/Courses/Intercollegiate_Courses/Cheminformatics/02%3A_Representing_Small_Molecules_on_Computers/2.04%3A_Line_Notation.

[31] Richard L. Apodaca. *Computing Extended Connectivity Fingerprints*. en. Ene. de 2019. URL: http://depth-first.com/articles/2019/01/11/extended-connectivity-fingerprints/.

[32] Flaviu Fuior. "Key elements for the success of the most popular Agile methods". En: *Revista Română de Informatică şi Automatică* 29 (dic. de 2019), págs. 7-16. DOI: 10.33436/v29i4y201901.

[33] Richard L. Apodaca. *A Smallest Set of Smallest Rings*. en. Ago. de 2020. URL: http://depth-first.com/articles/2020/08/31/a-smallest-set-of-smallest-rings/ (visitado 17-04-2023).

[34] Laurianne David et al. "Molecular representations in AI-driven drug discovery: a review and practical guide". En: *Journal of Cheminformatics* 12.1 (sep. de 2020), pág. 56. ISSN: 1758-2946. DOI: 10.1186/s13321-020-00460-5. URL: https://doi.org/10.1186/s13321-020-00460-5.

[35] Mario Krenn et al. "Self-referencing embedded strings (SELFIES): A 100 % robust molecular string representation". en. En: *Machine Learning: Science and Technology* 1.4 (dic. de 2020), pág. 045024. ISSN: 2632-2153. DOI: 10.1088/2632-2153/aba947. URL: https://iopscience.iop.org/article/10.1088/2632-2153/aba947.

[36] Digital.ai. *15th State of Agile Report*. 2021.

[37] Alok Mishra et al. "Organizational issues in embracing Agile methods: an empirical assessment". En: *International Journal of System Assurance Engineering and Management* 12 (oct. de 2021). DOI: 10.1007/s13198-021-01350-1.

[38] Rebeca Nayely Osorio-Yáñez y David Morales Morales. "Compuestos organometálicos y de coordinación: Más que sólo una buena relación de metales de transición y moléculas orgánicas: Organometallic and coordination compounds: More than just a happy relationship between transition metals and organic molecules". es. En: *TECNOCIENCIA Chihuahua* 15.3 (dic. de 2021), págs. 261-276. ISSN: 1870-6606. DOI: 10.54167/tecnociencia.v15i3.855. URL: https://vocero.uach.mx/index.php/tecnociencia/article/view/855.

[39] *10.1: Antecedentes históricos e introducción a los metalocenos*. es. Oct. de 2022. URL: https://espanol.libretexts.org/Quimica/Qu%5C%C3%5C%ADmica_Inorg%5C%C3%5C%A1nica/Qu%5C%C3%5C%ADmica_de_Coordinaci%5C%C3%5C%B3n_Inorg%5C%C3%5C%A1nica_(Landskron)/10%5C%3A_Qu%5C%C3%5C%ADmica_Organomet%5C%C3%5C%A1lica/10.01%5C%3A_Antecedentes_hist%5C%C3%5C%B3ricos_e_introducci%5C%C3%5C%B3n_a_los_metalocenos.

[40] Kevin Darza. *¿Cómo calcular la depreciación de un equipo de cómputo?* Ago. de 2022. URL: https://www.oliversoft.mx/depreciacion-de-un-equipo-de-computo/ (visitado 25-05-2023).

[41] Mario Krenn et al. "SELFIES and the future of molecular string representations". en. En: *Patterns* 3.10 (oct. de 2022). arXiv:2204.00056 [physics], pág. 100588. ISSN: 26663899. DOI: 10.1016/j.patter.2022.100588. URL: http://arxiv.org/abs/2204.00056 (visitado 20-11-2022).

[42] Alston Lo et al. *Recent advances in the Self-Referencing Embedding Strings (SELFIES) library*. arXiv:2302.03620 [physics]. Feb. de 2023. DOI: 10.48550/arXiv.2302.03620. URL: http://arxiv.org/abs/2302.03620.

[43] *12.7 Smallest Set of Smallest Rings (SSSR) considered Harmful*. URL: https://www.ics.uci.edu/~dock/manuals/oechem/pyprog/node123.html (visitado 17-04-2023).

[44] baoilleach. *We need to talk about Kekulization, Aromaticity and SMILES*. URL: https://www.slideshare.net/baoilleach/we-need-to-talk-about-kekulization-aromaticity-and-smiles (visitado 11-04-2023).

[45] *ChemDraw — Revvity Signals (Formerly Known as PerkinElmer Informatics)*. URL: https://revvitysignals.com/products/research/chemdraw (visitado 21-03-2023).

[46] *Daylight*. URL: https://www.daylight.com/.

[47] *Extreme Programming: A Gentle Introduction*. URL: http://www.extremeprogramming.org/ (visitado 07-05-2023).

[48] *Google Inc. Google Colaboratory*. URL: https://colab.research.google.com.

[49] ICIQ. *Prof. Mónica H. Pérez-Temprano, Research Group*. en. URL: https://www.iciq.org/research/research_group/dr-monica-h-perez-temprano/section/research_overview/ (visitado 10-03-2023).

[50] Mario Krenn et al. "SELFIES: a robust representation of semantically constrained graphs with an example application in chemistry". en. En: ().

[51] *NOMENCLATURA R/S*. URL: https://www.liceoagb.es/quimiorg/nomenrs.html.

[52] *Open Babel: Canonical Coding Algorithm*. URL: https://openbabel.github.io/api/3.0/canonical_code_algorithm.shtml.

[53] *OpenSMILES specification*. URL: http://opensmiles.org/.

[54] PubChem. *PubChem*. en. URL: https://pubchem.ncbi.nlm.nih.gov (visitado 01-02-2023).

[55] *RDKit Cookbook. RDKit Documentation*. URL: https://www.rdkit.org/docs/Cookbook.html (visitado 15-03-2023).

[56] *RDKit UGM 2016: Higher Quality Chemical Depictions*. URL: https://www.slideshare.net/NextMoveSoftware/rdkit-ugm-2016-higher-quality-chemical-depictions (visitado 10-05-2023).

[57] SciFinder. *SciFinder-help*. en. URL: https://scifinder-n.cas.org/help/#t=Working_with_Search_Results%5C%2FAll_Answer_Types_screen.htm (visitado 01-02-2023).

[58] *Scrum and Extreme Programming (XP) - a perfect hybrid model for software development projects*. es. URL: https://www.linkedin.com/pulse/scrum-extreme-programming-xp-perfect-hybrid-model-raghavan (visitado 08-05-2023).

[59] *SELFIES Derivation — selfies 1.0.0 documentation*. URL: https://selfies.readthedocs.io/en/latest/tutorial.html (visitado 06-03-2023).

[60] SigmaAldrich. *SigmaAldrich - About us*. es. URL: https://www.sigmaaldrich.com/ES/es/life-science/about-us/expertise (visitado 01-02-2023).

[61] *SMILES Tutorial*. URL: https://www.daylight.com/meetings/summerschool98/course/dave/smiles-intro.html (visitado 09-02-2023).

[62] NextMove Software. *A de facto standard or a free-for-all? A benchmark for reading SMILES*. URL: https://www.slideshare.net/NextMoveSoftware/a-de-facto-standard-or-a-freeforall (visitado 28-02-2023).

[63] Chris Swain. *Computation Chemistry Tools*. en. URL: https://www.cambridgemedchemconsulting.com/resources/lead_identification/computational_chemistry_tools.html (visitado 20-03-2023).

[64] Chris Swain. *Safari Extensions*. en. URL: https://www.macinchem.org/extensions/index.php (visitado 20-03-2023).

[65] *The Organometallic HyperTextBook: The OMHTB Reading Room*. URL: http://www.ilpi.com/organomet/readingroom.html (visitado 13-03-2023).

[66] *The RDKit Book*. URL: https://www.rdkit.org/docs/RDKit_Book.html (visitado 15-03-2023).

[67] *Universal Smiles: Finally a canonical SMILES string*. en. URL: https://www.slideshare.net/baoilleach/universal-smiles-finally-a-canonical-smiles-string (visitado 28-05-2023).

[68] Martin Vogt. "Algorithms in Chemoinformatics Canonical Representations and Substructure Searching". en. En: ().

Apéndice A

Manual de usuario

Aquí la idea es describir los procesos de instalacion de las herramientas mas importantes del proyecto, siendo, la instalacion y buildeado local de openbabel, y la instalacion de visual studio c++, y como utilizarlo para añadir nuevos ficheros al proyecto ya existente y ejecutar obabel por linea de comandos para generar resultados.

Describir un poco tb la estructura del prpyecto por soluciones y señalar las que he usado durante el trabajo