—————**ARTICLES**—————

Chemical Descriptors Library (CDL): A Generic, Open Source Software Library for Chemical Informatics

Vladimir J. Sykora* and David E. Leahy

Northern Institute for Cancer Research, Newcastle University, Newcastle Upon Tyne NE1 7RU, U.K.

Received April 16, 2008

In this article the Chemical Descriptors Library (CDL), a generic, open source software library for chemical informatics is introduced. The library is written using standard-compliant C++ programming language. The CDL provides a generic interface for traversing the structure of a molecular graph and accessing its properties. As a result, the software offers flexibility, reusability, and maintainability. This interface has been used to develop several chemical informatics algorithms, including molecular text format parsers and writers; substructure, pharmacophore, and atom type fingerprints; and both common substructure search and SMARTS search. The algorithms are described and evaluated on 3 data sets comprising 1000, 50000, and 100000 small molecules, respectively. The properties of the algorithms in terms of complexity analysis and processing times are presented and discussed.

1. INTRODUCTION

The area of chemical informatics (chemoinformatics or cheminformatics) is a recent field of study, in which informatics techniques are applied in the field of chemistry.[1] The field emerged from the aim of achieving data reliability and knowledge extraction from the vast amount of chemical data accumulated during the years and especially during the age of drug discovery and combinatorial chemistry. Thus, central to the cheminformatics field are databases of small molecules—where the structural information of compounds and their experimentally determined and/or predicted properties are stored.[2]

Recently, much effort has been directed in the development of algorithms that generate numeric information reflecting the physical properties of molecules. The motivation for this is typically the construction of empirical models that relate the structure of a molecule to its biological or physicochemical property values, a process known as QSAR (quantitative structure−activity relationship) when modeling biological properties and QSPR (quantitative structure−property relationship) when modeling physicochemical properties.[3] In addition to their use in QSAR/QSPR modeling, numerical representations of molecules are used in establishing quantitative values of similarity between molecules, such as 2D structural similarity,[4] 3D shape similarity,[5] or similarity in producing biological effects.[6,7] These quantitative indices of similarity are used in clustering groups of chemical compounds and increasingly in virtual screening.[8]

The principal motivation for the CDL is to provide a generic and high-performance software framework to use in the development of algorithms that address problems in chemical informatics, particularly in database design, QSAR/QSPR, and virtual screening. The secondary motivation is to provide a cross-platform software toolkit for chemical information storage/retrieval, QSAR/QSPR, and virtual screening.

The C++ language was used for this purpose since it supports a variety of features[9] to achieve these goals, including tight memory control, object-oriented programming, generic programming, and multiple inheritance.

Generic programming is a software engineering style in which data types are not explicitly defined while declaring objects and functions.[9] Data types that are not defined during declaration are said to be "abstracted" (or "parameterized") from the program, thus giving the flexibility to the object or funcion to operate on different data types. In C++, this technique is achieved by the use of templates.[9] Examples of generic objects are the containers of the C++ standard library (such as list or vector), where the contained type is abstracted, thus giving the flexibility to the container to hold objects of different types.

The CDL achieves software components that are generic with the development of an interface that abstracts the molecular data structure and associated properties. This interface allows access to the molecular data structure and properties while hiding the details of its implementation. Using this interface for the development of algorithms maximizes code reusability and gives the possibility for their use by external molecular data structures that provide the same interface. In addition, different molecular properties data structures can be implemented and used directly by the CDL without the need for further modifications.

The software is freely available[10] and distributed under the Boost Software License,[11] a permissive free software license. The use of this license is intended to achieve the

* Corresponding author e-mail: Vladimir.Sykora@newcastle.ac.uk. Corresponding author address: School of Natural Sciences, Bedson Building, Newcastle University, Newcastle Upon Tyne, NE1 7RU, U.K.

10.1021/ci800135h CCC: $40.75
Published on Web 09/20/2008

Table 1. Additional Open Source Software Libraries for Chemical Informatics Currently Available

library	description	programming language	license	main differences from CDL
OELib	original chemical information library from OpenEyes Scientific Software[12]	C++	GPL	nonpermissive license limited genericity in the definition of properties Most of the atom, bond, and molecule properties and functionality are hard-coded in the respective classes.
OpenBabel	freely available and open source continuation of OELib[13]	C++	GPL	same differences as of OELib
JOELib	redesigned Java successor of OELib[14]	Java	GPL	Java language same differences as of OELib
CDK	chemical information handling library originally developed to support different open source projects[15]	Java	LGPL	Java language semipermissive license most functionality hard-coded into classes

secondary goal of allowing the integration of the software into proprietary software, thus facilitating scientific cooperation between industry and academia by eliminating the need to overcome license limitations. This way, the use of the library in both commercial and noncommercial environments is encouraged.

Additional open source software libraries for chemical informatics are currently available, most of them distributed under the GPL, limiting its use in commercial environments. Table 1 lists currently available open source libraries for chemical informatics, their description, language, license, and main differences from the CDL.

A significant feature of the CDL is that it is portable. It is written in standard C++,[16] rendering it flexible for use on a variety of platforms and compilers.

The article is structured the following way: Section 2 describes the underlying representation of molecules and associated data structures and interfaces. Section 3 covers the initialization process of the molecular object. Section 4 details the main algorithms in CDL, including molecular text format parsers and writers, two-dimensional binary, pharmacophore and atom type fingerprints, substructure search (full search and fingerprint screened), and SMARTS search. Section 5 describes the methods used to evaluate the performance of the algorithms, and section 6 contains the performance results. The results of the performance tests are discussed in section 7, and concluding comments and a discussion of future work with the CDL are provided in section 8.

2. STRUCTURE REPRESENTATION

The CDL represents molecules as mathematical graphs, where vertices (or nodes) represent atoms and edges represent bonds. This representation is called a molecular graph.[17]

The CDL molecular graph represents a valence model, in which an edge connecting two vertices represents a covalent bond. The bond order (single, double, triple) is a property of the bond and is not represented by parallel edges between the vertices.

This representation is suitable for organic chemistry but does not handle properly the representation of inorganic compounds (metallic bonds in particular). See ref 18 for work extending data representations to include wider chemistry.

The CDL employs the C++ Boost Graph Library (BGL)[19,20] as the underlying data structure of the molecular graph, using an undirected adjacency list[21] specialization for the BGL graph data structure. In a cheminformatics context, the adjacency list is sometimes referred to as a "connection table".[22]

The BGL is part of the Boost project, a highly regarded community effort to provide high-quality, peer-reviewed C++ portable libraries intended for standardization. The BGL provides diverse graph data structures and algorithms, in addition to a generic interface for the traversal of the graph concept.

The BGL provides CDL with a rich collection of high performance graph algorithms for use with the underlying graph representation of molecules, including breadth-first search (BFS), depth-first search (DFS), and Dijkstra algorithms.

2.1. Properties. A set of properties that contains information about an atom within a molecule is attached to the corresponding vertex in the molecular graph. In the CDL, this set is referred to as an "atom", and the data structure that encapsulates it is called *atomic_properties*. Similarly, a set of properties that contain information about bonds is attached to each edge. This set is referred to as a "bond", and the data structure that encapsulates it is called *bond_properties*. In addition, there is a set of global properties that pertain to the whole molecule. These are contained outside the data structure of the graph, in a data structure called *molecular_properties*, and are referred to as the "molecule properties".

The CDL provides a generic[23] interface for the access of these atom, bond, and molecule properties. Using this interface, the property data structures are abstracted from the implementation. This means that users can provide their own property data structures and easily use CDL source code without the need for further modifications.

The properties interface is implemented with the use of "accessors" which are objects that provide necessary functions for the access of properties. Each property is referenced in the CDL by the accessor from which it belongs (either atom, bond, or molecule properties) and by a textual tag that identifies the individual property of the group (called a "selector"). For example, "atomic number" is a property contained in the *atomic_properties* data structure; hence it

Table 2. Atom Properties and Their CDL Selectors

property	selector
Static Properties (Read-Only)	
atomic number	atomic_numberS
atomic mass (Da)	atomic_massS
electronegativity as described by Burden (dimensionless)	burden_electronegS
electronegativity as described by Sanderson (Pauling units)	sanderson1_electronegS
atomic radius (m)	atomic_radiusS
melting temperature (K)	melting_temperatureS
boiling temperature (K)	boiling_temperatureS
critical temperature (K)	critical_temperatureS
standard physical state: 0 solid, 1 liquid, 2 gas	standard_stateS
atomic symbol	short_symbolS
van der Waals radii as described by Bondi (m)	bondi_vdw_radiiS
van der Waals radii as described by Rohrbaugh (m)	rohrbaugh_vdw_radiiS
van der Waals radii as described by Gavezzotti (m)	gavezzotti_vdw_radiiS
Dynamic Properties (Read-Write)	
charge	chargeS
flag to indicate heteroatom	hetero_atom_flagS
flag to indicate aromaticity	aromaticity_flagS
flag to indicate a chiral center	chiral_center_flagS
size of the smallest ring the atom belongs to	ring_sizeS
coordinates	coordinatesS
number of lone pairs the atom has	lone_pairsS
orientation of the chirality	atom_chirality_typeS

Table 3. Bond Properties and Their CDL Selectors

property	selector
type of bond	type_of_bondS
orientation of the bond respect to the chiral center	chiralityS
size of the smallest ring the bond belongs to	ring_bond_sizeS
flag indicates if the bond is rotatable	rotatable_flagS
flag indicates if the bond is aromatic	aromatic_bond_flagS
character of the bond: sigma or pi	bond_characterS
isomerism of the bond: cis, trans, or none	isomerismS

should be accessed by the atom property accessor (*Atom-Property*) and by the selector *atomic_numberS* which are hard-coded into the CDL.

The CDL provides default atom, bond, and molecule properties. Tables 2, 3, and 4 list the atom, bond, and molecule properties provided alongside their respective selectors.

The *atomic_properties* structure is distinctive from the *bond_properties* and *molecular_properties* in that it contains two classes of properties: one static (read-only), which contains immutable properties during the lifetime of the molecule, and the other dynamic (read-write) properties. Both *bond_properties* and *molecular_properties* data structures contain only dynamic properties.

For example, the atomic number is a static property since it defines the type of an atom and cannot change during the lifetime of the molecule. Changing the atomic number would change the constitution of the molecule itself. On the other hand, the atomic charge is a dynamic property since it gives information about its state.

Table 4. Molecule Properties and Their CDL Selectors

property	selector
sequence containing lists of vertices that belongs to each cyclic structure (i.e., rings) of the smallest set of smallest rings	cyclic_verticesS
sequence containing lists of edges that belongs to each cyclic structure (i.e., rings) of the smallest set of smallest rings	cyclic_edgesS
symmetric matrix with the shortest paths between the atoms of the molecular graph	distance_matrixS
adjacency matrix of the molecular graph	adjacency_matrixS
sequence containing the indices of rings which form fused ring systems	ring_systemsS
number of non-hydrogen atoms	num_non_h_atomsS

The use of static properties results in memory efficiency, since the application requires the allocation of memory for only one instance of each atomic number, independent of how many molecules are allocated in memory at any given time.

2.2. Structure Traversal. In addition to an interface for the access of properties, the CDL also defines a generic interface to traverse the structure of a molecular graph. Algorithms written using this interface are also generic, in the way that external (i.e., non-CDL) molecular data structures can use them, provided they implement the same generic interface.

Iterators[24] are the concept used for the implementation of this interface. Two global sets of iterators are provided by the CDL. The first traverses the graph structure of the molecular graph: vertices and edges. The second is an adaptation of the first and traverses the properties attached to the vertices and edges (atoms and bonds, respectively).

The CDL further classifies its iterators depending on the way the structure of the molecular graph is traversed. Each iterator belongs to one of three possible categories. The first category traverses all elements of the graph and its properties, i.e. vertices, edges, atoms, and bonds. The second category traverses the adjacent elements of vertices and atoms, i.e. adjacent vertices of a particular vertex or the adjacent atoms of a particular atom. The third category traverses the incident elements of vertices and atoms, i.e. incident edges of a particular vertex or incident bonds of a particular atom (incidence refers to the set of edges for which a particular vertex is a component).

Table 5 shows the iterators provided by the CDL, their category, and description.

Using these different traversal categories one can easily write algorithms for a particular context of application. For example, an iterator that traverses all atomic properties of the molecular graph is needed to calculate the molecular weight of a molecule. In another case, an iterator that traverses the adjacent atomic properties of a particular vertex is needed to check if that particular vertex has hydrogens attached.

Table 5. Molecular Graph Iterators in the CDL

iterator	traverses	category	description
vertex iterator	graph structure	all elements	traverses all vertices of the vertex set of the molecular graph
edge iterator	graph structure	all elements	traverses all edges of the edge set of the molecular graph
adjacency vertex iterator	graph structure	adjacent elements	traverses the adjacent vertices of a particular vertex
incident edge iterator	graph structure	incident elements	traverses the incident edges of a particular vertex
atom iterator	properties	all elements	traverses the atom properties of the vertex set of the molecular graph
bond iterator	properties	all elements	traverses the bond properties of the edge set of the molecular graph
adjacency atom iterator	properties	adjacent elements	traverses the adjacent atom properties of a particular vertex
incident bond iterator	properties	incident elements	traverses the incident bond properties of a particular vertex

2.3. The Molecule Object. The *molecule* class is the basic programming object in the CDL. It encapsulates the graph data structure and its associated properties (atom/bond/molecule). Also, it contains the iterator types and methods to access instances of them.

It is parametrized[25] on 3 main components: the atom, bond, and molecule properties. As a whole these encapsulate the actual properties of the molecular graph. Having the properties of the *molecule* parametrized allows the specification of custom-made property data structures for task-specific operations. For example, a fingerprint data structure can be added to the molecule property when writing fingerprinting algorithms and be discarded when using the molecule object in another context.

The *molecule* class functionality deals exclusively with the definition of an interface for the access of the molecular graph representation and its properties, while most of the functionality in the CDL is provided as free functions that operate on the *molecule* object.

The *molecule* object functionality can be divided into two main programming layers: the graph layer and the properties layer. The graph layer contains functionality that refers to the underlying graph representation of the molecule. In this layer, iterators traverse vertices and edges and point to their respective descriptors. A vertex descriptor contains a unique integer index that identifies the vertex within the molecular graph. An edge descriptor contains a pair of vertex descriptors that represent the source and target vertices of that edge. In a similar way, the property layer contains functionality that refers to the properties attached to the vertices and edges of the graph: atom and bonds, respectively. In this case, the iterators traverse the atoms and bonds. The actual property values are obtained using the atom/bond iterators in conjunction with the corresponding property accessor and selector. Figure 1 shows a diagram describing this layers separation in the context of the graph representation of the molecule trimethylamine.

3. MOLECULE INITIALIZATION

The *molecule* object is usually instantiated (constructed in C++ terms) by an existing molecular representation in the form of an accepted molecular text format. Some computationally intensive calculations are performed during instantiation, including ring perception, bond order assignment, the computation of the adjacency and distance matrices,[26] aromaticity perception, and the assignment of the different flags for atoms and bonds (such as heteroatom flag and rotatable bond flag). The results from these computations are stored in the internal properties of the *molecule* object, a process described as 'initialization' in the CDL. Having these results accessible from the internal properties of the molecular graph removes the requirement to recalculate them during calls to algorithms that require them or during repetitive calls to the same algorithm. The topological pharmacophore fingerprint generation and the atom type fingerprint generation algorithms (detailed in section 4) are examples where computation is saved due to the use of the internal properties stored in the *molecule* object.

Following, each phase of the initialization process is described in detail given in order of execution.

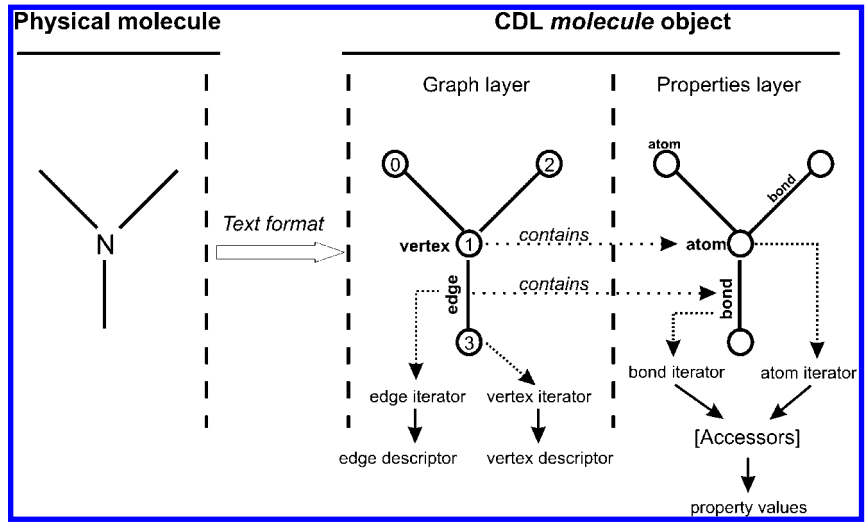

Figure 1. The CDL *molecule* object representing trimethylamine and its programming layers.

3.1. Ring Perception. Ring perception is the first step in the initialization phase of a CDL molecule object. The CDL perceives rings by the calculation of the union of all minimum cycle bases of the molecular graph.[27] This set of rings, originally named the set of *K*-rings by Plotkin,[28] has a set of properties that are necessary for the proper description of a molecule by information systems.[27] These properties include uniqueness, completeness, being discriminatory, efficiently computable, and chemically relevant. This set of rings is also known as the set of relevant cycles $R(G)$ of graph G.

The CDL employs an implementation of the Vismara's algorithm[29] for the calculation of the set of relevant rings $R(G)$, having a worst-case complexity of $O(ve^3)$, with v being the cyclomatic number (also referred to as nullity or Frerejacque number[30]) and e the number of edges.

3.2. Bond Order Assignment. The CDL does not allow explicit specification of conjugated systems (this also applies to tautomerism). The CDL assigns bond orders in fixed positions during initialization. In the case of aromatic ring systems (the way CDL detects aromaticity is given in section 3.5) the CDL flags the atoms and bonds that constitute the aromatic system as aromatic, while fixing the positions of the double bonds in the conjugated system.

In the case when the *molecule* object is instantiated from a text format that includes aromaticity information (and double bonds are not specified explicitly), double bonds are assigned to aromatic systems by an iterative procedure, in which double bonds are alternatively assigned (valence permitting) until an aromatic system is found.

3.3. Implicit Hydrogen Assignment. The *molecule* object constructor accepts a flag which specifies whether to assign implicit hydrogens to the molecular graph. If the flag is passed, hydrogens are added to the constituent atoms using valence rules.

3.4. Adjacency and Distance Matrices. These two matrices are often used in algorithmic procedures involving the molecular graph, a reason for which both are stored in the internal properties of the molecule. The adjacency matrix is assigned in $O(n)$, while the distance matrix is assigned in $O(en + n^2)$, with n being the number of vertices and e the number of edges in the molecular graph.

3.5. Aromaticity Perception. The CDL detects aromaticity in ring systems by the application of the Hückel rule.[31] This definition succeeds in identifying aromaticity in ring systems for most cases in organic chemistry; however, it fails for some polycyclic aromatic hydrocarbons, pyrene and coronene being notable examples.

In CDL, a ring system is conformed by the group of fused rings that belongs to the same biconnected component.[32] The size of a ring system refers to the number of rings that constitutes the system. In CDL, ring systems are readily calculated during the ring perception phase.

The aromaticity perception algorithm starts by checking aromaticity in each ring system as a whole. If a group of fused rings is aromatic (as defined by the Hückel rule), then all atoms and bonds belonging to that particular group are flagged as aromatic. On the other hand, if the group is not aromatic, then a recursive procedure is performed that partitions the original group of fused rings into groups of fused rings of decreasing size. For example, if a ring system is constituted by 5 fused rings, then the algorithm partitions that set into sets of 4 fused rings, then the sets of 3 fused rings, and so on, until completely partitioning the original fused system into the single constituent rings. At each level, the algorithm checks for aromaticity for each fused group, and if aromatic, then the constituent atoms and bonds are temporarily marked as aromatic. The algorithm finally tags as aromatic the partitioned group or combination of groups that comprises the largest number of aromatic atoms.

3.6. Flags Assignment. The final step of the initialization is the assignment of the diverse flags for atoms and bonds. These include heteroatoms, chiral centers, and rotatable bonds.

4. CORE ALGORITHMS

In this section, the core algorithms implemented within the CDL are described.

4.1. Molecular Text Parsers. Text format is the most common form of storing and manipulating chemical data sets on computers.

There are many approaches for text molecular representation,[22] but only two broad classes have received widespread attention: linear notation and the connection table. From the class of linear notation representations, SMILES[33] is the most popular, while MDL's SDfile[34] is the commonly used form of connection table. The CDL provides parsers and writers for these two formats. These algorithms operate on C++ standard streams.

4.1.1. SMILES. The SMILES notation is perhaps the most widely used molecular text format. It is defined by a set of rules published originally[33] and extended by Daylight Chemical Information Systems Inc.[35] This set of rules represents a valence model of a molecule and effectively defines a language.[36] This language has the drawback that it lacks a formal grammar specification, causing major variations in the acceptance of the different SMILES notations between different software vendors. See ref 37 for a current open initiative to specify a context free grammar for the SMILES notation.

The CDL SMILES parser algorithm follows a set of rules defined by Daylight.[35] It has a computational complexity of $O(c)$ where c is the number of characters in the SMILES expression.

4.1.2. SDfile. The SDfile format is also widely used. Originally specified by MDL Information Systems,[34] it contains concatenated MOLfiles (a connection table representing one molecule) and special definitions to attach external data (e.g., biological or chemical properties) to each MOLfile. The CDL SDfile format parser algorithm also has a complexity of $O(c)$ where c is the number of characters in the molecular format. It currently supports version V2000 of the format.

4.2. Molecular Text Writers. Molecular text writer algorithms produce molecular notations from the molecular graph, using either formal grammars, a set of rules, or file specification documents.

Some notations have the advantage of being able to be canonicalized. This feature is extremely useful for database design, since the canonicalized notation can serve as the unique key of a compound table in a database.

4.2.1. SMILES. The CDL SMILES writing algorithm is a version of the original SMILES canonicalization algo-

rithm.[38] This produces a unique SMILES expression from a molecular graph. The algorithm is conducted in two steps: first is the generation of invariants for each vertex of the molecular graph and then the invocation of a depth-first search algorithm to actually generate the unique SMILES expression.

The first step involves the generation of integer values for each vertex, drawing information from the atomic composition and connectivity. A procedure is then performed to differentiate the vertices that were assigned the same integer value. This procedure has a worst-case performance of $O(n^2 log(n))$ where n is the number of vertices in the molecular graph.

The second step of the algorithm involves the invocation of a DFS to write the structure into text. This step has a complexity of $O(n + e)$ where n is the number of vertices and e is the number of edges in the molecular graph.

Overall, the CDL SMILES writing algorithm has a worst-case complexity of $O(n^2 log(n) + n + e)$.

4.2.2. SDfile. The CDL SDfile format writer produces a textual molecular representation in the MDL SDfile format version V2000.

No canonicalization is performed, hence it has a performance of $O(n)$ where n is the number of vertices in the molecular graph.

4.3. Substructure Search. Solving the subgraph isomorphism problem[32] is one of the most common tasks in chemical informatics. Algorithms that solve this problem can be used to retrieve compounds from a database that contain a particular interesting fragment. Essentially, the technique determines whether a molecular graph MG1 is isomorphic (has the same connectivity) to a subgraph of molecular graph MG2.

The subgraph isomorphism problem is known to be NP-complete;[39] however, in chemistry, algorithmic implementations can take advantage of atomic and bond information to greatly reduce computational effort.

The CDL solves the subgraph isomorphism problem using an implementation of the Ullmann algorithm for subgraph isomorphism,[40] which Kukluk et al.[41] report to have a complexity of $O(n!n^3)$ where n is the number of vertices in the MG1 graph (the substructure).

The CDL implementation of the algorithm uses a refinement method that draws information about connectivity and molecular information to discard solution candidates.

Additionally, when searching a fragment in a compound database, two-dimensional binary fingerprints can be used to reduce the search space, so only a subset of compounds will be subjected to a full substructure search.

4.4. SMARTS Language for Molecular Patterns and Properties. SMARTS[42] is a popular language to specify molecular patterns and properties. With it, it is simple to specify fragments—or atomic connectivities—to search for in compound databases.

The SMARTS language is a superset of the SMILES notation, defined by a set of rules which specify how to encode atomic connectivity and properties into a text representation.

In the SMARTS language, primitives refer to rules that apply to atoms and bonds. For example, atomic primitives include the specification of the atomic number, number of connections, or the number of hydrogens attached to a particular vertex, while bond primitives include the bond type (single, double, triple, or aromatic) of a particular edge. Primitives can be combined using logical operators to form expressions.

The CDL provides an implementation wherein a SMARTS string is initially parsed and converted into a labeled graph. Each atomic and bond expression in the SMARTS string is transformed to a binary abstract syntax tree, where parent nodes contain logical operators and children nodes contain atomic (or bond) primitives. These binary abstract syntax trees become the labels within the vertices and edges of the SMARTS labeled graph.

Recursive SMARTS are multigraphs, in which the label for the recursive expression contains an additional SMARTS labeled graph.

The SMARTS labeled graph can be applied within the CDL substructure search algorithm.

The complexity of the SMARTS parsing is linear: $O(c)$ where c is the number of characters in the SMARTS expression.

For nonrecursive SMARTS expressions, the worst-case complexity equals to one application of the Ullmann algorithm for subgraph isomorphism, i.e. $O(n!n^3)$ where n is the number of atomic expressions in the SMARTS. In the case of recursive SMARTS, there is one further application of the Ullmann algorithm per recursive pattern present in the SMARTS.

During SMARTS search, the computing time is drastically reduced by the evaluation of the atomic primitives, since the search algorithm is terminated when an atomic primitive cannot be matched in the target molecular graph.

4.5. Two Dimensional Binary Fingerprints. The 2D binary fingerprints (substructure or hashed fingerprints[17]) are binary (or Boolean) vectors that contain information about the constituent atoms and fragments of a molecule. The CDL implementation of the algorithm (referred to as 2DBF) encodes all atom sequences along the shortest paths between two atoms present in a molecule, considering paths up to a maximum user-defined number of bonds separating two atoms (usually 7).

The 2DBF representation is useful to quickly search for the possible presence of a fragment in a database of molecules, a process called "screening" in a database context. 2DBF vectors may also be used to establish structural similarity between two molecules.[4]

Using 2DBF vectors to search for the presence of a particular fragment in a database is extremely fast compared to a full graph isomorphism matching computation.

To calculate a 2DBF vector of an input molecule, the CDL's implementation starts with the initialization of each element of a binary vector of user-defined size (usually 1024) to zero.

Next, the algorithm iterates over each vertex of the initial molecular graph, calculating the shortest walk paths between the considered vertex and the remaining vertices. The use of shortest paths for the calculation means that only linear paths between two vertices are considered (there are no branches or cycles).

The paths are then transformed to an array of integers, where each element represents the atomic number contained in each vertex of the path. Each of these arrays is then transformed to an integer value with the use of a hash

function. Then each hashed value is used to seed a pseudorandom number generator (PRNG) that generates integer values in the range of the indices of the considered 2DBF vector. Integers returned by the PRNG will be the index (position) in the 2DBF vector which is then switched on (assigned a value of *one* or *true*). Note that different path arrays can possibly produce the same hashed value. This situation is called "collision" and intrinsically arises from the use of a hash function. The result is that different constituent fragments can, with a very low probability, switch on the same bit of the 2DBF.

The shortest walk paths between a particular vertex and the remaining vertices of the molecular graph are calculated using a breadth-first search algorithm.[43] The BFS algorithm runs in $O(e + n)$, where e denotes the number of edges, and n is the number of vertices in the molecular graph. This computation has to be performed for each vertex in the molecular graph, resulting in an overall worst-case complexity of $O(en + n^2)$.

The CDL 2DBF implementation differs from Daylight's fingerprints[44] in that CDL 2DBF only considers the atomic numbers of atoms along the generated paths, while Daylight's considers both the atomic symbols and the symbol of the bond types (explicit single, double, triple, and aromatic).

To apply screening in a substructure search query in a database, each compound of the database must have a 2DBF representation. Next, the 2DBF vector of the query fragment is calculated, as is a vector containing the intersection (*AND* operation) from the query vector and the database vector. If the resulting intersection vector is identical to the 2DBF vector of the query fragment, then the fragment *may* be present in the database compound. In this situation, a full subgraph isomorphism between the fragment and the structure of the database is calculated to ensure the query fragment exists within the database compound.

4.6. Topological Pharmacophores Fingerprints. Pharmacophores are molecular structural features that, through their interaction with the receptor active site, are responsible for that molecule's biological activity.[45]

In the CDL, pharmacophores characterize atomic environments and encode information if the environment is 1) a hydrogen bond donor, 2) a hydrogen bond acceptor, 3) lipophilic, 4) positively charged, and/or 5) negatively charged.

To encode information about these environments and their mutual separation in the molecular topology, CDL implements a description similar to that of Schneider et al.,[46] who used it for virtual screening. This description, referred to as the topological pharmacophores fingerprint (TPF), encodes the number of bonds that separates two features present in different atomic environments and results in a histogram vector that contains all possible feature pairs and their topological separation up to a maximum of 10 bonds apart. The topological distance separating the features is calculated by the shortest path between the atomic environments.

The algorithm requires the shortest paths between each vertex of the molecular graph and the remaining vertices. This computation is performed by default in the CDL upon instantiation of a *molecule*, thus excluding this computation from the calculation of the TPF vector.

Next, the algorithm computes which features each atom exhibits (acceptor, donor, etc.). This procedure has linear complexity $O(n)$ where n are the number of vertices in the graph.

The number of possible combinations from 5 pharmacophore types is 15. Since each possible combination is searched within a separation of up to 10 bonds apart, the algorithm produces a [15 combinations $\times$ 10 max. bonds of separation] $=$ 150 dimensional vector.

4.7. Atom Types Fingerprint. The atom types fingerprint (ATF) method is similar to the TPF algorithm previously described, the main differences being the atomic features considered and that this method encodes features from a root atom and does not generalize over the whole molecule, thus fingerprinting only the environment of a base atom. Also the method counts the presence of individual atom types, not pairs as in the TPF.

The ATF method was described by Xing et al.[47] where it was used for the prediction of logP, pK_a, and LogD. The ATF is based on the hypothesis that the ionization state of a particular group is dependent upon its neighboring atoms and bonds.

In the CDL's implementation, the algorithm takes a user-selected root atom (usually an ionization center) and produces a histogram that counts the number of occurrences of atom types in short paths distances from itself up to a path length of 5 bonds. In this implementation we consider a total of 23 atom types, including sp, sp^2, sp^3, and aromatic carbon; carbon cation; sp, sp^2, sp^3, and aromatic and amide nitrogen; sp^3 nitrogen positively charged; sp^2 and sp^3 oxygen; oxygen in carboxylic and phosphoric acid; sp^2 and sp^3 sulfur; sulfoxide and sulfone sulfur; and hydrogen; fluorine, chlorine, bromine, and iodine. The presence of atom types in the proximity of the root atom is encoded in the resulting vector so that each vector position encodes both the atom type count and the number of bonds separating it from the root atom. Since the types of the root atom are encoded, as are the neighbor atoms up to 5 bonds apart from the root atom, the final dimensionality of the vector equals 138 integer elements. This procedure requires the calculation of the shortest path from the root atom to the remaining atoms. Again, since this computation is performed by default during instantiation of the *molecule* object, it is not required during generation of the ATF itself.

Once the shortest paths from the root vertex to the rest is calculated, the complexity of this algorithm is linear, where $O(n)$ computations are required to assign each vertex n to an atom type.

5. CDL ALGORITHM PERFORMANCE EVALUATION: METHODS

Three sets of experiments were used to quantify the performance of the algorithms described in section 4. In the first set of experiments, the computation time of each algorithm is measured as the number of atoms in a molecule is increased. This set of experiments yields a quantitative value of performance as a function of the molecular size, since much of the theoretical worst-case algorithmic complexity depends on the number of atoms in the molecule. The chemical structures used for this experiment were nonbranched, single-bonded, linear carbon chains, starting

Table 6. Molecular Characteristics of the Three Virtual Molecule Data Sets

	molecular weight (Da)				number of heavy atoms				number of bonds			
data set	mean	SD	min	max	mean	SD	min	max	mean	SD	min	max
1K	523.38	287.36	64.04	2072.93	40.66	22.56	4	161	43.57	24.94	3	175
50K	489.85	276.15	13.01	2332.95	38.23	21.6	2	181	40.94	23.89	1	195
100K	486.6	271.64	2.01	2332.95	38.07	21.28	2	181	40.78	23.57	1	195

	number of rings				max. ring size				min. ring size			
data set	mean	SD	min	max	mean	SD	min	max	mean	SD	min	max
1K	3.91	2.57	0	15	5.75	1.21	0	7	5.08	1.32	0	6
50K	3.71	2.48	0	15	5.78	1.15	0	7	5.18	1.28	0	7
100K	3.70	2.49	0	15	5.76	1.18	0	7	5.18	1.30	0	7

Table 7. Expt. Set 2: Worst-Case Theoretical Complexity and Relative Performance of CDL Algorithms (Excluding Substructure Search) As Evaluated on Three Sets of Virtual Molecules

algorithm	worst-case complexity[a]	1K compounds (s)	50K compounds (s)	100K compounds (s)
SMILES parsing	$O(c)$	0.07	3.4	6.74
SDfile parsing	$O(c)$	0.5	24.42	46
SMILES canonicalization and writing	$O(n(n\ log(n) + 1) + e)$	1.97	88.56	176.87
SDfile writing	$O(n)$	0.33	15.59	31.24
fingerprints generation	$O(en + n^2)$	2.75	123.82	244.68
topological pharmacophores generation	$O(e + n)$ for shortest-path computation, then $O(n)$ for type assignment	0.09	4.36	8.88
atom type fingerprints generation	$O(e + n)$ for shortest-path computation, then $O(n)$ for type assignment	0.01	0.3	0.61

[a] n refers to the number of vertices (atoms), e to the number of edges (bonds) of the molecular graph, and c to the number of characters in the text format.

Table 8. Expt. Set 3: Substructure Search Performance

		1K data set			50K data set			100K data set		
substructure SMILES	no. of atoms in substructure	full search (s)	2DBF screened (s)	hits	full search (s)	2DBF screened (s)	hits	full search (s)	2DBF screened (s)	hits
CCC	3	0.13	0.13	978	6.36	6.34	48667	12.67	12.76	97495
CCCC	4	0.31	0.3	935	14.87	14.82	46151	29.98	29.86	92569
CCCCC	5	0.77	0.74	824	36.67	35.52	41299	72.75	70.4	83169
CCCCCC	6	1.86	1.72	692	85.42	79.71	34526	168.95	157.27	69911
CCCCCCC	7	3.85	3.59	554	175.27	162.67	27411	345.4	320.08	55759
CCCCCCCC	8	6.92	5.86	451	317.41	268.54	21272	623.55	526.66	43448
CCCCCCCCC	9	11.6	10.23	349	515.57	453.84	16070	1003.99	882.4	33277
CCCCCCCCCC	10	17.99	16.29	264	779.22	704.11	11767	1515.12	1369.24	24794

with 3 carbon atoms and increasing one carbon atom per structure to a maximum of 30 carbon atoms. Elapsed times were measured using the ANSI C function *clock()*, which returns the total time used by a process. All calculation times are reported in seconds. The resolution of the *clock()* function is too low for the accurate reporting of the computational time for one chemical structure by each algorithm. Hence, in the first set of experiments each algorithm processed 5000 replicates of the chemical structure being studied.

The second set of experiments is used to quantify the time taken for an algorithm to complete when applied to data sets of increasing size, i.e. data sets comprising 10000, 50000, and 100000 molecules. These will be referred to as the 1K, 50K, and 100K data sets, respectively. These experiments provide a quantitative value of the time taken for an algorithm to complete as a function of the number of molecules that are processed. Each data set comprises virtual molecules (molecules that exist only as computational representations). These were generated by applying specialized genetic operators to a set of predefined molecular fragements.[48,49] Table 6 shows statistics of the molecular characteristics for each data set.

The third set of experiments is used to quantify the performance of the substructure search algorithm and the SMARTS algorithm. In the substructure search experiment, both "2DBF screened" and "full substructure search" of linear carbon chains of increasing size (shown as SMILES expressions in the first column of Table 8) are evaluated. This is performed on the same data sets of virtual molecules used in the second set of experiments. An equivalent test was performed for the SMARTS algorithm, searching SMARTS patterns with increasing number of atomic expressions (each expression having two primitives, shown in the first column of Table 9). Two primitives in each atomic expression were used with the aim of differentiating the SMARTS search from the substructure search algorithm.

Table 9. Expt. Set 3: SMART Search Performance

		1K compounds		50K compounds		100K compounds	
SMARTS	no. of atomic expressions	seconds	hits	seconds	hits	seconds	hits
[CR0][CR0]	2	0.36	893	16.97	43040	33.58	86130
[CR0][CR0] [CR0]	3	0.57	733	25.59	34123	50.89	68374
[CR0][CR0] [CR0][CR0]	4	0.81	457	35.92	20835	72.38	42673
[CR0][CR0] [CR0][CR0] [CR0]	5	1.11	291	48.72	12306	98.53	25800
[CR0][CR0] [CR0][CR0] [CR0][CR0]	6	1.46	182	62.86	7321	127.26	15548
[CR0][CR0] [CR0][CR0] [CR0][CR0] [CR0]	7	1.82	115	77.96	4196	158	9205
[CR0][CR0] [CR0][CR0] [CR0][CR0] [CR0][CR0]	8	2.2	74	93.48	2509	189.99	5389
[CR0][CR0] [CR0][CR0] [CR0][CR0] [CR0][CR0] [CR0]	9	2.62	40	109.99	1322	223.61	3034
[CR0][CR0] [CR0][CR0] [CR0][CR0] [CR0][CR0] [CR0][CR0]	10	3.01	24	126.46	812	257.49	1878

Excluding parsers, each algorithm requires the instantiation of a *molecule* object, a process not accounted for in the measurement of the running time for each algorithm.

The experiments were performed under the Linux operating system running on a computer equipped with two Dual-Core Intel Xeon processors with clock rates of 3 Ghz and 4 GB of RAM.

6. CDL ALGORITHM PERFORMANCE EVALUATION: RESULTS

The results from the first set of experiments are summarized in Figure 2. This shows the elapsed time taken for each CDL algorithm to complete the calculations of 5000 replicates of nonbranched, single-bonded linear carbon chains of increasing size, from 3 to 30 carbon atoms each. Figure 2 shows that the 2DBF generation and the SMILES canonicalization and writing algorithms exhibit nonlinear running times, as expected by the theoretical worst-case algorithmic complexities of $O(en + n^2)$ and $O(n^2\ log(n) + n + e)$, respectively. The remaining algorithms exhibit linear running times.

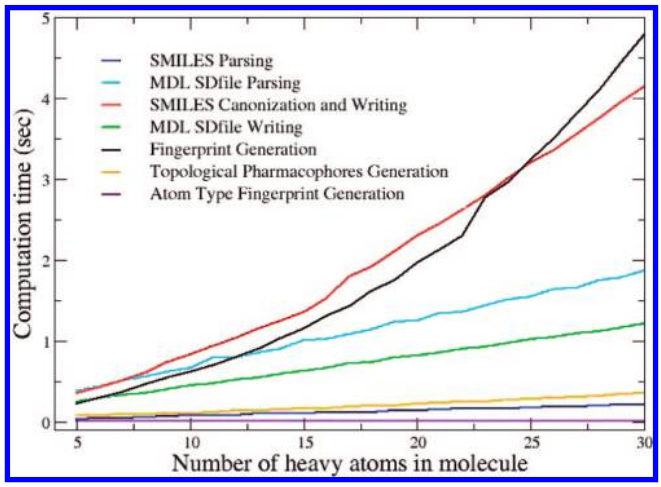

Figure 2. Expt. set 1: computation time of CDL algorithms (excluding substructure search) against number of atoms per molecule.

The results from the second set of experiments are summarized in Table 7 which displays both the theoretical algorithmic complexity for each algorithm (excluding substructure search and SMARTS) and the computation times for the 1K, 50K, and 100K virtual molecule data sets. The computation times listed in this table follow the same pattern as shown in Figure 2: from the fastest to the slowest. From this we can conclude that the testing of only nonbranched, single-bonded linear carbon chains in the first set of experiments is consistent with the theoretical algorithmic performance for each algorithm and, hence, is likely to be a good indicator of computational performance when operating on more complex molecules.

The results from the third set of experiments describing substructure search and SMARTS search are summarized in Tables 8 and 9, respectively. Table 8 lists the time taken for the substructure search algorithm to search structures of increasing size in the data sets of the second experiment, along with the number of hits. Both Fingerprint screened and full substructure search were performed. The same study was conducted for the SMARTS search (Table 9) without 2DBF screening.

7. DISCUSSION OF EXPERIMENTAL RESULTS

Molecular Text Format Parsers. The time required for SDfile parsing is orders of magnitude higher than that required for SMILES parsing. This result is expected since the SDfile format needs more characters than the SMILES format to encode the same chemical structure. For example, the SDfile representations of the molecules in the second set of experiments contained more than 41 times the characters than the equivalent SMILES representations. The reason for this is that the SDfile format is specified by a file specification document, requiring the presence of text in specified placeholders in order to successfully parse the format (whether or not they actually contain relevant information of the molecule). In contrast, the SMILES notation is specified by a set of formal rules where each character contains relevant information about the described molecule and so this representation is relatively compact. Furthermore, the SMILES notation implicitly describes the connectivity of the molecule, whereas the SDfile format must explicitly describe the atoms forming a bond, incurring the need for additional text characters.

The difference in the number of characters needed for each molecular text representation leads to a difference in disk storage requirements. This difference can be significant in situations where disk space is constrained or when storing data sets of many hundreds of thousands of compounds.

Molecular Text Format Writers. The SMILES canonicalization and writing algorithm exhibits polynomial running time, whereas the SDfile format writing algorithm runs linearly. This is due to the canonicalization process for SMILES, where a unique string is produced for the compound being written. There are a variety of situations where the computation overhead incurred by the canonicalization process is required by the implementation, such as normalization of databases of chemical structures or the removal of duplicated structures within data sets.

2D Binary, Pharmacophores, and Atom types Fingerprints Generation. The 2DBF generation is the most computationally expensive algorithm of those studied, requiring over 4 min to compute the data set of 100K molecules (Table 7). This is because it needs to calculate all walk paths from each vertex of the molecular graph to the remaining vertices, a computation requiring worst-case complexity of $O(en + n^2)$, with e being the number of edges and n the number of vertices in the molecular graph. Once computed however, the 2DBF is usually stored in a compound table of a database, where it can be subsequently used to screen substructure search queries.

The other fingerprint algorithms (TPF and ATF) run linearly, since the calculation of the distance matrix is performed upon the *molecule* object on initialization.

Substructure and SMARTS Search. It is interesting to see that the SMARTS search runs linearly with respect to the number of atomic expressions (Figure 3) when each expression has two atomic primitives (joined by the AND operator), as opposed to the full substructure search algorithm, which runs nonlinearly with respect to the size of the substructure. The SMARTS algorithm runs linearly for these expressions because the evaluation of the primitives terminates the search prematurely for the structures that can never match the expression.

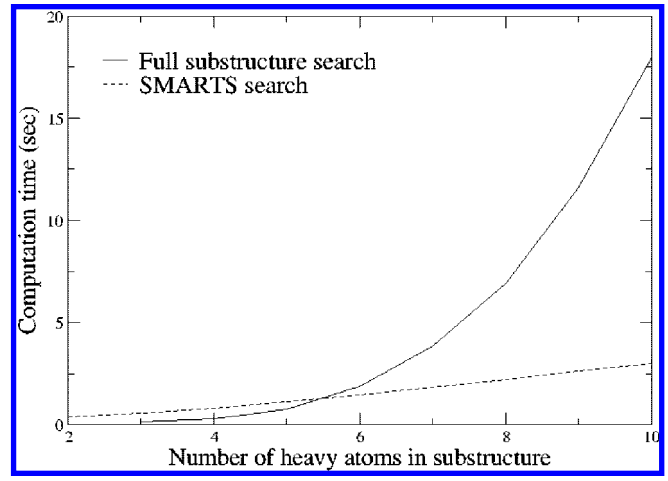

Figure 3. Dependence of substructure search performance on substructure size for full substructure search and SMARTS search.

Figure 3 shows the running times of the full substructure and SMARTS search for the data set of 1K molecules (see column 3 of both Table 8 and 9). The trend of the search time in the data sets of 50K and 100K molecules is very similar (not shown).

Both full substructure (Table 8) and SMARTS (Table 9) searching times increase with the size of the substructure to search. This is to be expected since both algorithms use the Ullmann algorithm for subgraph isomorphism, which increases its computation time with the number of vertices in the subgraph.

The improvement of searching times using 2DBF screening, as opposed to a full substructure search (Table 8), usually increases with the size of the substructure; however, further studies are needed in order to define a clear trend.

8. CONCLUSIONS

An overview of the Chemical Descriptors Library CDL, a generic, high performance C++ software library for chemical informatics algorithms development, has been presented. The CDL represents molecules as mathematical graphs, where nodes represent atoms and edges represent bonds. The flexibility of the library is obtained by the use of three modern programming techniques: 1) the parametrization of the internal properties of the molecular graph, 2) the use of iterators as a generic interface for the traversal of the structure of the underlying graph representation, and 3) the use of property accessors as a generic interface to access the parametrized internal properties of the graph.

Using these interfaces, various algorithms of high importance in chemical informatics have been implemented. These include SMILES and SDfile molecular text format parsers and writers and 2D binary, pharmacophoric, and atom type fingerprints generation as well as substructure search and SMARTS language for atomic pattern matching. The parametrization of the internal properties of the *molecule* object achieves the flexibility to attach necessary internal properties as the implementation requires. This saves computational and memory overhead in that all possible properties do not have to be attached in every area of application. In addition, having these properties parametrized allows external users to provide their own property data structures and still be able to use all

CDL code, as long as they provide necessary property accessors. In the same way, external molecular data structures can still use all algorithms of the CDL, as long as they provide the same iterators interface and the same property accessors. The resulting algorithms are modular, less error prone, and simple to incorporate into external software projects.

In terms of algorithmic performance (Figures 2 and 3 and Tables 7, 8, and 9), SMILES canonicalization and writing, 2D binary fingerprint generation and the full substructure search algorithms run nonlinearly with respect to the size of the input, whereas parsers, SDfile writing, topological pharmacophores, and atom types fingerprint generation run linearly with respect to the size of the input. To process a data set of 100000 small molecules, the 2D binary fingerprint generation algorithm took 244.68 s to complete (slowest), whereas it took 0.61 s for the atom types fingerprints generation algorithm to complete (fastest). With respect to substructure search in the same data set, it took over 25 min for the full substructure search algorithm to search the longest query (resulting in 24794 hits) and 12.67 s for the shortest query (97495 hits). In contrast, it took the SMARTS algorithm over 4 min to search for the longest expression in the same data set (1878 hits) and 33.58 s for the shortest expression (86130 hits), much less than the full substructure search algorithm.

Future work using the library includes the calculation of molecular shape similarity, 3D structure predictions, and the development of genetic operators that operate on reduced graph representation of chemical structures.

ACKNOWLEDGMENT

This study was partly funded by grant C18784/A8939 from Cancer Research U.K. The authors thank Damian Krstajic from the Research Centre for Cheminformatics for hardware and software support and Dr. Sandeep Pal and Dr. Dominic Searson for reading and commenting on the initial manuscript. V.J.S. additionally thanks Michael Almstetter and Dr. Henrik Kuhn for support during the initial stage of the CDL project.

REFERENCES AND NOTES

(1) Gasteiger, J. The Scope of Chemoinformatics In *Handbook of Chemoinformatics*; Gasteiger, J., Ed.; Wiley-VCH: Weinheim, Germany, 2003; Vol. 1, pp 3−5.
(2) Mitchell, A. Miller. Chemical Database Techniques in Drug Discovery. *Nat. Rev. Drug Discovery* **2002**, *1*, 220–227.
(3) Hansch, C.; Leo, A. *Exploring QSAR: Fundamentals and Applications in Chemistry and Biology*; American Chemical Society: Washington, DC, 1995.
(4) Willett, P.; Barnard, J. M.; Downs, G. M. Chemical Similarity Searching. *J. Chem. Inf. Comput. Sci.* **1998**, *38*, 983–996.
(5) Grant, J. A.; Gallardo, M. A.; Pickup, B. T. A Fast Method of Molecular Shape Comparison: A Simple Application of a Gaussian Description of Molecular Shape. *J. Comput. Chem.* **1996**, *17*, 1653–1666.
(6) Thornber, C. W. Isosterism and Molecular Modification in Drug Design. *Chem. Soc. Rev.* **1979**, *8*, 563–580.
(7) Patani, G. A.; LaVoie, E. J. Bioisosterism: A Rational Approach in Drug Design. *Chem. Rev.* **1996**, *96*, 3147–3176.
(8) Walters, W. P.; Stahl, M. T.; Murcko, M. A. Virtual Screening − an Overview. *Drug Discovery Today* **1998**, *3*, 160–178.
(9) Stroustrup, B. *The C++ Programming Language*, 3rd ed.; Addison-Wesley Professional: Indianapolis, IN, 1997.
(10) SourceForge.net: Chemical Descriptors Library (CDL). http://sourceforge.net/projects/cdelib (accessed June 18, 2008).
(11) http://www.boost.org/LICENSE_1_0.txt (accessed June 18, 2008).
(12) OpenEye Scientific Software | OELib. http://www.eyesopen.com/products/toolkits/oelib.html (accessed June 18, 2008).
(13) Main Page − Open Babel. http://openbabel.org/wiki/Main_Page (accessed June 18, 2008).
(14) Main Page − JOELib. http://joelib.sourceforge.net/wiki/index.php/Main_Page (accessed June 18, 2008).
(15) Steinbeck, C.; Han, Y.; Kuhn, S.; Horlacher, O.; Luttmann, E.; Willighagen, E. The Chemistry Development Kit (CDK): An Open-Source Java Library for Chemo- and Bioinformatics. *J. Chem. Inf. Comput. Sci.* **2003**, *43*, 493–500.
(16) ISO/IEC 14882:2003 − Programming Languages − C++. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=38110 (accessed June 18, 2008).
(17) Leach, A.; Gillet, V. Representation and Manipulation of 2D Molecular Structures. In *An Introduction to Chemoinformatics*; Springer: Dordrecht, The Netherlands, 2005; pp 1−19.
(18) Bauerschmidt, S.; Gasteiger, J. Overcoming the limitations of a connection table description: a universal representation of chemical species. *J. Chem. Inf. Comput. Sci.* **1997**, *37*, 705–714.
(19) Boost C++ Libraries − Table of Contents: Boost Graph Library. http://www.boost.org/doc/libs/1_35_0/libs/graph/doc/table_of_contents.html (accessed June 18, 2008).
(20) Siek, J.; Lee, L.-Q.; Lumsdaine, A. *The Boost Graph Library: User Guide and Reference Manual*; Addison-Wesley Professional: Indianapolis, IN, 2001.
(21) Cormen, T.; Leiserson, C.; Rivest, R.; Stein, C. *Introduction to Algorithms*, 2nd ed.; MIT Press: Cambridge, MA, 1990.
(22) Willett, P. A History of Chemoinformatics In *Handbook of Chemoinformatics*; Gasteiger, J., Ed.; Wiley-VCH: Weinheim, Germany, 2003; Vol. 1, pp 6−8.
(23) Matthew H. Austern. *Generic Programming and the STL: Using and Extending the C++ Standard Template Library*; Addison-Wesley Professional: Indianapolis, IN, 1998.
(24) Noble, J. Iterators and Encapsulation In *Technology of Object-Oriented Languages*; Proceedings of the 33rd international Conference, Mont-Saint-Michel, France, June 2000; IEEE: New York, NY, 2000; pp 431−442.
(25) Czarnecki, K.; Eisenecker, U. W. *Generative Programming: Methods, Tools and Applications*; Addison-Wesley Professional: Indianapolis, IN, 2000.
(26) Todeschini, R.; Consonni, V. *Handbook of Molecular Descriptors*; Mannhold, R., Kubinyi, H., Timmerman, H., Eds.; Wiley-VCH: Weinheim, Germany, 2000; Vol. 11.
(27) Berger, F.; Flamm, C.; Gleiss, P. M.; Leydold, J.; Stadler, P. F. Counterexamples in Chemical Ring Perception. *J. Chem. Inf. Comput. Sci.* **2004**, *44*, 323–331.
(28) Plotkin, M. Mathematical Basis of Ring-Finding Algorithms in CIDS. *J. Chem. Doc* **1971**, *11*, 60–63.
(29) Vismara, P. Union of all the minimum cycle bases of a graph. *Electron. J. Comb.* **1997**, *4*, 73–87.
(30) Downs, G. M.; Gillet, V. J.; Holliday, J. D.; Lynch, M. F. Review of ring perception algorithms for chemical graphs. *J. Chem. Inf. Comput. Sci.* **1989**, *29*, 172–187.
(31) Gutman, I.; Trinajstic, N. Graph The ory and Molecular Orbitals. XV. The Hückel rule. *J. Chem. Phys.* **1976**, *64*, 4921–4925.
(32) Jungnickel, D. *Graph, Networks and Algorithms*, 3rd ed.; Springer-Verlag: Berlin, Germany, 2007; Vol. 5.
(33) Weininger, D. SMILES, a Chemical Language and Information System. 1. Introduction to Methodology and Encoding Rules. *J. Chem. Inf. Comput. Sci.* **1988**, *28*, 31–36.
(34) Dalby, A.; Nourse, J.; Hounshell, W.; Gushurst, A.; Grier, D.; Leland, B.; Laufer, J. Description of Several Chemical Structure File Formats Used by Computer Programs Developed at Molecular Design Limited. *J.Chem. Inf. Comput. Sci.* **1992**, *32*, 244–255.
(35) Daylight Theory: SMILES. http://www.daylight.com/dayhtml/doc/theory/theory.smiles.html (accessed June 18, 2008).
(36) Grune, D.; Jacobs, C. Grammars as a Generating Device. In *Parsing Techniques − A Practical Guide*; Ellis Horwood: Chichester, England, 1990; pp 16−28.
(37) OpenSMILES Home Page. http://www.opensmiles.org (accessed June 18, 2008).
(38) Weininger, D.; Weininger, A.; Weininger, J. SMILES 2. Algorithm for Generation of Unique SMILES Notation. *J. Chem. Inf. Comput. Sci.* **1989**, *29*, 97–101.
(39) Garey, M. R.; Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*; W. H. Freeman: New York, NY, 1979.
(40) Ullmann, J. R. An Algorithm for Subgraph Isomorphism. *J. ACM.* **1976**, *23*, 31–42.
(41) Kukluk, J.; Holder, L.; Cook, D. Algorithm and Experiments in Testing Planar Graphs for Isomorphism. *J. Graph Algo. App.* **2004**, *8*, 101–104.
(42) Daylight Theory: SMARTS − A Language for Describing Molecular Patterns. http://www.daylight.com/dayhtml/doc/theory/theory.smarts.html (accessed June 18, 2008).

(43) Moore, E. The shortest path through a maze In *Theory of Switching*, Proceedings of the International Symposium on the Theory of Switching, Cambridge, MA, 2−5 April, 1957; Harvard University Press: Cambridge, MA, 1959; pp 285−292.

(44) Daylight Theory: Fingerprints. http://www.daylight.com/dayhtml/doc/theory/theory.finger.html (accessed June 18, 2008).

(45) Wermuth, G. C. Pharmacophores: Historical Perspective and Viewpoint from a Medicinal Chemist In *Pharmacophores and Pharmacophore Searches*; Langer, T., Hoffmann, R., Eds.; Wiley-VCH: Weinheim, Germany, 2006; Vol. 32, pp 3−11.

(46) Schneider, G.; Neidhart, W.; Giller, T.; Schmid, G. "Scaffold-Hopping" by Topological Pharmacophore Search: A Contribution to Virtual Screening. *Angew. Chem., Int. Ed. Engl.* **1999**, *38*, 2894–2896.

(47) Xing, L.; Glen, R. Novel Methods for the Prediction of LopP, pKa, and LogD. *J. Chem. Inf. Comput. Sci.* **2002**, *42*, 796–805.

(48) Sykora, V.; Krstajic, D.; Leahy, R.; Leahy, D. Evolution of Molecules Using Genetic Operators On Reduced Molecules. Presented at the 4th Joint Sheffield Conference on Chemoinformatics. [Online], Sheffield, United Kingdom, June 18−20, 2007; Poster T22. Sheffield Conference on Chemoinformatics. http://cisrg.shef.ac.uk/shef2007/conference.htm (accessed June 18, 2008).

(49) Chemical Descriptors Library: Table of Content. http://cdelib.sourceforge.net/doc/index.html (accessed June 18, 2008).

CI800135H