Journal of Bioinformatics and Computational Biology

AN EMPIRICAL STUDY OF THE UNIVERSAL CHEMICAL KEY ALGORITHM FOR ASSIGNING UNIQUE KEYS TO CHEMICAL COMPOUNDS

ROBERT GROSSMAN[i]
Laboratory for Advanced Computing
University of Illinois at Chicago
Chicago, IL 60607, USA
grossman@uic.edu

PAVAN KASTURI
Laboratory for Advanced Computing
University of Illinois at Chicago
Chicago, IL 60607, USA
pavan@lac.uic.edu

DONALD HAMELBERG
Laboratory for Advanced Computing
University of Illinois at Chicago
Chicago, IL 60607, USA
don@lac.uic.edu

BING LIU
Department of Computer Science
University of Illinois at Chicago
Chicago, IL 60607, USA
liub@cs.uic.edu

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

In this paper, we introduce an algorithm that assigns an essentially unique key called the Universal Chemical Key (UCK) to molecular structures. The molecular structures are represented as labeled graphs whose nodes abstract atoms and whose edges abstract bonds. The algorithm was tested on 236,917 compounds obtained from the National Cancer Institute (NCI) database of chemical compounds. On this database, the UCK algorithm assigned unique keys for chemicals with distinct molecular structures.

Keywords: Molecular structure, graph, keys, universal chemical key (UCK)

1. Introduction

Chemical compounds usually have several common names. Although unique identifiers attached to chemical compounds would be useful for a variety of purposes, there is no consensus about how to do this. Currently most nomenclatures for chemical compounds either do not provide unique keys or the unique keys provided are based upon convention, such as when the compound was entered into a database. For this reason, determining whether a compound has been entered into a database two or more times or comparing compounds across databases is difficult.

As an example, consider the compound Testosterone, whose structure is illustrated in Figure 1. In the National Cancer Institute (NCI) database, this molecule has 54 different names associated with it and is assigned a unique id of 9700. On the other hand, its Chemical Abstract Services (CAS) id is 58-22-0. Because of examples like these, it would be very useful

to have a unique key that is derived from the structural features of the compound. Using such a key, properties of a chemical contained in a database in one location could be combined with properties of the same chemical compound contained in a database in another location automatically. With the increasing use of distributed infrastructures for computing, such as data grids and web service-based platforms, having universal chemical keys that can be used to combine distributed data about chemical compounds is of growing importance. Indeed, we have used the algorithm introduced here to build distributed data web applications for docking chemical compounds in proteins from the Protein Data Bank (PDB)[17].

Our algorithm for computing what we call a Universal Chemical Key or UCK is based upon abstracting the chemical compound as a labeled graph, with atoms represented by nodes and bonds represented by edges. The nodes are labeled with the symbols corresponding to the atoms they represent. Note that two labeled graphs representing molecular structures are isomorphic if they are labeled using the same labels and can be mapped onto each other such that the labels of nodes or atoms and edges or bonds are conserved.

In this paper, we introduce an algorithm that given a labeled graph representing a chemical compound produces a long string, which is its UCK. This string has the properties:

i) Chemical compounds associated with the same-labeled graph are identical and produce the same UCK.
ii) UCKs of different labeled graphs are almost always different.

Since in general determining whether two graphs are isomorphic is NP-hard, we cannot expect to find a fast algorithm assigning distinct keys to graphs in general.[1] On the other hand, we show that in practice, on large collections of chemical compounds such as the NCI database, our UCK algorithm does have this property. Since the UCK strings can be quite long, we associate a shorter string using a standard hashing algorithm called MD5.[2] Although the MD5 hash is not guaranteed to be unique, in practice it usually is.

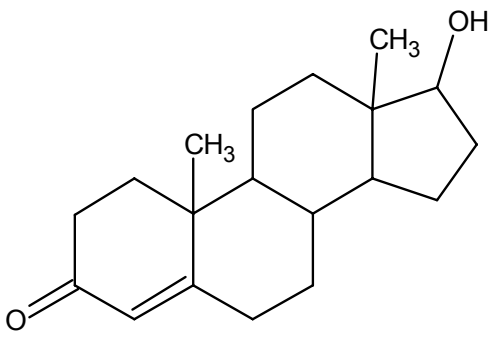


Figure 1. Structural formula of Testosterone, C19H28O2. Testosterone has the NSC id of 9700 and the CAS id 58-22-0. Some of the other names of Testosterone are 17-hydroxyandrost-4-en-3-one, Androlin, Cristerona T, and Homosteron.

For some applications, it would be desirable for similar chemical compounds to be hashed to similar UCKs. The algorithm described here does not have this property. Constructing hashes with this property is usually quite difficult and outside the scope of this paper. We also note that the UCK algorithm was not designed in such a way that the graph can be recovered from it.

Section 2 contains related work. Section 3 contains the UCK algorithm and Section 4 describes some of our experimental results using the National Cancer Institute (NCI) database of chemical compounds.

2. Related work

Numbering and ordering of atoms and groups of atoms of molecular structures have always been done by organic chemists, and there are several different systems of naming compounds. The International Union of Pure and Applied Chemistry (IUPAC) nomenclature rules[3] are the most widely used. However, these rules only work effectively for very small molecules and generally are inconsistent, hard to understand, and easily cause mistakes.[4, 5, 6]

SMILES[18] is a string of characters that encapsulate the atom and bond information of a molecule. One can rebuild the molecule structure from a SMILES string. SMILES is very versatile and widely used. It can be interpreted as a string of characters or as a molecular graph. It can be used for substructure matching or searching. Unfortunately, SMILES are not unique by nature. One compound can have more than one SMILES representation. For a simple example, 1-methyl-3-bromo-cyclohexene-1 can be represented by the two following SMILES strings[18]:

CC1=CC(Br)CCC1
CC1=CC(CCC1)Br

In contrast to SMILES, the UCK algorithm described below does assigns unique chemical ids to chemical compounds. On the other hand, the chemical ids assigned are quite long and are designed to be used by software and not by chemists directly.

The idea of using graph theory to assign chemical ids was put forward some twenty years ago,[7, 8] but never really caught the attention of chemists. Recently, the International Union of Pure and Applied Chemistry (IUPAC) established a project to develop unique ids for chemical compounds[9] that is based in part on graph theory. However, the details are yet to be published. The documentation and algorithm will be published at the end of the project. Their aim is to generate a unique key from a graphical input of structural information of known and as yet unknown compounds. The keys, known as IUPAC Chemical Identifiers, are alphanumeric text strings that can lead back to the structure of the compound and that can be digitized to be used in printed and electronic information.

If a chemical compound is viewed as a graph, then the problem of assigning unique chemical ids is closely related to the problem of determining whether two graphs are isomorphic. It is important to note that general graphs can be much more complicated than the graphs that arise from molecular structures (bond graphs), and consequently determining whether two graphs are isomorphic is in general much more complex than determining whether two bond graphs are isomorphic.

Observe that if the nodes of a bond graph can be ordered 1, 2, 3, …, then reading off the corresponding labels of the graph in the same order gives rise to a string associated with bond graph that can be used as the chemical id. This is an example of what are sometimes called canonical orderings or canonical labels[9]. Since a bond graph does not come with such an ordering, a canonical ordering must be generated externally.

One method for assigning canonical labels to a bond graph relies upon the incidence or adjacency matrix. Recall that the adjacency matrix a_{ij} is the matrix of 0's and 1's defined by a_{ij} is 1 if there is an edge connecting node i and node j and is zero otherwise. Note that re-ordering the rows of the adjacency matrix generates an isomorphic graph since it simply corresponds to re-ordering the nodes. Since the adjacency matrix consists of 1's and 0's, each row may be interpreted as a binary number. In this way, an adjacency matrix gives rise to a tuple of binary numbers, with one element in the tuple coming from each row of the adjacency matrix. One can then define a partial order on adjacency matrices by lexicographically ordering these tuples of binary numbers. Given a graph, one can consider all possible adjacency matrices associated with the graph and select the one which is maximal in this ordering. Prokurowski[9] has studied canonical orderings generated this way using the incidence matrix. Tinhofer et. al.[13] has studied canonical orders generated this way using the adjacency matrix.

A significant limitation with this approach is that two non-isomorphic bond graphs can give rise to the same chemical ids. Another difficulty is that computing canonical orderings can be very expensive, since the number of different node orderings varies exponentially with the number of nodes. To overcome this difficulty, Brendon D. McKay[14, 15] has introduced a back-tracking approach which can significantly reduce the cost of computing canonical labels. He and his colleagues have developed a program called "nauty" which can canonically label graphs.

In contrast, the UCK algorithm described below does not rely on using incidence or adjacency matrices to produce canonical labels. Instead, the UCK algorithm re-labels each node of the graph by examining the labeled paths originating from each node. An advantage of this approach is that these local paths capture some of the local properties of the graph. Based upon our empirical studies, this approach is more likely to lead to unique chemical ids than an approach which relies simply on a canonical ordering of the nodes. It is also important to note that the UCK algorithm, unlike nauty, is not designed to be efficient on general graphs, but simply the types of graphs which occur as bond graphs.

Another approach is to directly exploit the structure of bond graphs and to use a heuristic based approach to assign chemical ids. There is a fair amount of work that takes this approach. For example, Read[6] first breaks a bond graph into rings and side chains and then selects a special walk along the ring to provide a canonical number of the atoms in the ring. As another example, Klin[4] considers a class of compounds associated with benzene rings and creates chemical ids based upon extracting cycles of maximum length. In contrast, the UCK algorithm described below applies to any chemical graph and does not depend upon heuristics.

3. Computational methods and algorithms

3.1 Overview

We begin with an overview of the algorithm. The algorithm is based on abstracting the molecule as a labeled graph: Each atom in the molecule is represented as a node in the graph and labeled with the symbol for the atom. Two vertices are connected with an edge in case there is a covalent bond between them of any type.

The algorithm consists of three steps. In Step 2, a sequence of longer and longer labels $\lambda^{(d)}$ (a) is assigned to each vertex a, with the sequence reflecting the structure of larger and larger neighborhoods of the vertex. In Step 2, a long string is created by concatenating strings of the form

$$\mu^{(d)}(a,b) = \text{the concatenated string } [\ \lambda^{(d)}(a)\ n\ \lambda^{(d)}(b)\ ],$$

for each vertex a and b in the graph, where n is the length of the shortest path between a and b. In Step 3, the UCK is formed by concatenating all the $\mu^{(d)}(a,b)$.

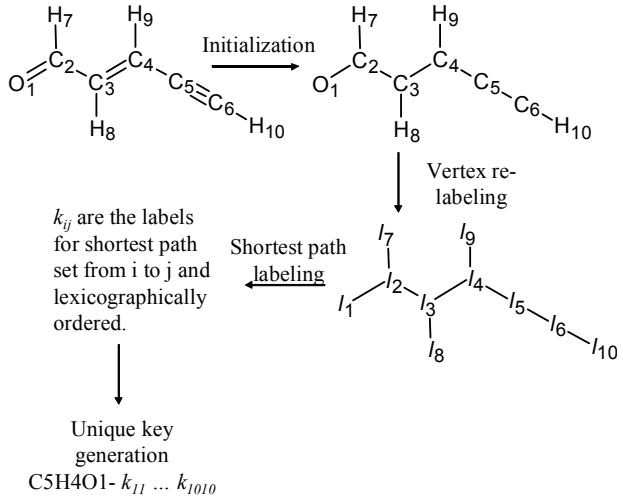


Figure 2. An example showing the procedure to generate a unique string for a given molecule. The subscripts represent vertex numbers.

Here is part of an example. A more detailed description of the algorithm follows. A schematic representation of a molecule that may or may not exist is shown in Figure 2 with single, double and triple bonds and shows an overview of the algorithm. We represent the double and triple bonds as single bonds, so that it can be represented as a graph G = (V, E) where the vertex set V = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} and the edge set E = {(1,2), (2,3), (3,4), (4,5), (5,6), (2,7), (3,8), (4,9), (6,10)}. The list <O, C, C, C, C, C, H, H, H, H> is the label list for elements corresponding to V. Observe that though the connectivity of the second vertex is different from that of the third to the sixth they all have the same labels.

3.2 Algorithm

3.2.1 *The labeled bond graph G*

To begin, we form the labeled graph G = (V, E), where the vertex set V consists of the atoms in the molecule. We label the vertices with the labels of the corresponding atoms. Two vertices are connected with an edge in case the corresponding atoms have a chemical bond (of any type) between them.

3.2.2. *Labeling nodes with the map* $\lambda^{(d)}$ *using local paths of depth d*

In the first step, examine local paths of depth d and use the information to re-label each node of the bond graph G. To do this, for each depth d, we define a label map $\lambda^{(d)}$ inductively from $\lambda^{(d-1)}$ as follows:

1. Define $\lambda^{(0)}$ (b) = label (b), where b is a node in Graph G.
2. For a node a with children $b_1, \ldots, b_k$, compute the labels $\lambda^{(d-1)}(b_1), \ldots, \lambda^{(d-1)}(b_k)$
3. Lexicographically order the labels $\lambda^{(d-1)}(b_1), \ldots, \lambda^{(d-1)}(b_k)$ and concatenate to produce the string $[\lambda^{(d-1)}(b_{i1}) \ldots \lambda^{(d-1)}(b_{ik})]$.
4. Define $\lambda^{(d)}(a) = [\text{label (a)}\ \lambda^{(d-1)}(b_{i1}) \ldots \lambda^{(d-1)}(b_{ik})]$

We use the map $\lambda^{(d)}$ to re-label each node in G. For the empirical studies described in Section 4, depths d=2 and d=3 are used.

3.2.3. *Labeling pairs of nodes with the map* $\mu^{(d)}$

In the second step, for each pair of vertices a and b in the graph, we define

$$\mu^{(d)}(a,b) = \text{the concatenated string } [\ \lambda^{(d)}(a)\ n\ \lambda^{(d)}(b)\ ],$$

where n = length of shortest path from a to b.

3.2.4. *Defining the UCK* $\mu^{(d)}(G)$

In the third and final step, we use the map the $\mu^{(d)}(a,b)$ to construct a string which labels the graph as a whole. We do this by computing $\mu^{(d)}(a,b)$ for all pairs of nodes, lexicographically ordering the resulting strings, and then concatenating them:

$\mu^{(d)}(G)$ = Lexicographically order the collection of strings $\{\ \mu^{(d)}(a,b) \mid a \in V, b \in V \text{ of } G\ \}$ and then concatenate

3.3 Variations of the UCK Algorithm

For our experimental studies, we used two simple variations of the UCK defined above. First, since the second label map $\lambda^{(2)}$ for the children of a node a always contains the label a itself, we simply remove one occurrence of the label a from each label $\lambda^{(2)}$ before lexicographically ordering the strings $\lambda^{(d-1)}(b_1), \ldots, \lambda^{(d-1)}(b_k)$ in Step 1 above. In other words, we replaced Step 1 above with the variant:

Variant of map $\lambda^{(d)}$: Remove one occurrence of a from each of the strings $\lambda^{(d-1)}(b_1), \ldots, \lambda^{(d-1)}(b_k)$ and then lexicographically order the resulting strings to produce the string $[\lambda^{(d-1)}(b_{i1}) \ldots \lambda^{(d-1)}(b_{ik})]$.

This slightly reduces the length of the UCK.

Second, searches in databases of UCKs can be made much faster by excluding any UCK with different chemical formulas. For this reason, we pre-pend the chemical formula to each UCK so that $\mu^{(d)}(G)$ was defined as follows:

$\mu^{(d)}(G)$ = Chemical formula-Lexicographically order the collection of strings $\{\ \mu^{(d)}(a,b) \mid a \in V, b \in V \text{ of } G\ \}$ and then concatenate

3.4 Example

Let us consider benzoic acid ($C_7H_6O_2$) shown in Figure 3.

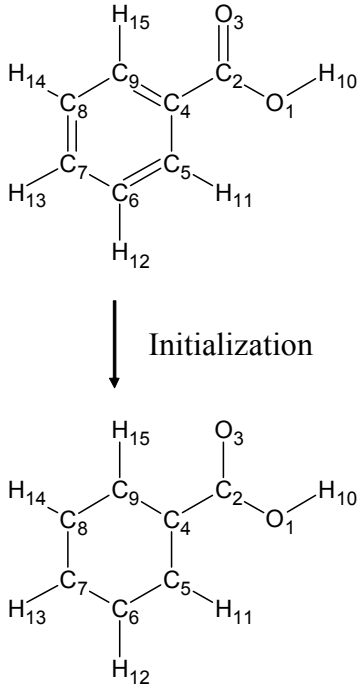


Figure 3. Structure of benzoic acid (NCI number = 149). The subscripts represent vertex numbers.

The new recursively generated labels for each vertex for the example of benzoic acid is shown in Table 1 below. The table contains labels using the variation of the algorithm described in Section 3.2.6

Table 1. Showing the recursive generation of the labels using the λ and μ maps.

Vertex Number	$\lambda^{(0)}$	$\lambda^{(1)}$	$\lambda^{(2)}$
1	O	OCH	OCCOH
2	C	CCOO	CCCCOOH
3	O	OC	OCCO
4	C	CCCC	CCCHCCHCOO
5	C	CCCH	CCCCCCHH
6	C	CCCH	CCCHCCHH
7	C	CCCH	CCCHCCHH
8	C	CCCH	CCCHCCHH
9	C	CCCH	CCCCCCHH
10	H	HO	HOC
11	H	HC	HCCC
12	H	HC	HCCC
13	H	HC	HCCC
14	H	HC	HCCC
15	H	HC	HCCC

$\mu^{(2)}$ (a,b)	String
$\mu^{(2)}$ (1,2)	OCCOH1CCCCOOH
$\mu^{(2)}$ (1,3)	OCCOH2OCCO

$\mu^{(2)}$ (2,3)	CCCCOOH1OCCO
$\mu^{(2)}$ (2,15)	CCCCOOH3HCCC
$\mu^{(2)}$ (4,13)	CCCHCCHCOO4HCCC
$\mu^{(2)}$ (6,4)	CCCHCCHH2CCCHCCHCOO
$\mu^{(2)}$ (15,14)	HCCC3HCCC
…	…

The UCK $\mu(G)$ is shown below:

C7H6O2-CCCCCCHH0CCCCCCHHCCCCCCHH0CCCCCCHHCCC
CCCHH1CCCHCCHCOOCCCCCCCHH1CCCHCCHCOOCCCCCCCHH1CCCHCCHHCCCCCCCHH1CCCHCCHHCCCCCCCHH1HCCCCCCCCCCHH1
HCCCCCCCCCHH2CCCCCCHHCCCCCCHH2CCCCCCHHCCCCCCHH2CCCCOOHCCCCCCHH2CCCCOOHCCCCCCHH2CCCHCCHHCCCCC
CHH2CCCHCCHHCCCCCCHH2HCCCCCCCCCCHH2HCCCCCCCCCCHH3CCCHCCHHCCCCCCHH3CCCHCCHHCCCCCCHH3HCCCCCCCCCH
H3HCCCCCCCCCHH3HCCCCCCCCCHH3HCCCCCCCCCHH3OCCOCCCCCCHH3OCCOCCCCCCHH3OCCOHCCCCCCHH3OCCOHCCCCCC
HH4HCCCCCCCCCHH4HCCCCCCCCCHH4HOCCCCCCCHH4HOCCCCCOOH0CCCCOOHCCCCOOH1CCCHCCHCOOCCCCOOH1OCCOCC
CCOOH1OCCOHCCCCOOH2CCCCCCHHCCCCOOH2CCCCCCHHCCCCOOH2HOCCCCCOOH3CCCHCCHHCCCCOOH3CCCHCCHHCCCCO
OH3HCCCCCCCOOH3HCCCCCCCOOH4CCCHCCHHCCCCOOH4HCCCCCCCOOH4HCCCCCCCOOH5HCCCCCCHCCHCOO0CCCHCCHCO
OCCCHCCHCOO1CCCCCCHHCCCHCCHCOO1CCCCCCHHCCCHCCHCOO1CCCCOOHCCCHCCHCOO2CCCHCCHHCCCHCCHCOO2CCCH
CCHHCCCHCCHCOO2HCCCCCCHCCHCOO2HCCCCCCHCCHCOO2OCCOCCCHCCHCOO2OCCOHCCCHCCHCOO3CCCHCCHHCCCHCCH
COO3HCCCCCCHCCHCOO3HCCCCCCHCCHCOO3HOCCCCHCCHCOO4HCCCCCCHCCHH0CCCHCCHHCCCHCCHH0CCCHCCHHCCCHC
CHH0CCCHCCHHCCCHCCHH1CCCCCCHHCCCHCCHH1CCCCCCHHCCCHCCHH1CCCHCCHHCCCHCCHH1CCCHCCHHCCCHCCHH1CC
CHCCHHCCCHCCHH1CCCHCCHHCCCHCCHH1HCCCCCCHCCHH1HCCCCCCHCCHH1HCCCCCCHCCHH2CCCCCCHHCCCHCCHH2CCC
CCCHHCCCHCCHH2CCCHCCHCOOCCCHCCHH2CCCHCCHCOOCCCHCCHH2CCCHCCHHCCCHCCHH2CCCHCCHHCCCHCCHH2HCCCC
CCHCCHH2HCCCCCCHCCHH2HCCCCCCHCCHH2HCCCCCCHCCHH2HCCCCCCHCCHH2HCCCCCCHCCHH3CCCCCCHHCCCHCCHH3CC
CCCCHHCCCHCCHH3CCCCOOHCCCHCCHH3CCCCOOHCCCHCCHH3CCCHCCHCOOCCCHCCHH3HCCCCCCHCCHH3HCCCCCCHCCHH
3HCCCCCCHCCHH3HCCCCCCHCCHH4CCCCOOHCCCHCCHH4HCCCCCCHCCHH4HCCCCCCHCCHH4OCCOCCCHCCHH4OCCOCCCHCC
HH4OCCOHCCCHCCHH4OCCOHCCCHCCHH5HOCCCCHCCHH5HOCCCCHCCHH5OCCOCCCHCCHH5OCCOHCCCHCCHH6HOCHCCC0H
CCCHCCC0HCCCHCCC0HCCCHCCC0HCCCHCCC0HCCCHCCC1CCCCCCHHHCCC1CCCHCCHHHCCC1CCCHCCHHH
CCC1CCCHCCHHHCCC2CCCCCCHHHCCC2CCCCCCHHHCCC2CCCHCCHCOOHCCC2CCCHCCHCOOHCCC2CCCHCCHHHCCC2CCCHCC
HHHCCC2CCCHCCHHHCCC2CCCHCCHHHCCC2CCCHCCHHHCCC2CCCHCCHHHCCC3CCCCCCHHHCCC3CCCCCCHHHCCC3CCCCCCH
HHCCC3CCCCCCHHHCCC3CCCCOOHHCCC3CCCCOOHHCCC3CCCHCCHCOOHCCC3CCCHCCHCOOHCCC3CCCHCCHHHCCC3CCCHCC
HHHCCC3CCCHCCHHHCCC3CCCHCCHHHCCC3HCCCHCCC3HCCCHCCC3HCCCHCCC3HCCCHCCC3HCCCHCCC3HCCCHCCC3HCCCH
CCC3HCCCHCCC4CCCCCCHHHCCC4CCCCCCHHHCCC4CCCCOOHHCCC4CCCCOOHHCCC4CCCHCCHCOOHCCC4CCCHCCHHHCCC4C
CCHCCHHHCCC4HCCCHCCC4HCCCHCCC4HCCCHCCC4HCCCHCCC4HCCCHCCC4HCCCHCCC4HCCCHCCC4HCCCHCCC4OCCOHCCC4
OCCOHCCC4OCCOHHCCC4OCCOHHCCC5CCCCOOHHCCC5HCCCHCCC5HCCCHCCC5HCCCHCCC5HCCCHCCC5HOCHCCC5HOCHCCC
5OCCOHCCC5OCCOHCCC5OCCOHHCCC5OCCOHHCCC6HOCHCCC6HOCHCCC6OCCOHCCC6OCCOHHCCC7HOCHOC0HOCHOC1OCCO
HHOC2CCCCOOHHOC3CCCHCCHCOOHOC3OCCOHOC4CCCCCCHHHOC4CCCCCCHHHOC5CCCHCCHHHOC5CCCHCCHHHOC5HCCCH
OC5HCCCHOC6CCCHCCHHHOC6HCCCHOC6HCCCHOC7HCCCOCCO0OCCOOCCO1CCCCOOHOCCO2CCCHCCHCOOOCCO2OCCOHOC
CO3CCCCCCHHOCCO3CCCCCCHHOCCO3HOCOCCO4CCCHCCHHOCCO4CCCHCCHHOCCO4HCCCOCCO4HCCCOCCO5CCCHCCHHOCC
O5HCCCOCCO5HCCCOCCO6HCCCOCCOH0OCCOHOCCOH1CCCCOOHOCCOH1HOCOCCOH2CCCHCCHCOOOCCOH2OCCOOCCOH3CC
CCCCHHOCCOH3CCCCCCHHOCCOH4CCCHCCHHOCCOH4CCCHCCHHOCCOH4HCCCOCCOH4HCCCOCCOH5CCCHCCHHOCCOH5HCC
COCCOH5HCCCOCCOH6HCCC

Because of the length of the generated string is so large, we used the *md5* algorithm to generate a *hex digest* for the string. The *hex digest* is 39BF9B334B172E4E71E76B93C830B47E. From this point on, we will only use *hex digest* of *md5* to represent the unique string.

4. Results

We implemented the algorithm in the C programming language and tested it on 236,917 chemical compounds in the NCI database of chemical compounds. The input was a trimmed version of a PDB file that contains the atomic symbols and their spatial coordinates. We calculated the bond information and used this to construct a graph for each compound as described in the section above. We ran our UCK algorithm using depths of d=2 and d=3. The output was then piped to the md5 algorithm to generate the UCK hex digest.

The 236,917 chemical compounds include variants of the same molecule. By variants of the same molecule we mean data sets (which represent chemical compounds) that differ in the ordering of the atoms and/or their spatial orientation. The NCI database has many molecules whose structures are very similar but are different molecules. There are also molecules that have the same molecular formula but different structural formulas.

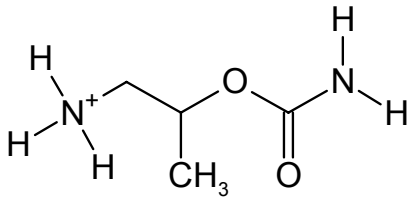
The results are summarized in Table 2. Note that UCK algorithm identifies 33,527 chemical compounds which occur more than once in the NCI database – we know this since the program we wrote automatically identifies two compounds which map to the same UCK. For fixed depth d, two types of errors are possible. First, distinct compounds may be mapped to the same UCK – a false positive. Second, the same compound occurring more than once in the database may be mapped to different UCKs – a false negative. For example, as detailed in Table 2, three pairs of distinct compounds are mapped to the same UCK for depth d=2, but to different UCKs for depth d=3. To identify false positives we examined all 33,527 compounds with two or more entries manually. This is somewhat error prone and in an earlier version of the paper[16], we missed the six compounds that require a depth of d=3 in order to have distinct UCKs. For an example, see Table 6 and Figure 7. To rule out false negatives, we checked those 203,390 chemical compounds which had the same chemical formula, which is the prefix of our UCK, but different UCKs to make that the chemical compounds were in fact different. They were.

Table 2. Summary of the UCK algorithm to generate unique chemical keys applied to the NCI database of chemical compounds.

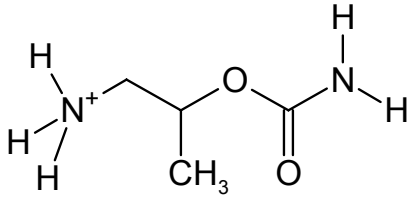
Description	***Number***	***Remarks***
Total number of chemical compounds.	236,917	Includes some compounds with duplicate entries.
Number of chemical compound with single entry.	203,390	All gave unique UCK.
Number of chemical compound with two or more entries.	33,527	The UCK algorithm gave unique labels to distinct compounds. The UCK algorithm gave the same label to the same compound occurring in multiple entries.
Number of compounds that required depth d=2 for the UCK algorithm.	236, 911	The UCKs for these compounds were computed with d=2.
Number of compounds that required depth d=3 for the UCK algorithm.	6	The UCKs for these compounds were computed with d=3.

On the NCI database, the UCK satisfied our two requirements described in Section 1: i) the algorithm produced the same key for variants of the same molecule and ii) different compounds produced different keys. The algorithm is invariant to changes in input ordering of atoms and rigid transformations. We were also able to recognize molecules that have more than one entry in the database. For example, consider two molecules with NCI numbers 30783 and 206631 (Figure 4). Though they have different NCI numbers they are the same molecule.

Two different entries in the NCI database are illustrated below in Figure 4. Table 3 contains the corresponding UCKs. The two data sets have different ordering of atoms and slightly different spatial orientation. Also shown in Table 3 are the hex digest for two molecules with NCI numbers 91771 and 97338 that are the same but different in their respective conformations.



30783



206631

Figure 4. Structure of 1-methyl-2-(trimethyl-λ^5-aznyl)ethyl carbamate with different NSC numbers.

Table 3. Some examples of different entries from the NCI database which are sent to the same UCK and are in fact the same chemical compound.

NSC Number	***Formula***	***md5 hex digest generation of UCK***
30783	$C_7H_{17}N_2O_2$	02994BC7A283073ED9C 1730E2F37EFCD
206631	$C_7H_{17}N_2O_2$	02994BC7A283073ED9C 1730E2F37EFCD
91771	$C_{38}H_{42}N_2O_6$	5C0F9A8F0ECC0BAF32 CBCA62DA571F42
97338	$C_{38}H_{42}N_2O_6$	5C0F9A8F0ECC0BAF32 CBCA62DA571F42

The algorithm is sensitive to changes in connectivity even at the remotest portions of the molecule, which makes it very effective in detecting different chemical compounds that are very similar. Figure 5 shows two pairs of compounds that are quite similar. Table 4 gives the results of our algorithm and illustrates the efficiency of the algorithm for molecules with small and difficult to detect changes in structures. We can observe that their UCK keys are different.

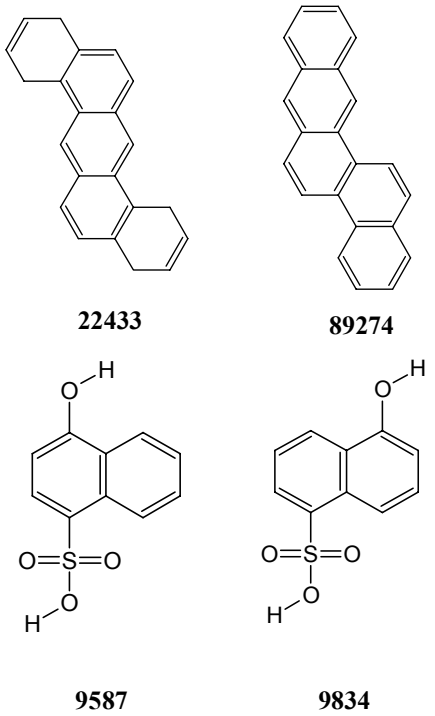


Figure 5. Structures of dibenzo[a,h]anthracene (22433) and benzo[b]chrysene (89274), 4-hydroxy-1-naphthalenesulfonic acid (9587) and 5-hydroxy-1-naphthalenesulfonic acid (9834).

Table 4. Some examples of entries from the NCI database which have the same chemical formula but are in fact different chemical compounds. Note that the UCKs are different as desired.

NSC Number	***Formula***	***md5 hex digest generation of UCK***
22433	$C_{22}H_{14}$	A15F6359F7AC44C1A90C0 F90598664B4
89274	$C_{22}H_{14}$	7A4A4AF922300C12704238 10B748FAF5
9587	$C_{10}H_8O_4S$	6BEAF3C856A2C1318F306 DDB0E7F888A
9834	$C_{10}H_8O_4S$	64373D3DB8329F56B6AFA E59F07AAFC3

Figure 6 shows a pair of compounds – 5-methoxy-2,2,7,8-tetramethyl-2H,6H-pyrano[3,2-g]chromen-6-one and 5-methoxy-2,3,8,8-tetramethyl-4H,8H-pyrano[2,3-f]chromen-4-one – which have the same molecular formula ($C_{17}H_{18}O_4$). The two compounds have very similar structures, with subtle differences. Our algorithm distinguishes these two compounds and gives them different UCKs. Table 5 summarizes these results.

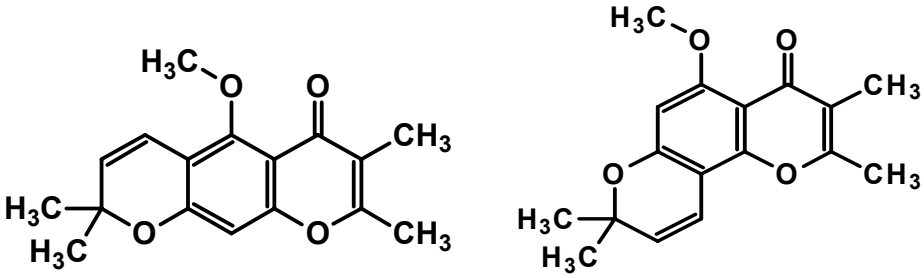


Figure 6. Structures of 5-methoxy-2,2,7,8-tetramethyl-2H,6H-pyrano[3,2-g]chromen-6-one (682322) and 5-methoxy-2,3,8,8-tetramethyl-4H,8H-pyrano[2,3-f]chromen-4-one (682323).

Table 5. Some entries from the NCI database for two compounds that have very similar but different structures. Note that the UCKs are different.

NSC Number	*Formula*	*md5 hex digest generation of UCK*
682322	$C_{17}H_{18}O_4$	132020EB36654634EB960CF003547644
682323	$C_{17}H_{18}O_4$	098900E618644A0F508569D3BE0179F6

The UCK algorithm requires depth d = 2 for most cases but requires a higher depth of d = 3 for six cases in our study. Table 6 show an example of one of the three pairs of compounds that requires a higher depth of d=3. Figure 7 contains the corresponding molecular structures.

Table 6. Some entries from the NCI database for NSC IDs 2212 and 5268 and the hex digest (last column) generated by our algorithm for two compounds that have very similar but different structures.

NSC Number	*Common Name*	*md5 of UCK*
2212	anthra[9,1,2-cde]benzo[rst]pentaphene-5,10-dione	4FD3362DD587F814977D9CF7CCA1CB8A
5268	benzo[rst]phenanthro[10,1,2-cde]pentaphene-9,18-dione	0AAE922FD7CC5532941472C2D3255904

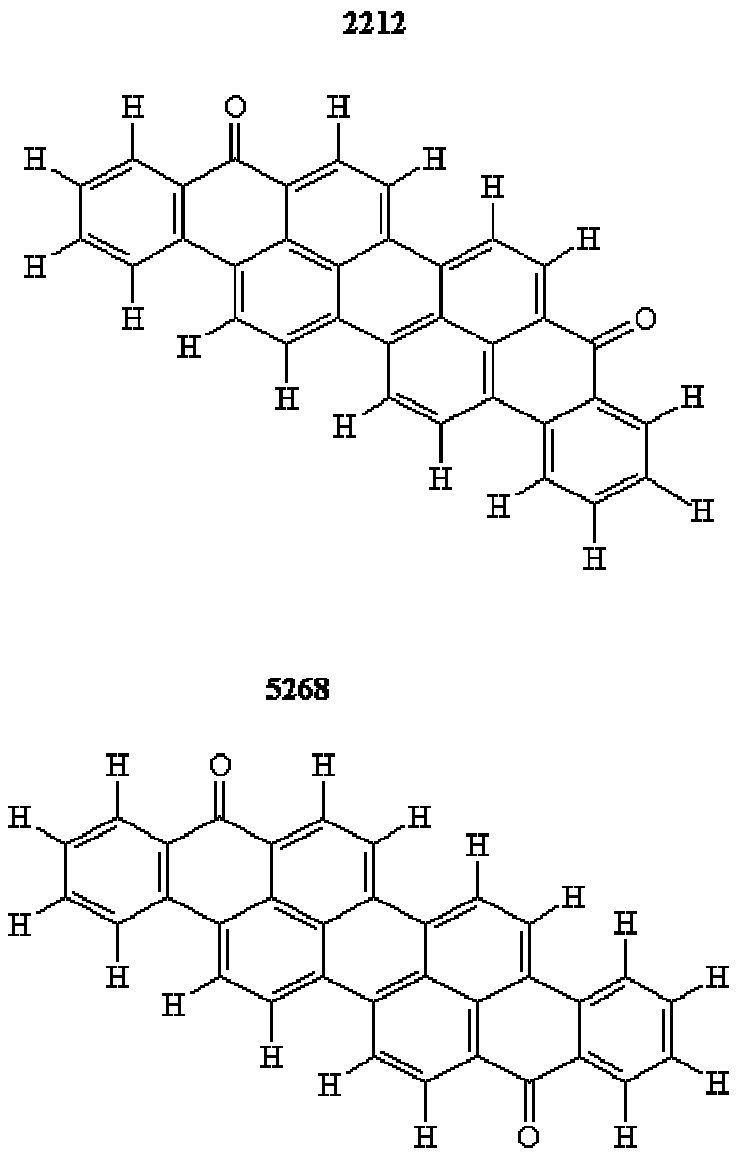


Figure 7. Structures of anthra[9,1,2-cde]benzo[rst]pentaphene-5,10-dione (2212) and benzo[rst]phenanthro[10,1,2-cde]pentaphene-9,18-dione (5268)

On average, we could compute about 20 UCKs per second. We haven't optimized the implementation yet and expect an optimized implementation could be substantially faster. The largest compound in the database contained 579 atoms and required about 5 seconds to generate its UCK. Additional performance information is presented in Tables 7 and 8. The tests were carried out on a Dell machine with Dual Xeon 2.4 GHz processors with hyper threading, 2 gigabyte of RAM, and approximately 236 gigabyte of disk space on RAID-5 and running Red Hat Linux 8.0 operating system.

Table 7. Properties and execution time of the large molecule studied.

Largest Molecule	
NCI Number	57300
# Atoms	579
Formula	$C_{166}H_{328}N_2O_{83}$
md5 hex digest generation of UCK	CE6892FDEBE05614AAC08560A5D4AE8B
Time taken for UCK generation	5.204 seconds
Total length of the key (not md5)	5278576 characters

Table 8. Total time required to process the entire data set.

# Compounds	***Time taken (approx.)***
236,917	12478.42 sec (3.466 hrs)

5. Summary and Conclusions

We have developed an algorithm called the UCK algorithm that generates unique keys for a wide variety of chemical compounds. The UCK algorithm views molecular structures as labeled graphs. The atoms are represented as the vertices and the bonds as the edges. The algorithm was experimentally tested on 236,917 compounds from the NCI database, and generated unique keys for all the uniquely identified structures. We call these keys Universal Chemical Keys or UCKs. We have used the UCKs to build distributed web-service based bioinformatics applications with data pulled from multiple distributed databases. Without a unique key such as the UCK, the application would have no way of knowing whether two distributed chemicals were the same or not.

The UCK algorithm depends only upon the structure of the labeled graph. Distinct labeled graphs give rise to distinct UCKs. On the other hand, two different labeled graphs could, in theory, give rise to the same UCK. This is a calculated trade-off. Distinguishing arbitrary labeled graphs is NP-hard and hence a fast deterministic algorithm cannot be expected. On the other hand, chemical compounds give rise to a restricted class of labeled graphs — it is likely that our algorithm can be proven to be unique on various restricted classes of chemical compounds. We plan to investigate these types of results and to verify experimentally the properties of our UCK algorithm on additional databases of chemical compounds.

In particular, in this study the UCK algorithm uses a depth d = 2 or 3 for labeling the vertices. Using different depths and dynamically assigning the depth depending upon the compound gives rise to UCKs that are more expensive to compute but stronger. We plan on investigating these trade-offs in future work.

To summarize, the UCK algorithm is a fast and effective algorithm that provides intrinsic and unique keys for a wide class of commonly occurring chemical compounds.

6. References

1. Michael R. Garey and David S. Johnson, Computers and Intractability, Freeman, 1979.
2. R. Riverst. *The MD5 message digest algorithm*, RFC1321, 1992.
3. IUPAC, *Nomenclature of Organic Chemistry*, Pergamon Press: Oxford, 1979.
4. M. H Klin, O. V. Lebedev, T. S. Pivina and N. S. Zefirov, "Nonisomorphic cycles of maximum length in a series of chemical graphs and the problem of application of IUPAC nomenclature rules", *MATCH 27*, 133-151 (1992).
5. R. C. Read, "The Coding of various Kinds of unlabeled Trees", *Graph Theory and Computing*, Academic Press, New York, 1972.
6. R. C. Read, "A new system for the design of chemical compounds. Coding of cyclic compounds", *Journal of Chem. Inf. and Comp. Sci.,* 25, 116-128 (1985).
7. M. Randic, "On recognition of identical graphs representing molecular topology", *J. Chem. Phys*, 60, 3920-3928 (1974).
8. A. Prokurowski, "Search for unique incidence matrix of a graph", *BIT*, 14, 209-226 (1974).
9. *Chemistry International*, 23(3), 2001.
10.V. L. Arlazarov, I. I. Zuev, A. V. Uskov and I. A. Faradzev, "An algorithm for the reduction of finite non-oriented graphs to canonical form", *Zh. Vycheil. Mat. Mat. Fiz.*, 14(3), 737-743 (1974).
11. T. Beyer and A. Proskurowski, "Symmetries in graph coding problem", *Proc. NW76 ACM/CIPC Pac. Symp*, 198-203 (1976).
12. C. Bohm and A. Santolini, "A quasi-decision algorithm for the p-equivalence of two matrices", *ICC BULLETIN*, 3(1), 57-69 (1964).
13. G. Tinhofer and M. Klin, "Algebraic combinatorics in mathematical chemistry. Methods and algorithms. III. Graph Invariants and Stabilization Methods", *Technical Report, Technische Universital Munchen, TUM-M9902*, 1999.
14. B. D. McKay, "Computing automorphism and canonical labeling of graphs", *International Conference on Combinatorial Mathematics*, Canberra , Lecture Notes in Mathematics 686, Springer-Verlag, 223-232 (1977).
15. B. D. McKay, "Practical Graph Isomorphism", *Congressus Numerantium*, 30, 45-87 (1981).
16. Robert Grossman, Donald Hamelberg, Pavan Kasturi, and Bing Liu, "Experimental Studies of the Universal Chemical Key (UCK) Algorithm on the NCI Database of Chemical Compounds", *Proceedings of the 2003 IEEE Computer Society Bioinformatics Conference (CSB 2003),* IEEE Computer Society, Los Alamitos, California, pages 244-250.
17. Donald Hamelberg, Pavan K. Kasturi, and Robert L. Grossman, "Data Webs for Bioinformatics Data," *Information Sciences*, to appear.
18. *Smiles Tutorial*, retrieved from http://www.daylight.com/smiles on October 10, 2003

---

[i] Robert Grossman is also with Open Data Partners.