Computerized Isomer Enumeration of the Alkane Series

Kevin Ballard

Advisor M. Eicher and Mentor N. Preyer

Senior Thesis for the Academic Magnet High School

Revised February 23, 2003, Version 1.1

Table of Contents

Abstract

An algorithm and data structure were written for the purpose of determining the individual structures of all the isomers of an alkane, a hydrocarbon characterized by a non-looping branch structure. This was accomplished by cycling through possible structures, one at a time, and comparing different sections of each structure against itself during the creation process. Each structure would either be deemed unique and recorded, or discarded, based on a set of priority rules. The resulting computations for the number of isomers for alkanes with carbon contents up to 18 match the literature exactly. However, after 18 carbons a small number of excess structures are returned. The algorithm's efficiency is its greatest accomplishment, far surpassing that of previous attempts. It uses only a very small amount of memory, and exhibits a linear relationship between the required computational time and the number of isomers calculated.

Specialized Terms

The first terms that need clarification are counting and enumeration. In reference to the calculation of isomers, the term ‘counting’ means finding the number of isomers of a given formula, without necessarily determining anything further about those isomers. The term ‘enumeration’ means finding the number of isomers, as well as all individual structures, for the given formula. Counting has historically been performed by mathematical equations, whereas enumeration has been a result of computer programs, which can store vast amounts of data.

Two terms that have specific meaning to isomer enumeration are ‘irredundant’ and ‘exhaustive.’ The term ‘irredundant’ means that a particular algorithm does not generate any extra or equal structures; that is, every structure it generates is unique. The term exhaustive simply means that an algorithm calculates all possible isomers without exception. A perfect algorithm needs to be both exhaustive and irredundant.

When referring to chemical structures, the terms cyclic and acyclic have important meaning. In a cyclic compound the bonds between atoms form at least one closed loop or ring. An acyclic compound is one that does not create a closed loop but is characterized by a branching structure. Finally the term degree, or valence, when used in reference to an atom means the number of bonds that atom will form.

Review of Literature

Introduction to Hydrocarbons: Types and properties

A hydrocarbon is any chemical compound containing only carbon and hydrogen. Hydrocarbons can be separated into acyclic and cyclic types. The acyclic hydrocarbons are characterized by a branched tree structure, and can be separated into three categories.

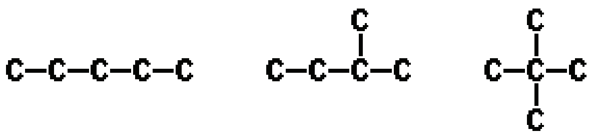
1. Alkanes contain only single (σ) bonds, and have the general formula C_nH_{2n+2}.
2. Alkenes contain a double (π) bond, and have the general formula for C_nH_{2n}.

3. Alkynes contain a triple (π) bond and have the general formula C_nH_{2n-2}.

The cyclic hydrocarbons are characterized by a closed structure, creating a ring. Cyclic hydrocarbons can be separated into two categories.

1. The cycloalkanes contain only single (σ) bonds, and have the general formula C_nH_{2n}.
2. The aromatic hydrocarbons are based on permutations of the benzene molecule, which has the formula C_6H_6 (Brescia 566-582).

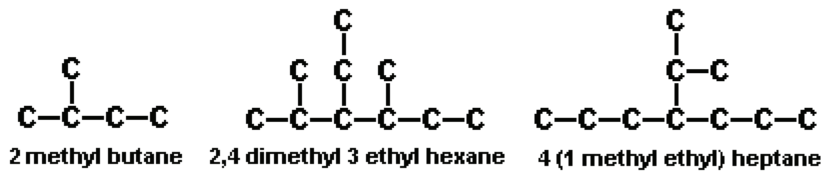
Isomers are any set of chemical compounds that have the same chemical formula, but a different arrangement of the atoms involved (555-556). For example, pentane (C_5H_{12}) has the following three isomers (the hydrogens are omitted for ease of reading):



(Brescia 569).

Nomenclature

In order to systematically name all possible hydrocarbon isomers, the International Union of Pure and Applied Chemistry (IUPAC) created a set of regulations. The first step is to find the longest continuous carbon chain, which then becomes the root of the name. In the next step, number the carbons on the longest chain in order, starting at the end that gives the carbons with branches the lowest numbers. Next, each branch is named as if it were a non-branching alkane, except the –ane ending is replaced with –yl. Then each branch has the number of its position on the main chain written in front of it. Any branches with the same number of carbons are combined, with both numbers in front of the name, and an appropriate prefix denoting the number of these branches. Any branches that have their own branches are named as if they were separate, and added in parentheses. The process for alkenes and alkynes is the same, except the double or triple bond has to be on the longest chain, and it is given numbering priority. A structure that has no branches is giver the n- prefix before its root name (Brescia 573-574). The following are examples of alkanes and their nomenclature:



History of Isomer Counting and Enumeration

Mathematical Counting

The counting and enumeration of chemical isomers has been of interest to chemists and mathematicians for more than 125 years, yet the need for an efficient algorithm still exists. Originally, the task was to mathematically calculate the number of isomers without necessarily generating any structures.

The first such attempt to calculate the number of isomers within the alkane series was carried out by Cayley in 1875. He realized that the number of isomers of an alkane with a given carbon is dependent on the number of isomers in the alkane with one fewer carbon atom. Using this fact, and mathematical graph theory, he predicted the number of isomers for alkanes with a carbon content through 13. However, his calculations for $C_{12}H_{26}$, and $C_{13}H_{28}$ were incorrect. Five years later, Herman corrected Cayley's calculation for $C_{12}H_{26}$, and $C_{13}H_{28}$, by classifying structures into groups based on branching from the main chain. His method, however was limited to $C_{13}H_{28}$, and could not be applied to alkanes with higher carbon contents.

In the mid-1890's Delannoy attempted to solve the problem by using an equation where a new term was added for every increase in the carbon content. However, this formula also was only accurate through $C_{13}H_{28}$. Another attempt at solving the problem was made by Losanitsch, by using combinations, permutations, and variations. He calculated the isomer count through $C_{14}H_{30}$, but was incorrect in his calculation for $C_{12}H_{26}$, and $C_{14}H_{30}$.

In 1898, observing that previous formulas were complicated and difficult to use, Goldberg attempted to create a more simplified formula. His work, however, only proved to be accurate through $C_{10}H_{22}$ (Henze 3077-3079).

In 1931, Henze and Blair developed a less complicated method for counting the number of isomers of the alkane series. Their method was based on the relationship between isomeric alkanes, and isomeric alcohols of a lesser carbon content (an alcohol is essentially an alkane with an OH^- group in the place of a carbon). The alkanes were separated into four groups depending on whether they had an even or odd number of carbon atoms, and whether their structure could be evenly divided into equal parts. Some groups were then divided into subgroups. Each group or subgroup was then assigned its own equation, which was based on the previously calculated number of isomeric alcohols of a given carbon content one less than the alkane in question. The results of all the equations would then be summed to obtain the final isomer count. Using this method, Henze and Blair were able to accurately calculate the number of isomers in the alkane series up to $C_{20}H_{42}$, but veered off for higher carbon contents (Henze 3079-3084).

Next, Polya developed his theorem to calculate isomeric tree structures. Using complex algebraic methods, Polya was able to calculate all possible combinations and permutations of a set of trees. His theorem proved to be useful for calculating the number of isomers of alkanes beyond $C_{20}H_{42}$ (Balaban 304-310). To present, his theorem has been applied to the counting of more complex and cyclic structures.

<u>Computerized Enumeration</u>

With the invention of the computer, the problem of counting isomers changed to enumerating them. Instead of calculating the number of isomers through mathematical processes, computers could now be used to generate, count, and store all possible chemical structures with a given chemical formula.

In 1968, Lederberg et al developed the first computer program for the purpose of enumerating acyclic chemical isomers, known as DENDRAL. It worked by converting chemical structures into a linear notation, and then generating all possible combinations and permutations thereof. The program also made available the option to include substituents that either must, or must not be present in all final structures. DENDRAL was also able to consider compounds containing elements other than just carbon and hydrogen. Finally, the program was theoretically able to calculate cyclic compounds, but only with changes to the code and a very low efficiency (Lederberg 2973-2976).

In 1974, Masinter et al created a new program to calculate cyclic and acyclic isomers. He did so by incorporating the acyclic generator from the DENDRAL program, and creating a cyclic generator. The new cyclic generator was based on graph theory, using nodes and edges to represent atoms and bonds. The generator started by creating a list of structures with degrees higher than three. It then diluted those structures with unlabeled atoms of degree two. Next it gave valences to the unlabeled nodes, and then labeled the nodes as appropriate atoms. Finally, the acyclic generator was used to attach branched structures to the produced cyclic parts. At each step in this process, all possible combinations and permutations were applied to the newly generated structures. The program was successful in irredundantly and exhaustively generating all possible structural isomers, cyclic and acyclic, with few exceptions. The major drawback to this algorithm is that it was very time consuming to execute (Masinter 7702-7714).

In 1990, Henderickson and Parks developed a new program for calculating carbon skeletons. Their method was based on calculating adjacency matrices to represent molecular structures. Each matrix was filled with binary values, one row at a time. After each row the program would check to see if the matrix was still valid. All fully filled valid matrices were then added to the final list. Once the final list was completely generated, it was sorted through to check for redundancies. This method was able to exhaustively calculate cyclic and acyclic structures for any chemical formula, but the program generated large numbers of redundant isomers, causing the sorting process to be extremely time consuming. This drawback made the algorithm impractical for use on structures with more than ten atoms (Hendrickson 101-105).

In 1993, Jackson and Bieber developed a new method for enumerating isomers in the alkane series. Their method was based on the number of bonds each carbon atom had to other carbon atoms, also called degree distribution. A carbon atom bonded to two other carbon atoms has a degree distribution of two. They start by manually generating substructures for the alkane that they want to enumerate, containing only carbons with a degree distribution of three or four. Then, those substructures were extended with carbons of a degree distribution of one. Finally a computer program is written to dilute the extended substructures, one at a time, with carbons of a degree distribution of two, in all possible arrangements. Using this method, Jackson and Bieber were able to enumerate alkane isomers up to a carbon content of 10. However, their method requires a considerable amount of manual work, and the writing of a new computer algorithm for each substructure, thereby making their method impractical for use on alkanes with large carbon contents (Jackson 701-707).

In 2000, van Almsick, Dolhaine and Honig presented a new algorithm to count isomers using Polya's theorem. The program, written in the algebraic language MATHEMATICA, used complex mathematical formulas to count isomers. The program was designed to calculate the number of isomers without generating any actual structures (van Almsick 956-966).

The Algorithm

Processes and Methods

The goal of this algorithm is to exhaustively and irredundantly enumerate the isomers of any member of the alkane series. In order to accomplish this, an original algorithm was developed in the C++ computer language, which was not based on any previously attempted methods. The algorithm makes extensive use of recursion to accomplish its goal. A custom data structure for use in this program was also developed.

Data Structure

The data structure was designed to simulate a tree structure, consisting of a main tree, or long chain, with branches splitting off of it. The structure named a 'Tree' consists mainly of a vector of vectors containing pointers to trees. The first vector is the 'length' of the tree, with each element representing one carbon atom. The second vector represents the valence of the atom in that position, minus its bonds within the first vector. In the case of alkanes, this vector will always be of length two. Each element in the secondary vectors contains either a blank pointer, or a pointer to another tree, which makes this a recursive data structure.

The data structure also incorporates a number of helper functions. There is one function used to resize the tree, three integer and two boolean functions used to access the Tree's properties, and three integer and three boolean functions used for left and right Tree End comparisons (Tree Ends are explained later).

Main Program

The main algorithm starts with the carbon content of an alkane, and runs through with long chains of a diameter from the carbon content to a diameter of 2. On each long chain, the algorithm places available carbons one at a time until all available carbons are used, at which point the structure is saved. The algorithm then goes back one placement, resizes that branch to be one carbon longer, and branches that branch, and so on. It then moves each of these branches across the tree to all legal places. At every step, either moving or resizing, the structure is checked for uniqueness.

This algorithm is accomplished through extensive use of recursive functions; there are four main functions as well as two legality and uniqueness checking functions. A structure's uniqueness is determined by assigning sets of priorities to branches, and in some cases, sections of the long chain. These priorities and sets are used to ensure that all structures saved are unique, and not redundant. There are three types of sets. The first is the 'top' and 'bottom' positions (top and bottom are arbitrary labels for branches originating from the same carbon; 'top' always exists first, and 'bottom' only exists where there is a top). The second type of set is corresponding positions on the long chain that are equal distances from the center of the structure. Finally, the third type of set is left and right Tree Ends, and Branch Ends, which will be explained later. Priorities are then assigned within the sets, which have to be based on the order in which the branches are created. This order is left to right, and 'top' to 'bottom' (See Figure 1).

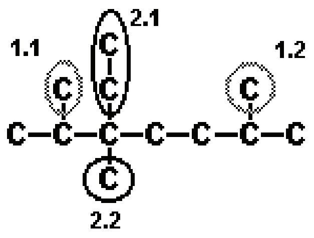


Figure 1.
Priorities for opposite (1.1-1.2), and 'top' & 'bottom' (2.1-2.2) branches.

The priorities are used to compare branches. Branches of lesser priority have to be 'less than or equal to' branches of higher priority in order to ensure uniqueness. To determine if a branch is less than or equal to another, first their respective diameters are compared. If those are equal, then the sub-branches at the far ends of each branch, in the 'top' position, are compared. If those are equal, the sub-branches in the 'bottom' position at the same location are compared, and so forth down the branches. The lesser branch is the one that contains the first lesser sub-branch. If all sub-branches are equal, the two branches are equal. Note that a branch of a lower priority can have a greater sub-branch than the respective sub-branch on a branch of higher priority, within the same set, given that the sub-branch occurs after a point where the lesser is less than the greater branch (See Figure 2).

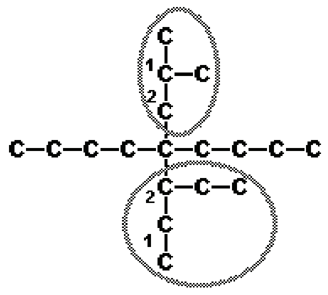


Figure 2.
This is a unique structure. The branch at position 1 on the 'top' allows the branch at position 2 on the 'bottom' to exist.

A left or right Tree End is a branch on the long chain whose length makes the distance from its end to its branching point the same as the distance from the branching point to the end of the long chain. When this occurs, the enclosed section of the long chain and the branch become interchangeable, and priorities have to be set between them (See Figure 3). However, this priority complicates the comparison of right Tree Ends. When Tree Ends are created past the initial right Tree End, they continually take priority from the long chain (left to right priorities), requiring a specialized comparison algorithm (See Figure 4).

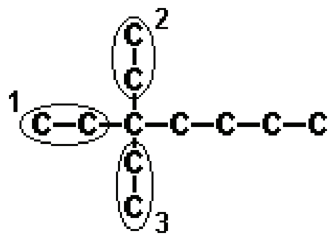


Figure 3. Left Tree End Priorities

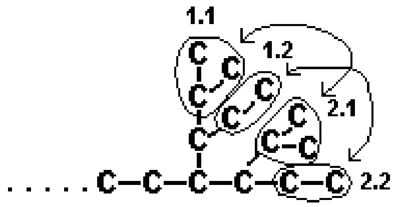


Figure 4. Complex right Tree End Priorities

Another layer to prioritizing occurs when opposite locations on the long chain have the same number of Tree Ends. This makes the two sides of the long chain interchangeable, and priorities have to be set between them (left to right). The first priority branch on the left is compared to the first priority on the right, the second to the second, and so on (see Figure 5).

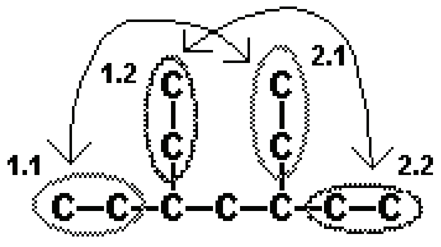


Figure 5.
Priorities and comparison of left and right Tree End set

A Branch End is a sub-branch on a branch such that its length is equal to the distance from its branching point to the far end of the branch. When this occurs, priorities are assigned according to the order in which the sections where created (See Figure 6).

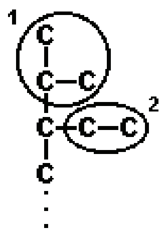


Figure 6. Priority assignments on branch ends.

It should be noted that priority sets of left Tree Ends versus right Tree Ends take priority over sets of opposite positions on the long chain. That is, if the right side of a left vs. right Tree End set is less then the left, what happens between them is unimportant (See Figure 7). Also, if the first priority on the right of a right vs. left Tree End set is less than the first priority of the left, then the second priorities and down are irrelevant. This however does not extent to 'top' vs. 'bottom' sets, left or right Tree End sets, or Branch End sets (See Figure 7).

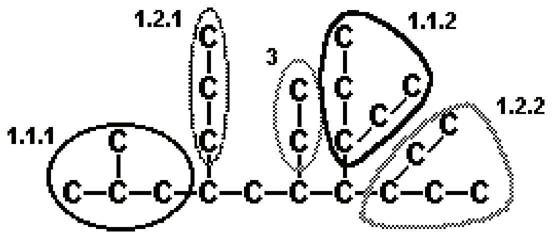


Figure 7.
This structure is valid. 1.2.2 can be greater than 1.2.1 because 1.1.1 is greater than 1.1.2. Also, 3 is allowed because the left Tree End is greater than the right Tree End

If a structure passes all sets as legal and unique, and has no available carbons left to place, it is recorded, and the number of isomers is incremented.

The purpose of these sets of priorities is to allow the algorithm to generate isomers exhaustively, while at the same time never generating redundant structures. In this way, the final list of isomers never has to be rechecked, or referenced within the algorithm, making it considerably more efficient.

Development Process

The algorithm was developed by first creating a data structure to represent an alkane. Next, the general frameworks of the functions in the main program were written, and the recursive structure was implemented. After that, rules were established governing the creation of all physically possible branches, and processes for output were added. At this point, the program could be run and the output analyzed. A preliminary set of priority rules were written and tested. The output was analyzed, faults were determined, and modifications made. Through the process of modification and analysis, the algorithm was made increasingly accurate. However, this process of analysis does becomes tedious when it involves comparing hundreds, or thousands of isomers to determine which ones are in excess, or missing. Unfortunately, there is no alternative method that is more effective, and so a large amount of time was spent developing the algorithm in this way.

Results

Output

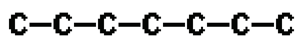
Specific Output

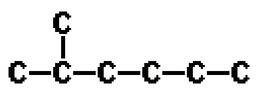
The algorithm produces a file consisting of a coded form of each isomer, and the total number of isomers. In this code, the integer –1, followed by another integer, represents a new branch with a length corresponding to the second integer. An integer not preceded by –1 represents the position of a new branch, and will be followed by a –1. Finally, –2 represents the end of a branch started by a –1. Each isomer occupies one line within the file.

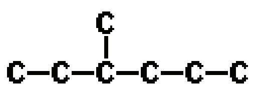
The following is the exact output for heptane (C_7H_{16}), which has 9 isomers:

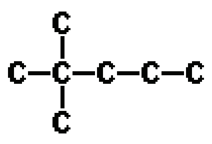
```
-1 7 -2
-1 6 1 -1 1 -2 -2
-1 6 2 -1 1 -2 -2
-1 5 1 -1 1 -2 1 -1 1 -2 -2
-1 5 1 -1 1 -2 2 -1 1 -2 -2
-1 5 1 -1 1 -2 3 -1 1 -2 -2
-1 5 2 -1 1 -2 2 -1 1 -2 -2
-1 5 2 -1 2 -2 -2
-1 4 1 -1 1 -2 1 -1 1 -2 2 -1 1 -2 -2

9
```

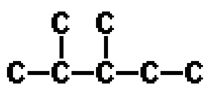
This translates into the following structures:

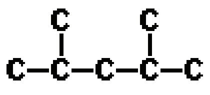

n-heptane

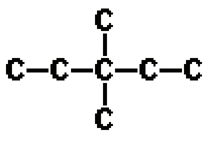

2 methyl hexane

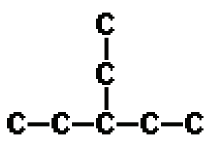

3 methyl hexane

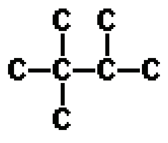

2,2 dimethyl pentane


2,3 dimethyl pentane


2,4 dimethyl pentane


3,3 dimethyl pentane

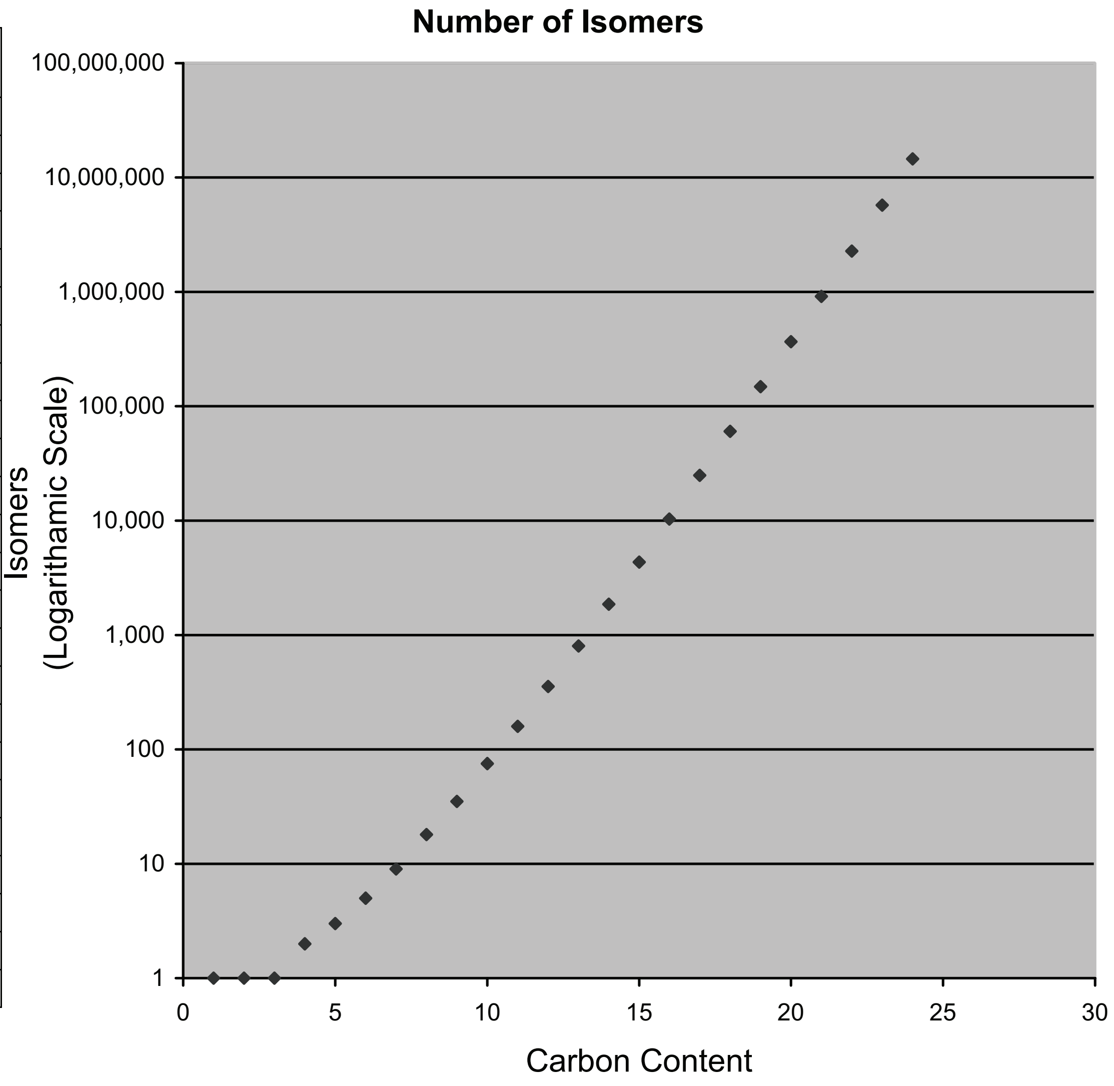

3 ethyl pentane


2,2,3 trimethyl pentane

General Output

The following is a summary of the number of isomers the algorithm found for alkanes with a carbon content of 1 through 24:

Carbon Content	# of Isomers
1	1
2	1
3	1
4	2
5	3
6	5
7	9
8	18
9	35
10	75
11	159
12	355
13	802
14	1858
15	4347
16	10359
17	24894
18	60524
19	148294
20	366376
21	910994
22	2279778
23	5737905
24	14506015



After a carbon content of seven, the graph takes on a linear appearance, which indicated that the number of isomers increases logarithmically with the carbon content. Before seven however, there does not appear to be an easily discernable pattern. Even the linear relation after seven could only be used to estimate the number of isomers at greater carbon contents.

Analysis

Accuracy

The following is a comparison between the number of isomers and published results:

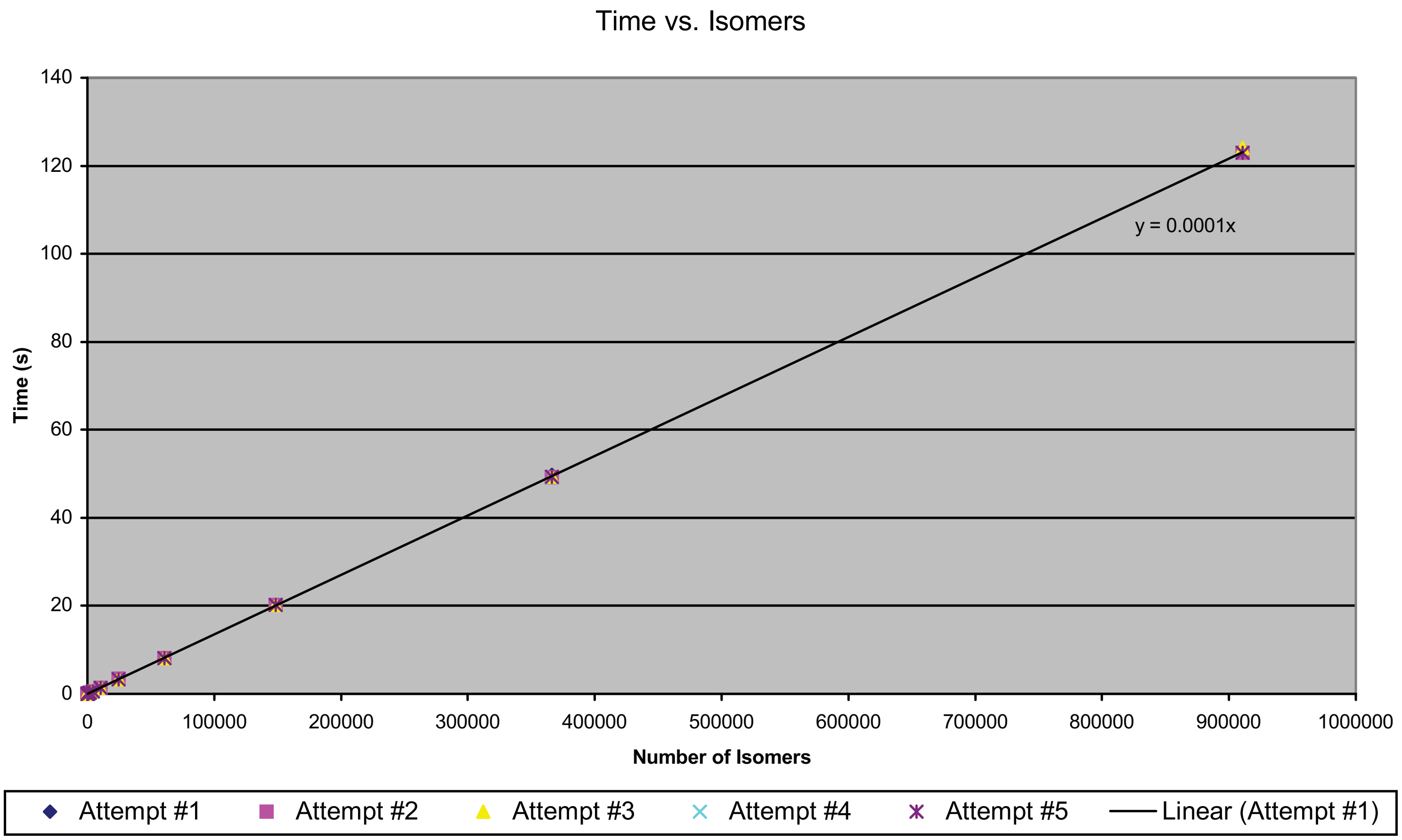
Carbon Content	# calculated	Literature	Difference	Percent Extra
1	1	1	0	0%
2	1	1	0	0%
3	1	1	0	0%
4	2	2	0	0%
5	3	3	0	0%
6	5	5	0	0%
7	9	9	0	0%
8	18	18	0	0%
9	35	35	0	0%
10	75	75	0	0%
11	159	159	0	0%
12	355	355	0	0%
13	802	802	0	0%
14	1858	1858	0	0%
15	4347	4347	0	0%
16	10359	10359	0	0%
17	24894	24894	0	0%
18	60523	60523	0	0%
19	148286	148284	2	0.00134876%
20	366335	366319	16	0.00436778%
21	910815	910726	89	0.00977242%

In all alkanes with a carbon content of 1 to 18, the algorithm is accurate. After this point, the algorithm does generate excess structures. This error, however, represents less than 0.01% of all possible isomers. Also, this type of error is acceptable because it is generating structures more than once, which is favorable to having structures missing. Because the error is so small, it is reasonable to believe that it could be fixed through very minor changes to the code. This has not been done here because the time that would be required to determine the flaw would be impractical, given the scope of this thesis. Also, doing so would require more resources than benefits it would provide.

Efficiency

This algorithm represents a considerable improvement over previous algorithms in both calculation time, and memory usage. The following table is a list of the computation times required for carbon contents of 1 through 21[1]:

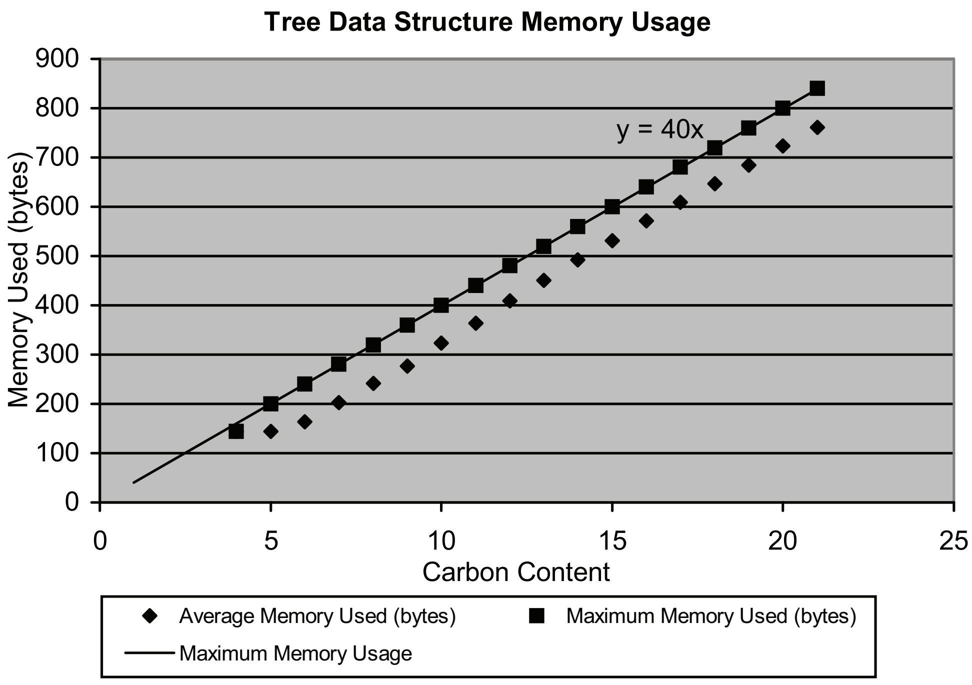
Carbon Content	Time (s)					
	Attempt #1	Attempt #2	Attempt #3	Attempt #4	Attempt #5	Average Time
1	0.000	0.000	0.000	0.000	0.000	0.000
2	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	0.000	0.000	0.000	0.000
4	0.000	0.000	0.000	0.000	0.000	0.000
5	0.000	0.000	0.000	0.000	0.000	0.000
6	0.000	0.000	0.000	0.000	0.000	0.000
7	0.000	0.000	0.000	0.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000
9	0.032	0.000	0.016	0.015	0.000	0.013
10	0.000	0.016	0.015	0.016	0.016	0.013
11	0.015	0.031	0.015	0.016	0.031	0.022
12	0.047	0.046	0.047	0.047	0.047	0.047
13	0.125	0.110	0.109	0.109	0.110	0.113
14	0.250	0.235	0.250	0.250	0.250	0.247
15	0.594	0.593	0.579	0.578	0.578	0.584
16	1.391	1.375	1.406	1.407	1.406	1.397
17	3.359	3.395	3.344	3.344	3.328	3.354
18	8.078	8.049	8.125	8.109	8.049	8.082
19	20.156	20.172	20.172	20.156	20.175	20.166
20	49.719	49.250	49.260	49.250	49.281	49.352
21	123.031	123.015	124.125	122.906	123.016	123.219



The efficiency of this algorithm is by far its strongest point. As demonstrated in the preceding graph, the amount of computational time required varies linearly with the number of isomers. This is notable because in past algorithms, computational times have increased logarithmically or greater, with the increasing number of isomers. One of the most evident examples of this is Hendrickson and Parks SYNGEN program, which required 51min 24sec to calculate the isomers of nonane, 12h 7min 11sec for decane, and 192h 44min 50sec for undecane. These times translate into output rates of 0.0113 isomers per second, 0.0017 isomers per second and 0.0002 isomers per second respectively; a logarithmically decreasing efficiency. Even after taking into consideration the improvements in computer technology since Hendrickson and Parks work was published, the algorithm presented here represents a radical improvement in efficiency. Based on the efficiency graph's trendline, this algorithm achieves a constant efficiency of 7200 isomers per second.

The algorithm is also very efficient in memory usage. The following represent the amount of memory used by the Tree data structure throughout the execution of the algorithm:

Carbon Content	Average Memory Used (bytes)	Maximum Memory Used (bytes)
4	144	144
5	144	200
6	164	240
7	202	280
8	242	320
9	277	360
10	323	400
11	364	440
12	409	480
13	451	520
14	492	560
15	531	600
16	571	640
17	609	680
18	647	720
19	685	760
20	723	800
21	761	840



This demonstrates a direct linear relationship between the carbon content and the maximum amount of memory required by the Tree structure. The trendline on the graph indicates that the addition of one carbon increases this maximum by 40 bytes. At this rate, it would require a carbon content of over 26 000 for a Tree structure to occupy even 1Mb of memory. Also, the fact that there is only one Tree created throughout the entire algorithm keeps its memory requirements very low.

Local variables within individual instances of the recursive functions, which include indexing vectors, checking vectors, and an assortment of integers, require minimal amounts of memory. In practice, with all required system resources, and running normally, the algorithm's memory usage almost never exceeds a net 800kb. This is an important feature because the algorithm will not rapidly expand in memory, avoiding allocation conflicts that can occur when large amounts of memory are requested.

Because of the computationally intensive nature of the algorithm, the one resource that it requires in large amounts is CPU time. When running, the algorithm will occupy all available CPU time. However, this is to be expected, and is in no way a drawback to the usefulness of the algorithm.

Applications

This algorithm would primarily be used by biochemists, in order to identify possible compounds. If the mass of a hydrocarbon were known, this algorithm could be used to determine all possible structures in which that hydrocarbon is arranged. This is of particular interest to biochemists researching cellular function, where structure greatly affects how a molecule interacts with its surroundings. The output of the algorithm could also be used to perform statistical analyses on the possible structures of alkanes. This is of interest to both biochemists and industrial chemists in the hydrocarbon production process.

An important aspect of the algorithm is that it has the ability to be expanded and adapted. In the future, it could be modified to calculate isomers of alkenes or alkynes. The algorithm could also be expanded to incorporate elements other than just carbon, extending the its usefulness beyond only hydrocarbons.

Conclusions

The algorithm presented here has been successful in accomplishing the majority of the goals it was created to accomplish. Although the original intent was to have the final algorithm 100% accurate for all carbon contents, it was later determined that doing so would require far more resources than were available. Also, the type of inaccuracy present is an acceptable one. The efficiency of the algorithm far surpasses its original estimates. Especially successful were the memory usage, which stays within a small bound, regardless of carbon content, and the linear relationship between the computation time and the number of isomers. In comparison to other computerized algorithms mentioned in this document, the one presented here represents a significant improvement in efficiency, which has been a major drawback in the past. Despite the imperfection in this algorithm, it proves that its concept is feasible, executable, and an improvement over past efforts.