MATCH
Communications in Mathematical and in Computer Chemistry

MATCH Commun. Math. Comput. Chem. **58** (2007) 385-401

ISSN 0340 - 6253

Backtracking to Compute the Closed-Shell Independence Number of a Fullerene

Sean Daugherty[1] Wendy Myrvold[*1] Patrick W. Fowler[†2]

sdaughe@cs.uvic.ca wendym@cs.uvic.ca P.W.Fowler@sheffield.ac.uk

(Received July 1, 2006)

Abstract

A *fullerene* is a molecule consisting of carbon atoms such that each atom is adjacent to three others, with the bonds forming the edges of pentagonal and hexagonal faces. The study of fullerene reactivity has motivated the concept of *closed-shell independent sets*. Such sets specify the location of bulky addends on a fullerene such that the residual π system is a closed shell. The cardinality of the maximum-sized set is the closed-shell independence number. This paper presents a backtracking algorithm to calculate this number for fullerenes. The algorithm was run on all fullerene isomers with 100 or fewer carbons and from the results, some observations are made related to the properties of the fullerene isomers that indicate a minimum or maximum closed-shell independence number. It is conjectured that exactly three isomers have identical closed-shell independence and independence numbers.

1 Introduction

Fullerenes are all-carbon molecules that have polyhedral structures in which each of the atoms is directly bonded to its three neighbors and in which all faces of the polyhedron are either pentagonal or hexagonal. Thus the molecular graph is cubic and polyhedral, with face sizes limited to five and six. A fullerene with n carbon atoms is denoted by C_n. In chemical terms, these molecules are *unsaturated*: the fourth valence of each carbon atom is used to form a more or less delocalized π system on the surface of the

[*]Supported by NSERC.
[†]Supported by the Royal Society/Wolfson Research Merit Award Scheme.
[1]Dept. of Computer Science, University of Victoria, Victoria, BC, Canada V8W 3P6
[2]Dept. of Chemistry, University of Sheffield, Sheffield S3 7HF, UK

polyhedron, augmenting the localized σ bonds along the edges. A general discussion of theoretical chemistry aspects of fullerene structure, isomerism and properties is given in [11]. At accessible values of n, there are typically many combinatorially distinct polyhedra (isomers) satisfying the fullerene definition, of which only a few (if any at all) have been isolated in experiment. The archetypical icosahedral [60]-fullerene, C_{60}, for example, is one of 1812 isomeric possibilities, and is apparently the only one of this large set that can be isolated in quantity from the products of the arc synthesis. Evaluation of graph invariants is one strategy for the identification of the experimentally most relevant isomers from the mass of mathematical possibilities. Hence, the development of fast algorithms for construction of fullerenes [2, 3] and computation of invariants is of practical chemical as well as theoretical computer-science interest.

Applicability of fullerenes in functional compounds, materials and devices will require the ability to tune stability and properties. Here too, graph invariants have a part to play in rationalization of the chemistry of fullerene derivatives. Even the simplest addition reactions—of atoms of a single type X to a single fullerene C_n—generate a combinatorial explosion of possible isomers C_nX_q differing in the number and placing of addends, and may lead to difficult experimental problems of separation and characterization. One way to avoid such problems may be to drive the reaction to completion, when the product distribution may become simpler, and the question then arises: what is the maximum number of addends of a given type that will ultimately attach to a given fullerene, if the addends are too bulky to occupy neighboring carbon sites? In the absence of other restrictions on addition, the answer will be represented by taking the X positions to be a maximum independent set of the vertices of the fullerene graph, and the stoichiometry C_nX_q will reflect the independence number q of the graph.

However, chemical theory does impose a further restriction in that the set of carbon atoms that do not bear an addend constitute a (possibly disconnected) unsaturated system and should have a stable distribution of π electrons. This amounts to the requirement that every component of the graph induced by the complement of the independent set must have a closed-shell, i.e., an adjacency matrix in which exactly half of the eigenvalues are positive. An independent set with this additional property in the complement is called a *closed-shell independent set* and has been studied in the context of fullerenes [6, 10, 12, 13, 16]. The cardinality of such a set is the closed-shell independence number of the fullerene. In the case of icosahedral C_{60}, the final product of bromination, $C_{60}Br_{24}$, is explained in stoichiometry, symmetry and structure by the unique maximum closed-shell independent set [10].

The problem of finding a maximum independent set has been studied on general graphs [1, 4, 15, 17] and fullerenes both theoretically [9, 14] and algorithmically [12].

Backtracking methods have been used to search for a maximum independent set of all fullerenes with 120 or fewer vertices [12]. It is clear that the closed-shell independence number is at most the independence number, since the former is a restricted version of the latter. For icosahedral fullerenes, it has been shown that these values are equal only on the two fullerene isomers with 20 and 60 carbon atoms [12]. There has been a proposal to predict the relative stabilities of fullerene isomers themselves by minimization [8], but the performance of the independence number as an index of stability is not good [9, 12].

Before this work, the closed-shell independence number was known for only six fullerenes [13, 12]. This paper describes a backtracking algorithm that was used to calculate the closed-shell independence number of all fullerenes with 100 or fewer atoms. The algorithm seeks early detection of the nonexistence of a closed-shell independent set of a given order or larger. This is done using a combination of dynamic upper bounds, prioritized vertex selection, and the identification of special cases to avoid lengthy eigenvalue calculations when possible.

This paper is organized as follows. First, the problem is formalized (§2). Then the high-level pseudo-code for the algorithm is discussed (§3). Next, specially designed data structures used to implement the algorithm are described (§4) as well as some additional approaches to reduce the run time (§5). Some observations are made on the computational results (§6) and the results are compared with previous calculations of the independence number (§7). Finally, ideas for further improvement and research are discussed (§8).

2 Definitions and Notation

A fullerene can be modeled as a graph by using vertices for atoms and edges for bonds. The next definition describes the structure of the graphs representing fullerene skeletons.

Definition 2.1 (Fullerene Graph). *A graph for a fullerene molecule with n carbon atoms is a 3-regular planar graph*[3] *on n vertices with face sizes equal to five or six. Such a graph is denoted by* C_n.

The notation C_n is ambiguous because n does not uniquely characterize the structure of the fullerene. For $n > 26$, multiple isomers exist, resulting in non-isomorphic representative graphs. To specify the structure of the fullerene, one provides the isomer number i with the notation $\mathrm{C}_n{:}i$. Isomer numbers are determined by sorting the fullerenes based on their lexicographically smallest face spiral sequence [11, §2.6].

To model a closed-shell independent set, a characterization of a closed-shell graph is needed. A mathematical interpretation is given by the eigenvalues of a graph because

[3]All graphs are assumed to be simple, connected, and undirected.

they correspond directly to the molecular-orbital energy levels in the simplified model of electronic structure given by Hückel theory [19].

Definition 2.2 (Closed Shell)**.** *A graph $G = (V, E)$ is said to be closed shell if*

(i) n is even and

(ii) exactly half of the eigenvalues of the adjacency matrix of G are positive.

This definition refers to a *properly closed shell* [11, p. 47], which is shortened to *closed shell* throughout this paper. A closed shell restriction is added to the definition of an independent set to yield the definition of a closed-shell independent set below.

Definition 2.3 (Closed-Shell Independent Set)**.** *On a graph $G = (V, E)$, a vertex subset $S \subseteq V$ is called a closed-shell independent set if*

(i) $\forall u, v \in S$, $(u, v) \notin E$ (independence) and

(ii) each component of $G - S$ is closed shell.

The largest such set is useful for studying fullerene reactivity. The following definition gives a corresponding invariant for a graph.

Definition 2.4 (Closed-Shell Independence Number)**.** *The closed-shell independence number of a graph is:*

$$\alpha^{-}(G) = \max_{S \subseteq V} |S|$$

where S is a closed-shell independent set.

The algorithm to be presented relies on two important properties of closed-shell independent sets.

Property 2.5. *Every closed-shell independent set of a fullerene has an even number of vertices.*

This property follows from the facts that every fullerene has an even number of vertices and that for a closed-shell independent set S, each component of $G - S$ has an even number of vertices. The next property follows from this one and the theorem in [13, eqn. (7)] that states $\alpha^{-}(G) \leq 2n/5$.

Property 2.6. *For a fullerene, $\alpha^{-}(G) \leq 2 \lfloor n/5 \rfloor$.*

Note that $2 \lfloor n/5 \rfloor$ is the largest even integer less than or equal to $2n/5$.

3 The Backtracking Algorithm

This section discusses the general structure of the algorithm to find the closed-shell independence number of a fullerene. The heart of the backtracking algorithm is recursive, as shown in the pseudo-code of Algorithm 1.

The algorithm maintains the status of each vertex as red, blue, or uncolored. *Red* denotes vertices in the closed-shell independent set and *blue* denotes those not in the set. Uncolored vertices are the ones for which set membership has not yet been decided. A *blue component* is a maximal connected subgraph in which all vertices are blue. A closed-shell independent set has no adjacent reds and any blue component surrounded by reds must be closed shell.

Initially, all of the vertices are uncolored. At each step of the algorithm, an uncolored vertex is selected. It is first colored red and then blue. For each case, extensions to the partial coloring are explored by applying the algorithm recursively. When a coloring is complete, it corresponds to a closed-shell independent set. In order to achieve faster performance, the algorithm aborts exploration of partial colorings for which it can be concluded that either there are no feasible completions or that any feasible completions result in a closed-shell independent set size that is less than or equal to the best found so far.

Three parts of Algorithm 1 will now be described in more detail: how to select an uncolored vertex (line 6), how to determine if a partial coloring is feasible (lines 8 and 12), and how to detect when it is no longer possible to complete a partial coloring to obtain a new larger-sized set (lines 9 and 13).

3.1 Selecting an Uncolored Vertex

Uncolored vertices are stored in a priority queue as they await coloring. The priority is assigned such that vertices with the least color flexibility are chosen first. Thus, infeasible colorings are detected faster. A vertex is assigned the highest possible priority according to the following descriptions:

Highest	vertices with at least 1 red neighbor
↑	vertices with 3 blue neighbors
	vertices with 2 blue neighbors
	vertices with 1 blue neighbor
Lowest	vertices with 0 blue neighbors

A vertex with a red neighbor has the highest priority, since those must be colored blue to avoid adjacent reds. The priority of a vertex is increased as appropriate until it is

```
ClosedShell ($F$)
Input    : A fullerene $F$
Returns: Closed-shell independence number of $F$
begin
    mark all vertices of $F$ as uncolored
    return ClosedShellLevel($F$, $0$, $0$)
end

ClosedShellLevel ($F$, $NumColored$, $LargestSeen$)
Input    : A fullerene $F$ with a feasible partial coloring having $NumColored$
           colored vertices and $LargestSeen$: the size of the largest c.s.i. set found
           previously
Returns: The larger of (i) the input value of $LargestSeen$ and (ii) the size of the
           largest c.s.i. set possible by completing the given partial coloring
begin
    if $NumColored = n$ then no uncolored vertices remain
        $LargestSeen \leftarrow$ max($LargestSeen$, $red vertex count$)
        return $LargestSeen$
    end
    Select an uncolored vertex $v$ from $F$
    Color $v$ red
    if current coloring is feasible then
        if larger set size may be possible then
            $LargestSeen \leftarrow$ ClosedShellLevel($F$, $NumColored + 1$, $LargestSeen$)
        end
    end
    Color $v$ blue
    if current coloring is feasible then
        if larger set size may be possible then
            $LargestSeen \leftarrow$ ClosedShellLevel($F$, $NumColored + 1$, $LargestSeen$)
        end
    end
    Uncolor $v$
    return $LargestSeen$
end
```

Algorithm 1: Essentials of the backtracking algorithm

eventually selected and removed from the queue. Vertices are selected from the highest occupied priority in a first-in first-out manner.

3.2 Feasibility

After an uncolored vertex is selected, it is colored first red then blue. Then the algorithm proceeds with this coloring only if it is feasible (lines 8 and 12 of Algorithm 1). This section gives the conditions under which a coloring is feasible and how feasibility is determined.

At all times, the fullerene has a partial coloring with each vertex being either red, blue, or uncolored. Such a coloring is said to be *feasible* if it does not violate any requirements of a closed-shell independent set. Thus, a feasible partial coloring obeys the following rules:

1. No two red vertices are adjacent.
2. All blue components not adjacent to uncolored vertices (surrounded by red vertices) are closed shell.

To maintain feasibility, these rules are checked after a selected uncolored vertex is colored. Rule 1 is only checked after a vertex is colored red.

Rule 2 is checked after a vertex is colored either red or blue. If the selected vertex was colored blue, then the algorithm examines the blue component containing the selected vertex. A blue component is first checked to see if it is surrounded by red vertices. In such a case, the blue component is tested to see if it is closed shell. If a blue component is not surrounded by red vertices, then one of the adjacent uncolored vertices may be colored blue in the future, so no decision is made at this time as to whether or not it is closed shell. The application of rule 2 after coloring a selected vertex red is similar to the blue case. Here, the algorithm considers the blue components that contain the blue neighbors of the selected vertex.

3.3 Improvement After Coloring

Once a coloring is determined to be feasible, two dynamic upper bounds are calculated based on the current partial coloring (lines 9 and 13 of Algorithm 1). These values bound the total number of red vertices that can be present if the remaining uncolored vertices are colored according to the feasibility rules. The bounds are compared to $LargestSeen$ and if they indicate that $LargestSeen$ cannot be improved by coloring the rest of the vertices, then the algorithm does not continue with the current partial coloring.

The first upper bound is calculated after a vertex is colored either red or blue. Define an *uncolored edge* to be an edge incident to at least one uncolored vertex. Each additional red vertex will require 3 uncolored edges and because red vertices are non-adjacent, these edges are disjoint. Hence, the number of additional red vertices is at most 1/3 the number of uncolored edges. Furthermore, by Property 2.5, every closed-shell independent set must have an even number of vertices. Thus, the algorithm proceeds with the current coloring only when the current red vertex count plus 1/3 the number of uncolored edges is at least $LargestSeen + 2$.

The second upper bound is calculated only after a vertex is colored blue (line 13 only). This bound relies on the following property [13].

Property 3.1. *Every closed-shell independent set of a fullerene satisfies*

$$r = (2n - b_2 - 2b_3)/5$$

where r is the number of red vertices and b_i is the number of blue vertices with i blue neighbors.

A new maximum-sized set would have cardinality r such that $r \geq LargestSeen + 2$. Hence, the algorithm proceeds with the current coloring only when

$$5(LargestSeen + 2) \leq 2n - b_2 - 2b_3.$$

Blue vertices with uncolored neighbors are included in this calculation because the future addition of blue neighbors cannot increase the right-hand side. Implementation of this second upper bound resulted in the algorithm running approximately 20 times faster than when only the first upper bound is used.

4 Data Structures

As mentioned earlier, the algorithm considers components induced by the blue vertices in order to detect violations to feasibility. As well, there is a priority queue used to order the selection of the uncolored vertices. This section describes the specialized data structures used to maintain and access this information.

4.1 Priority Queue

As described earlier, the uncolored vertex selection order is maintained in a priority queue. The algorithm requires the ability to remove the highest-priority vertex, change (increase) the priority of a vertex, and undo either of these changes when the algorithm backs up. Each of these operations has been implemented to run in constant time. This is crucial because approximately one quarter of the execution time of the algorithm is spent on these operations.

Because there are only five priority levels, the priority queue is implemented as five queues. The vertex at the head of each queue, if any, is stored. Predecessor and successor values for each vertex are used to create a circular doubly-linked list for each queue. Arrays of size n are used to store the priority, predecessor, and successor of each vertex.

Each operation is performed in constant time. When a vertex is removed, it is taken from the head of the highest-priority nonempty queue. When the priority of a vertex changes, due to the coloring of a neighbor, the vertex is removed from its current linked list and added to the tail of the list for the new priority. When the algorithm backs up, either of these changes is undone by remembering the old priority, predecessor, and successor of the vertex and using these values to re-insert the vertex into its earlier position. By returning vertices to their original position in the queue instead of simply adding them to the tail of the appropriate queue, the algorithm runs in less than a third of the time.

4.2 Blue Components

The algorithm maintains information about the components induced by the blue vertices. This information includes the vertices in the component as well as descriptive values such as whether or not a closed-shell calculation should be run on the component, the cached result of such a calculation, and other values that identify components with faster methods of calculation as described in Section 5.

Each blue vertex is in exactly one blue component, allowing a disjoint-set data structure using rooted trees to store these components. For an overview of disjoint sets and their use in connected components, see [5, Ch. 21]. An implementation without path compression was chosen because the data structure must be able to quickly undo previous union operations when the algorithm uncolors a vertex. For the same reason, each parent-child relationship is doubly-linked instead of only the child-to-parent direction.

Neighboring blue vertices are always in the same component. In the union operation, a newly colored blue vertex v is merged with the blue components that contain v's blue neighbors. Vertex v is assigned to be the root of the merged component and its children are the unique roots of its blue neighbors' components.

When the algorithm backs up by uncoloring a blue vertex, the previous union operation must be undone. The algorithm guarantees that vertices are uncolored in the opposite order in which they were colored. Hence, a newly uncolored vertex is always the root of its component and its removal splits the component into the original components.

Values that describe the component and aid future closed-shell calculations are associated with the root. When performing a union, these values are calculated from the corresponding values of the children. When a union is undone, the previous values are recovered because they are still associated with the roots of the resulting components. For example, each root stores the number of vertices of each degree (within the component), 0 to 3. This compact storage of the degree sequence can be used to identify paths, cycles, and trees.

A blue component with no adjacent uncolored vertices cannot be extended and therefore may be tested to see if it is closed shell. To identify this situation, a count of the number of edges from a vertex in the component to an uncolored vertex is stored at the root. When this count is zero, the closed-shell calculation is performed.

The last value stored at a root is a cache of the result of the relatively slow closed-shell calculation. This avoids re-calculations that would otherwise occur when multiple neighbors of a newly colored red vertex are in the same blue component.

5 Decreasing the Run Time

A large run time bottleneck is the closed-shell calculation that is performed each time a blue component becomes surrounded by red vertices. From the definition of a closed-shell independent set, the distribution of the eigenvalues must be known to determine if exactly half of them are positive. It is desirable to avoid lengthy explicit eigenvalue calculations, which is possible for a large percentage of the cases that are encountered.

For paths and cycles, the closed-shell calculation is very easy. It has been shown [18] that the eigenvalues of a path satisfy the equation

$$\lambda_k = 2\cos\left[\frac{\pi k}{n+1}\right]$$

and the eigenvalues of a cycle satisfy the equation

$$\lambda_k = 2\cos\left[\frac{2\pi k}{n}\right]$$

for $1 \leq k \leq n$. One may assume that n is even because this is a requirement for closed shells. It follows from the equations above that all even-length paths are closed shell and an even-length cycle is closed shell if and only if n is not divisible by 4. Hence, the calculation only requires knowledge of n for paths and cycles.

Although not as fast as paths and cycles, the closed shell calculation for trees is also quick. For any bipartite graph, if λ is an eigenvalue then $-\lambda$ is an eigenvalue with the same multiplicity [7, Theorem 3.11]. A direct application of [7, Theorem 1.3] shows that 0 is an eigenvalue if and only if the tree has no perfect matching. Because trees are bipartite, their eigenvalues are symmetric and therefore a tree is closed shell if and only if it has a perfect matching. This reduces the calculation to one taking linear time.

After filtering out the blue-component graphs having fast eigenvalue computations, some graphs remain to be tested using numerical eigenvalue computations. Many methods exist to compute the eigenvalues, but the relatively robust representation (RRR) method was selected for the implementation due to its speed. The DSYEVR subroutine of the LAPACK libraries was used for the computation. The computations were carried out using double precision for the highest precision possible using LAPACK. In practice, there was always a significant gap between the magnitudes of the observed non-zero eigenvalues and those eigenvalues that lay within the specified tolerance of zero. This yields confidence in the accuracy of the computations. The results were also reproduced using matrix inertia computations (using the DSPTRF subroutine to factor the matrix using Bunch-Kaufman diagonal pivoting) and identical conclusions were obtained.

There are ways to improve the run time other than avoiding eigenvalue calculations. Note that the dynamic upper bounds discussed in Section 3.3 become more restrictive as larger-sized sets are found. However, test runs of the algorithm showed that for most isomers large-sized sets are not found quickly. It is desirable to have restrictive bounds so that more partial colorings are eliminated from consideration, thereby decreasing the run time. Property 2.6 states that $\alpha^{-}(G) \leq 2\lfloor n/5 \rfloor$. The test runs indicated that the closed-shell independence number is not far below this value for fullerenes with 100 or fewer vertices. Furthermore, by definition, every closed-shell independent set has an even number of vertices.

This information leads to a method that decreases the run time by looking for a set of a given large size rather than any size. Instead of setting $LargestSeen$ to be 0 initially, it is set to $2\lfloor n/5 \rfloor - 2$. This makes the algorithm search only for sets of size $2\lfloor n/5 \rfloor$. The bounds are then more restrictive and the algorithm runs faster. If such a set is found, then the search can stop immediately because it matches the theoretical upper bound of $\alpha^{-}(G)$. If no such set is found, the algorithm is restarted with $LargestSeen = 2\lfloor n/5 \rfloor - 4$. Then if a set of size $2\lfloor n/5 \rfloor - 2$ is found, the algorithm can stop immediately because it already checked for larger sets. Otherwise, the initial value of $LargestSeen$ will continue to decrease by 2 until a set is found. One disadvantage of this method is that the run time may increase if either the closed-shell independence number is far below $2\lfloor n/5 \rfloor$ or if a large set would be found quickly by the original ($LargestSeen = 0$) method. However, in most cases of interest this method decreased the run time of the algorithm.

n	$x = 6$	$x = 4$	$x = 2$	$x = 0$	n	$x = 6$	$x = 4$	$x = 2$	$x = 0$
20	-	-	-	1	62	-	1	2347	37
24	-	-	-	1	64	-	-	2789	676
26	-	-	1	-	66	-	651	3827	-
28	-	-	-	2	68	-	4	6200	128
30	-	-	3	-	70	-	6005	2144	-
32	-	-	3	3	72	-	1072	10107	11
34	-	-	-	6	74	-	2	13841	403
36	-	-	13	2	76	-	12681	6470	-
38	-	-	2	15	78	-	1308	22773	28
40	-	-	39	1	80	2	30753	1169	-
42	-	-	35	10	82	-	23058	16660	-
44	-	-	2	87	84	-	1641	49854	97
46	-	-	115	1	86	1	60544	3216	-
48	-	-	117	82	88	-	41480	40258	-
50	-	2	268	1	90	1748	98095	75	-
52	-	-	416	21	92	-	118707	7702	-
54	-	-	211	369	94	-	67341	86152	-
56	-	-	924	-	96	1958	189627	254	-
58	-	-	1074	131	98	1	213326	17690	-
60	-	460	1344	8	100	107378	178536	-	-

Table 1: Number of fullerenes C_n with $\alpha^-(G) = 2\lfloor n/5 \rfloor - x$

6 Computational Results

The algorithm was run on all 1,456,598 fullerene isomers on up to 100 vertices. Table 1 shows the number of fullerene isomers on n vertices with $\alpha^-(G)$ a given distance below the theoretical upper bound of $2\lfloor n/5 \rfloor$. This data set is useful for checking how the quality of the upper bound and the spread of $\alpha^-(G)$ vary with increasing n. The bound remains a reasonable estimate as n grows.

A characterization of the fullerenes that maximize $\alpha^-(G)$ for a given n is useful for understanding the closed-shell independent set problem. Notice that $2\lfloor n/5 \rfloor$ increases when $n \equiv 0$ or $n \equiv 6 \pmod{10}$. As expected, Table 1 shows a step away from $2\lfloor n/5 \rfloor$ for most isomers at these values of n. The shifts are somewhat larger at multiples of 10, as expected.

An interesting observation is that there seems to be a correlation between $\alpha^-(G)$ and the isomer number, i.e., the index of the isomer in the list of canonical face spirals [11, §2.6]. For values of n with relatively few isomers having the largest $\alpha^-(G)$, the isomer numbers tend to be larger. Fullerenes with larger isomer numbers tend to have fewer *pentagon adjacencies*: the number of edges incident to two pentagons. Now consider only the values of n in which relatively few isomers achieve the maximum $\alpha^-(G)$: 60, 62, 68, 72, 74, 78, 84, 90 and 96. Of the 1041 isomers that maximize $\alpha^-(G)$ for these values of n, the number of pentagon adjacencies ranges from 0 to 7 with a mean of only 2.68. Note, however, that on these values of n there is a plethora of isomers that have few pentagon adjacencies but do not achieve the maximum $\alpha^-(G)$. Furthermore, for values of n not in this group, the isomer numbers with maximum $\alpha^-(G)$ are more evenly distributed.

The next natural question is whether or not there is a correlation with the isomer number or number of pentagon adjacencies for those isomers that minimize $\alpha^-(G)$ for a given n. Consider only the values of n for which relatively few isomers achieve the minimum $\alpha^-(G)$: 50, 62, 68, 74, 80, 86 and 98. In these cases the isomer numbers have no preference toward higher or lower values, but the number of pentagon adjacencies tends to be high. Of the 13 isomers that minimize $\alpha^-(G)$ for these values of n, the number of pentagon adjacencies ranges from 8 to 20 with a mean of 12.85. However, there are many isomers with a large number of pentagon adjacencies that do not achieve the minimum.

7 Relationship to the Independence Number

There is a strong interest in the relationship between the independence number, $\alpha(G)$, and the closed-shell independence number. The numerical difference between the two is of particular interest. Table 2 summarizes this relationship by giving the number of isomers on n vertices with a given distance between $\alpha^-(G)$ and $\alpha(G)$.

Table 2 shows that there are exactly three isomers in which $\alpha(G) = \alpha^-(G)$. These are C_{20}, $C_{40}:40$, and $C_{60}:1812$[4]. Though the calculations here only go as far as 100 vertices, $\alpha(G)$ has been calculated for all fullerenes on 120 or fewer vertices [12]. These results confirm that no additional fullerenes exist with $\alpha(G) = \alpha^-(G)$ through 120 vertices because all values of $\alpha(G)$ are strictly greater than $2\lfloor n/5 \rfloor$ for fullerenes on at least 100 vertices. Hence, the following conjecture is made.

Conjecture 7.1. *The only fullerene isomers with* $\alpha^-(G) = \alpha(G)$ *are* C_{20}, C_{40}:40, *and* C_{60}:1812.

It has already been shown [12] that the only icosahedral fullerene isomers with $\alpha^-(G) = \alpha(G)$ are C_{20} and C_{60}:1812. Data given in Table 2 and [12] further support Conjecture 7.1 by showing that $\alpha^-(G)$ and $\alpha(G)$ move further away from $2\lfloor n/5 \rfloor$ as n increases.

The three isomers in Conjecture 7.1 have several things in common. They each have $\alpha^-(G) = 2n/5$. Therefore, Property 3.1 implies that every blue component is a copy of K_2. The fullerene C_{20} has only one non-isomorphic maximum closed-shell independent set, since the selection of two vertices for a blue component completely determines the set.

[4]C_{60}:1812 is the icosahedral Buckminsterfullerene often referred to simply as C_{60}.

n	$x=0$	$x=1$	$x=2$	$x=3$	$x=4$	$x=5$	$x=6$	$x=7$	$x=8$	$x=9$	$x=10$	$x=11$	$x=12$	$x=13$	$x=14$
20	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-
24	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
26	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-
28	-	1	1	-	-	-	-	-	-	-	-	-	-	-	-
30	-	-	2	1	-	-	-	-	-	-	-	-	-	-	-
32	-	2	1	1	2	-	-	-	-	-	-	-	-	-	-
34	-	-	5	1	-	-	-	-	-	-	-	-	-	-	-
36	-	2	-	12	1	-	-	-	-	-	-	-	-	-	-
38	-	-	13	2	2	-	-	-	-	-	-	-	-	-	-
40	1	-	-	36	3	-	-	-	-	-	-	-	-	-	-
42	-	-	10	-	32	3	-	-	-	-	-	-	-	-	-
44	-	-	3	79	5	1	1	-	-	-	-	-	-	-	-
46	-	-	1	10	101	4	-	-	-	-	-	-	-	-	-
48	-	-	8	74	9	103	5	-	-	-	-	-	-	-	-
50	-	-	1	25	238	5	2	-	-	-	-	-	-	-	-
52	-	-	6	15	77	332	7	-	-	-	-	-	-	-	-
54	-	-	1	100	268	22	180	9	-	-	-	-	-	-	-
56	-	-	-	1	257	656	10	-	-	-	-	-	-	-	-
58	-	-	2	84	45	295	771	8	-	-	-	-	-	-	-
60	1	3	4	3	644	697	88	361	11	-	-	-	-	-	-
62	-	-	2	32	7	1049	1284	10	1	-	-	-	-	-	-
64	-	-	-	18	521	137	1278	1494	17	-	-	-	-	-	-
66	-	-	-	-	24	2370	1433	188	448	15	-	-	-	-	-
68	-	-	-	13	112	49	3843	2295	16	2	2	-	-	-	-
70	-	-	-	1	81	1817	261	3578	2394	17	-	-	-	-	-
72	-	-	-	6	6	199	7358	2549	433	615	24	-	-	-	-
74	-	-	-	-	100	297	261	9957	3611	18	2	-	-	-	-
76	-	-	-	-	1	580	5469	540	8845	3688	28	-	-	-	-
78	-	-	-	3	25	-	1104	17747	3923	642	639	26	-	-	-
80	-	-	-	-	1	416	747	1677	23734	5318	29	2	-	-	-
82	-	-	-	-	-	3	2669	13303	1261	17283	5168	31	-	-	-
84	-	-	-	-	13	77	11	5496	38503	5858	908	686	40	-	-
86	-	-	-	-	-	7	1697	1502	6773	46497	7255	29	1	-	-
88	-	-	-	-	-	-	27	10827	28314	3674	31798	7055	43	-	-
90	-	-	-	-	-	8	64	37	19135	70685	8258	1026	658	47	-
92	-	-	-	-	-	-	80	5079	2556	23748	84998	9901	47	-	-
94	-	-	-	-	-	-	-	150	32555	51897	9011	50764	9066	50	-
96	-	-	-	-	-	-	59	181	283	57477	120769	11157	1195	659	59
98	-	-	-	-	-	-	-	318	13253	4225	63871	136465	12841	43	1
100	-	-	-	-	-	-	-	1	1040	86672	88633	21709	76178	11611	70

Table 2: Number of fullerenes C_n with $\alpha(G) - \alpha^{-}(G) = x$

In [13], uniqueness (up to isomorphism) was shown for C_{60}:1812. Enumeration verifies that C_{40}:40 also has a unique maximum-sized set.

These isomers differ in the distribution of their pentagons. The dodecahedron C_{20} has only 12 faces, all of which are pentagons. The pentagons of C_{40}:40 are in four groups of 3, evenly spaced throughout the fullerene. The truncated icosahedron C_{60}:1812 has isolated pentagons evenly spaced throughout the fullerene. The isomer C_{40}:40 has 24 automorphisms and its point group is T_d, whereas the others have 120 automorphisms and point group I_h.

8 Future Work

This paper has described an algorithm with an implementation that is fast, despite all of the eigenvalue calculations. This enabled the calculation of the closed-shell independence number of all fullerene isomers with 100 and fewer vertices. From this data set, some interesting observations were made including a correlation with pentagon adjacencies. Most notable is the conjecture that there are only three fullerene isomers that have a maximum independent set that is closed shell.

Several opportunities exist to extend this work, including a more comprehensive data analysis and speeding up the implementation to obtain fast calculations for larger fullerenes. A faster implementation may be possible by the use of better data structures or solving some of the open problems below. It is desirable to have calculations that extend as far as the 120-vertex fullerenes. In the initial implementation, the calculation for a single fullerene with 120 vertices took about 12 minutes, compared to about 24 seconds for a 100 vertex fullerene. At that rate, calculations for the 1,674,171 fullerenes only of size 120 would take approximately 38 CPU years.

Additionally, the following problems remain open:

1. Identify infinite families of fullerenes that maximize or minimize $\alpha^-(G)$ over all isomers of the associated n.

2. In addition to paths, cycles, and trees, identify classes of graphs that are easily recognizable and for which the closed-shell computation is fast.

3. Continue the calculations on larger icosahedral fullerenes. These are frequently-studied highly-symmetrical models of general fullerenes.

4. Although most fullerenes have no symmetry, taking advantage of any symmetry that is present is expected to speed up the algorithm by ignoring isomorphic sets. This would greatly aid the search for sets on large icosahedral fullerenes.

References

[1] I. Bomze, M. Budinich, P. Pardalos, and M. Pelillo. The maximum clique problem. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume 4. Boston, MA: Kluwer Academic Publishers, 1999.

[2] G. Brinkmann and A. W. M. Dress. A constructive enumeration of fullerenes. *Journal of Algorithms*, 23:345–358, 1997.

[3] G. Brinkmann and A. W. M. Dress. Penthex puzzles: a reliable and efficient top-down approach to fullerene-structure enumeration. *Advances in Applied Mathematics*, 21:473–480, 1998.

[4] V. Chvátal. Determining the stability number of a graph. *SIAM Journal on Computing*, 6:643–662, 1977.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* Cambridge, MA: MIT Press, 2nd edition, 2001.

[6] J. D. Crane. Maximal non-adjacent addition to fullerene-70: Computation of all the closed shell isomers of $C_{70}X_{26}$. *Fullerene Science and Technology*, 7:427–435, 1999.

[7] D. M. Cvetković, M. Doob, and H. Sachs. *Spectra of Graphs: Theory and Application.* New York: Academic Press, 1980.

[8] S. Fajtlowicz and C. E. Larson. Graph-theoretic independence as a predictor of fullerene stability. *Chemical Physics Letters*, 377:485–490, 2003.

[9] P. W. Fowler, S. Daugherty, and W. Myrvold. Independence number and fullerene stability. Preprint, 19 pages, 2007.

[10] P. W. Fowler, P. Hansen, K. M. Rogers, and S. Fajtlowicz. $C_{60}Br_{24}$ as a chemical illustration of graph theoretical independence. *Journal of the Chemical Society, Perkin Transactions 2*, pages 1531–1534, 1998.

[11] P. W. Fowler and D. E. Manolopoulos. *An Atlas of Fullerenes.* Oxford: Clarendon Press, 1995.

[12] P. W. Fowler and W. Myrvold. Some chemical applications of independent sets. In P. Hansen, N. Mladenovic, J. A. M. Perez, B. M. Batista, and J. M. Moreno-Vega, editors, *Proceedings of the 18th Mini Euro Conference on Variable Neighborhood Search*, Tenerife, Spain, 2005.

[13] P. W. Fowler, K. M. Rogers, K. R. Somers, and A. Troisi. Independent sets and the prediction of additional patterns for higher fullerenes. *Journal of the Chemical Society, Perkin Transactions 2*, pages 2023–2027, 1999.

[14] J. E. Graver. The independence number of fullerenes and benzenoids. *European Journal of Combinatorics*, 27:850–863, 2006.

[15] D. S. Johnson and M. A. Trick, editors. *Cliques, Colouring, and Satisfiability, Second DIMACS Implementation Challenge, Oct. 11-13, 1993*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. AMS, 1996.

[16] W. Myrvold and P. W. Fowler. Fast enumeration of all independent sets of a graph. Preprint, 10 pages, 2005.

[17] J. M. Robson. Algorithms for maximum independent sets. *Journal of Algorithms*, 7:425–440, 1986.

[18] D. H. Rouvray. Chapter 7. In A. T. Balaban, editor, *Chemical Applications of Graph Theory*. London: Academic Press, 1976.

[19] A. Streitwieser, Jr. *Molecular Orbital Theory for Organic Chemists*. New York: John Wiley & Sons, 1961.