2006

Graph reconstruction numbers

Jennifer Baldwin

Master's Project Report
Graph Reconstruction Numbers

Jennifer Baldwin

jlb5851@cs.rit.edu

Rochester Institute of Technology

Department of Computer Science

102 Lomb Memorial Drive

Rochester, NY 14623-5608

Chairman: Edith Hemaspaandra
Reader: Stanisław P. Radziszowski
Observer: Darren Narayan
July 30, 2004

Abstract

One of the most important open questions in graph theory is the graph reconstruction conjecture, first proposed by P. J. Kelly and S. M. Ulam in 1941. The conjecture states that every graph with at least 3 vertices is reconstructible, i.e., if there exists some multi-subset of vertex-deleted subgraphs that reconstructs G uniquely up to isomorphism. This project computes the existential and universal reconstruction numbers for all graphs of order at most eight and 26,000 graphs of order nine. The existential reconstruction number is the number of vertex-deleted subgraphs, or cards, required to reconstruct G uniquely up to isomorphism. The universal reconstruction number is the minimum number of cards for which all multi-subsets of that size reconstruct G uniquely up to isomorphism.

The graph reconstruction conjecture is still unproven, but the results of this project help provide more information. Many theorems relating to the existential reconstruction number were verified by my results. We also refute a conjecture of Harary and Plantholt. The conjecture stated that the upper bound of the existential reconstruction number is $\frac{n}{2} + 1$. We show two graphs that have an existential reconstruction number of 6 when the upper bound defined conjectured by Harary and Plantholt is 5. The universal reconstruction number has been researched very little so far. The reconstruction numbers produced by this project help support the opinion that most graphs are easily reconstructible.

Contents

List of Figures

List of Tables

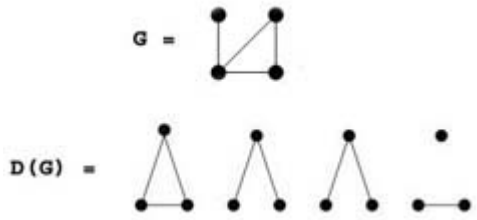


Figure 1: Example Graph and its Deck

1 Background

In order to understand what graph reconstruction is, one must first understand what a deck and a card is. Given a graph G, a card represents the subgraph created by deleting a single vertex of G. A deck for graph G, denoted $D(G)$ is the multi-set containing all the vertex-deleted subgraphs of G [3, 13, 8]. The deck is considered a multi-set because it is possible that $G - v_i \cong G - v_j$, but the two subgraphs were created by removing different vertices and are therefore defined as different cards. A graph G is reconstructible if there exists some multi-subset of the deck that can be used to create a unique graph H that is isomorphic to G. H is considered unique up to isomorphism if all of the graphs that can be constructed from its multi-subset of cards have the same canonical labeling. A graph G is isomorphic to a graph H, if there exists a bijection $\sigma : V(G) \to V(H)$ such that an edge $uv \in E(G) \iff \sigma(u)\sigma(v) \in E(H)$. In other words, there must exist a function for which every edge in G corresponds to a single edge in H. A canonical labeling is a way of relabeling the vertices of G, while retaining G's connectivity. There exist many algorithms that have the purpose of canonically labeling a graph. If two graphs have the same canonical labeling, then they are isomorphic. Each algorithm approaches the problem in a different way. Another way of determining that G is unique is via a top-down approach. If no other graph of the same order as G has a multi-subset of cards equivalent to the one that creates G, then G is unique up to isomorphism. Figure 1 shows a graph G and its deck. Note that in this case G is the only graph up to isomorphism with deck $D(G)$.

The Reconstruction Conjecture (**P.J. Kelly, [7], and S.M. Ulam, [14]**):
Every graph with at least three vertices is reconstructible [3, 8, 1, 13].

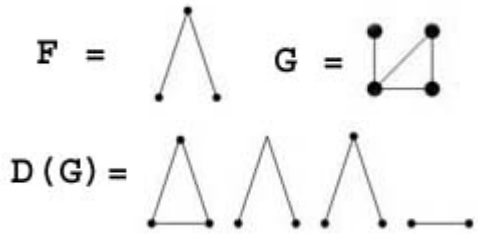
Since the conjecture has been introduced many special classes of the conjecture have been proven as well as lemmas and theorems which are useful in verifying reconstruction.

Kelly's Lemma: For any two graphs F and G such that $|V(F)| < |V(G)|$, the number of subgraphs of G isomorphic to F, $s(F, G)$, can be determined from $D(G)$, where $|V(F)|$ is the size of the vertex set of F [3, 13].

Proof: Given $|V(F)| < |V(G)|$ each subgraph that is isomorphic to F occurs in exactly $|V(G)|-|V(F)|$ of the vertex-deleted subgraphs $G-v$.

$$s(F,G) = \frac{1}{|V(G)| - |V(F)|} \sum_{v \in V} s(F, G - v)$$

The summation is divided by $|V(G)| - |V(F)|$ since each copy of an isomorphic subgraph is counted this many times. The following is an example:



$$\begin{aligned} s(F,G) &= \frac{1}{4-3} * (s(F, G - v_1) + s(F, G - v_2) + s(F, G - v_3) + s(F, G - v_4)) \\ &= 0 + 1 + 1 + 0 \\ &= 2 \end{aligned}$$

Theorem 1: For any two graphs F and G such that $|V(F)| < |V(G)|$, the number of subgraphs of G that are isomorphic to F and contain a given vertex v can be computed from $D(G)$ [3].

Proof: This number is $s(F, G) - s(F, G - v)$. The value of $s(F, G - v)$ is the number of subgraphs of G that are isomorphic to F and do not include the vertex v, and can be computed given the card $G - v$.

$$
\begin{aligned}
s(K_2, G) &= \frac{1}{4-2} * (s(K_2, G - v_1) + s(K_2, G - v_2) + s(K, G - v_3) + s(K, G - v_4)) \\
&= \frac{1}{2} * (3 + 2 + 2 + 1) \\
&= \frac{1}{2} * 8 \\
&= 4 \\
d(v_1) &= s(F, G) - s(F, G - v_1) \\
&= 4 - 3 \\
&= 1 \\
d(v_2) &= s(F, G) - s(F, G - v_2) \\
&= 4 - 2 \\
&= 2 \\
d(v_3) &= s(F, G) - s(F, G - v_3) \\
&= 4 - 2 \\
&= 2 \\
d(v_4) &= s(F, G) - s(F, G - v_4) \\
&= 4 - 1 \\
&= 3
\end{aligned}
$$

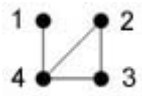
$G =$ 

Figure 2: Reconstruction of G

Theorem 2: The number of edges and the degree sequence of a graph are reconstructible [3]. The degree of a vertex v is the number of edges incident to v. The degree sequence is an ordered list of the degree of each vertex of a graph G.

Proof: Let $F = K_2$, an edge. The number of edges in G can be determined by calculating $s(F, G)$. The degree sequence of the graph can be determined by calculating $s(F, G) - s(F, G - v)$ for each vertex.

Figure 2 shows a reconstruction of the previous graph using the information provided by the above theorems. Note that that the degree sequence is not always enough to reconstruct a graph uniquely up to isomorphism.

Theorem 3: Regular graphs, graphs with all vertices of the same degree, are reconstructible.

Proof: Let G be a k-regular graph. The degree sequence is reconstructible and will show that all vertices have the same degree, k, thus reconstructing G up to isomorphism.

Theorem 4: Disconnected graphs are reconstructible [3, 13].

Proof Sketch: Given a graph G of order ≥ 3 and its deck, G is disconnected if and only if at most one card in the deck is a connected graph. If G is disconnected, then each component will appear in at least one card of G's deck. The largest component of G is the largest component of all of G's cards. Once this component is identified, the other components can be identified and reconstructed.

One new area of research related to the reconstruction conjecture is determining the reconstruction number of a graph. The reconstruction number of a graph G, denoted by $\exists rn(G)$, is the number of cards required to reconstruct the original graph up to isomorphism [6]. Figures 3-7 show why $\exists rn(G) \neq 2$. Although it is obvious that $\exists rn(G)$ cannot be 2, the approach used in figures 3- 7 are shown since the approach mimics the one used in my program. The value of $\exists rn(G)$ can never be 2 since there will always be at least two graphs that can be created from the deck. The difference between the two graphs will be one edge. The extension graphs shown in figure 4 are graphs created by extending each card of G's deck by one vertex. The first

graph in the figure is the graph being extended. In the attempts to reconstruct G using only 2 cards, figures 5-7, 2 cards are chosen from $D(G)$. The cards chosen are the first graph on each row. The remaining graphs is the set of extensions graphs for that one graph. In each figure a different subdeck of size 2 is chosen. Once a subdeck is chosen, the extension graphs of each card are compared. The graphs that can be recreated from the subdeck are graphs contained in the intersection of the set of extension graphs of each card. In the figures these graphs have been circled in red. In the first three attempts, both decks have at least two non-isomorphic graphs in common, therefore $\exists rn(G) \geq 3$. The actual value of $\exists rn(G)$ is 3, and the cards used to reconstruct G are the three unique graphs within G's deck as shown in figure 8.

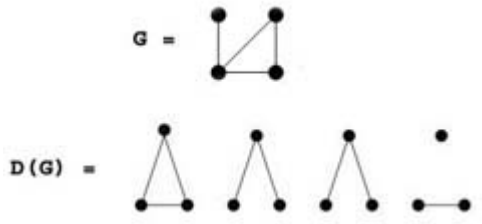


Figure 3: G and $D(G)$

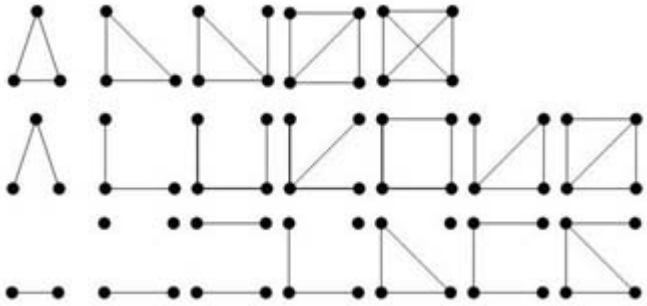


Figure 4: Extension graphs of $D(G)$

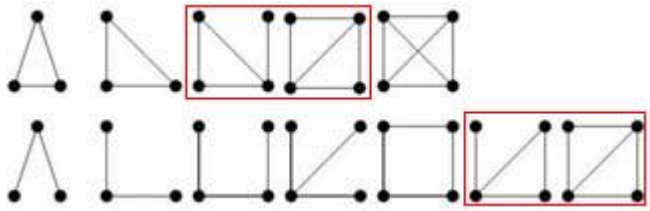


Figure 5: Reconstruction attempt #1

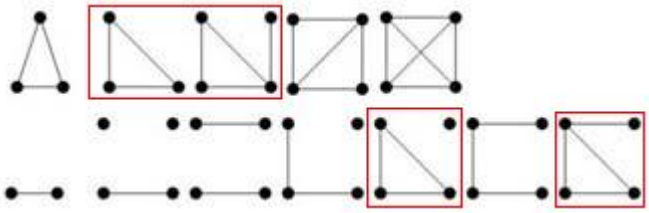


Figure 6: Reconstruction attempt #2

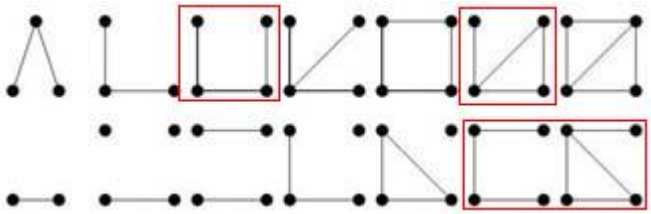


Figure 7: Reconstruction attempt #3

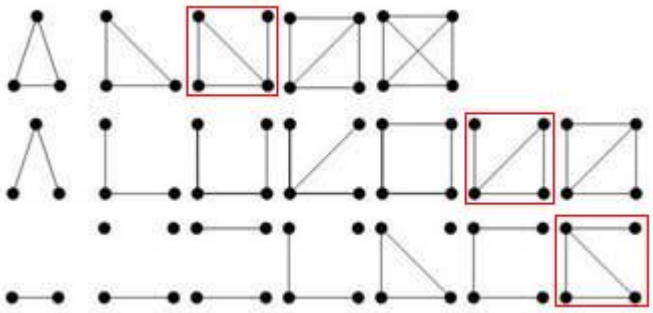


Figure 8: Reconstruction attempt #4

Conjecture (**Harary, Plantholt**): The reconstruction number, $\exists rn(G)$, of any graph G is at most $\frac{n}{2} + 1$ [12, 11].

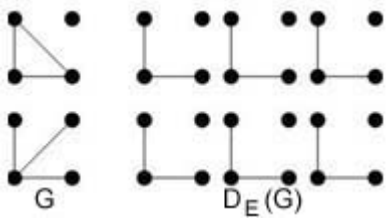
Bollobás proved in [2] via probability and statistics that almost all graphs G have $\exists rn(G) \geq 3$.

Theorem 5 (**Myrvold**): Disconnected graphs with at least two non-isomorphic components have a reconstruction number of 3 [12, 11].

Myrvold determined a way to choose the 3 cards based on the properties of disconnected graphs and their vertex-deleted subgraphs.

Theorem 6 (**Myrvold**): The reconstruction number of disconnected graphs with all isomorphic components is less than or equal to $c + 2$ where c is the order of the component [12, 11].

Another conjecture that has its basis in the Reconstruction Conjecture is the Edge Reconstruction Conjecture. The idea behind this conjecture is similar, attempt to reconstruct a graph from its collection of edge-deleted subgraphs. The minimum number of edges required for a graph to be reconstructible is 4. As can be seen in figure 9 there exist two non-isomorphic graphs with 3 edges that have the same deck, denoted $D_E(G)$. As a consequence the Edge Reconstruction Conjecture requires that $|E(G)| \geq 4$.



The edge decks for the above two graphs are the same, therefore the number of edges must be greater than 3.

Figure 9: Two non-isomorphic graphs with the same edge deck

There has also been research conducted to validate the conjecture in a computational manner. Brendan McKay created a program that aids in achieving this goal. The name of the program is *nauty*, and it is able to canonically label graphs. Once a graph has been canonically labeled it is easier to identify graph isomorphisms. In McKay's reconstruction program the decks of all graphs were created first. Once a deck was created each graph within it was canonically labeled. After all of the decks had been relabeled they were parsed for duplicates. A duplicate deck would be a counterexample to the Reconstruction Conjecture. McKay's research looked at 5 different classes of graphs: all graphs with maximum order 11, all graphs of order 12 and maximum vertex degree 5, all triangle-free graphs with maximum order 14, and all square-free graphs with maximum order 15. Within each of these classes no contradiction to the conjecture was found [9].

2 General Project Description

The purpose of this project is to computationally find minimum reconstruction numbers ($\exists rn$) as well as universal reconstruction numbers. ($\forall rn$) for all graphs with at most order 8. Universal reconstruction numbers were introduced in [6].

Definitions:

$\exists rn(G)$: The minimum reconstruction number, i.e., the size of a single multi-subset that uniquely reconstructs G.

$\forall rn(G)$: The minimum multi-subset size, i.e., the size for which all multi-subsets of that size uniquely reconstruct G.

In the original project proposal 3 modules were proposed, one to create the input data, one to compute $\exists rn(G)$, and one to compute $\forall rn(G)$. The end result ended up being composed of two modules, one that created the input data and one that calculated both reconstruction numbers. Also the proposal stated that all graphs with order at most 11 would be worked with. Unfortunately due to time limitations the program could only be run on graphs with order at most 8. Each graph of order 9 takes anywhere from one second to one minute to complete computation, but there are 274.668 graphs of order 9

so full computation is timely. As a result only approximately 26,000 graphs have been analyzed at this time.

3 Implementation

Instrumental to the completion of this project was the availability of the *nauty* program. Without this program which identifies isomorphisms, I would not have been able to verify that G is reconstructible up to isomorphism. Also bundled with the *nauty* program was a package called *gtools*. I found many of these tools, such as *labelg*, *showg*, and *amtog*, very useful in the the computation of reconstruction numbers and the validation of my results [10]. Since these tools were stand-alone programs, I ended up copying some of the code so I could use these programs as functions. Since McKay had already written code to read and print graphs, I used these function within my programs as well.

One of the first decisions that needed to be made concerning this project was how graphs were going to be represented within the program and also when stored in a file. Should they be stored in the same format? Should the formats be different and conversion functions be written?

When thinking of graphs, the first two representations that come to mind are adjacency matrices and adjacency lists. A matrix is easier to traverse within a program, but a list is easier to store. Both matrices and lists will require a lot of disk space when stored in a file, especially when dealing with larger graphs. As a result the decision to use two different graph representations was made. The representation used for storing a graph in a file was to be small and compact, and the representation for use within the programs was to be easy to traverse and manipulate.

For the file representation the decision to use *nauty*'s graph6 format was quickly made. This format is an ASCII representation of the upper triangle of a graph's adjacency matrix. The upper triangle is converted to a bit string which is then converted to ASCII. The length of the bit vector is $\frac{n*(n-1)}{2}$. Each ASCII character represents 6 bits of the bit vector. If the length of the vector is not a multiple of 6, then the vector is padded on the right with 0's [10]. An example of the conversion from matrix to bit string is provided in figure 10.

Given that graph6 format prints the graph given a bit vector representation, I tried to use this bit vector as my internal graph representation.

Figure 10: Conversion from matrix to bit vector to graph6 format

Although conversion between the two formats was very easy, manipulation of the vector itself proved more difficult. The vector listed information column by column from the adjacency matrix. As a result adding or deleting a vertex from a graph required bits to be rearranged in multiple places within the vector. The number of bits that needed to be added or deleted was also not constant, since only the upper triangle of the matrix was represented. Figure 11 shows an example of what needs to change when a vertex is added to a graph.

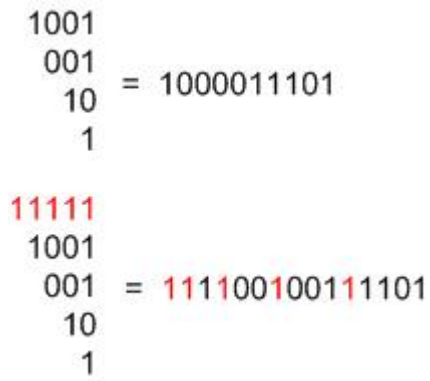


Figure 11: Adding a vertex to the bit vector representation

Figure 12 shows an example of what needs to change when a vertex is deleted from a graph.

As can be seen in the examples, the process is not trivial. A lot of bit shifting is required and an algorithm is needed to determine which bits are affected.

In the interest of saving time I decided to use a complete adjacency matrix

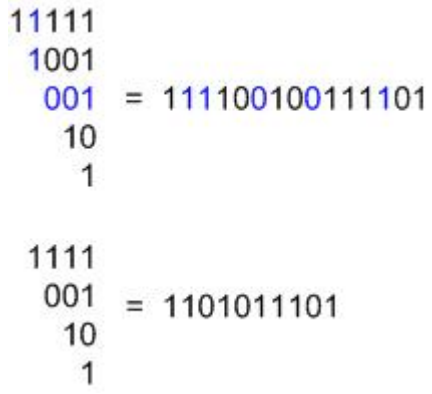


Figure 12: Removing a vertex from the bit vector representation

for my internal representation. McKay's *gtools* included two programs which performed the conversions that I required. One of these programs is *showg* which reads a graph in in graph6 format and returns an adjacency list or matrix. The other program is *amtog* which reads in an adjacency matrix and returns the graph in graph6 format [10]. Since both of these programs were designed as stand-along programs, I copied the code that I needed to perform the conversions and created two functions.

Once the decision of graph representation was made, the generation of information needed to reconstruct a graph needed to be completed. Since there are two distinct pieces of information needed, the generation of information was split into two programs. The program named *expand* generates all extensions of a graph by one vertex. Each line of input is a different graph for which the extension graphs are to be created. The program named *shrink* generates all the vertex-deleted subgraphs of a graph G. This program also reads in graphs to "shrink" one at a time. Since the internal graph representation was an adjacency matrix, these two tasks were quite simple.

Extensions are made by first copying the original graph into a $n+1$ x $n+1$ in rows 1 to n and columns 1 to n. Row and column 0 are left empty for the vertex that is to be added. There are 2^n possible extensions of a graph by a single vertex. Isomorphism is not a concern at this time so all graphs are output, even if they exist. Figure 13 shows an example run of this program. The first line of the run is the input, the graph from which the extension graphs will be created. Figure 14 is a key for the graph6 format

and shows the actual graphs with its graph6 format below. Looking at the key you can verify that each graph is an extension of Cr, the input, by one vertex. Since *expand* may be extending multiple graphs the original graph is output before the computed graph to keep the data organized and easily identified.

```
bash-2.05$ expand
Cr
DIK
DBw
Dd[
DBw
Dd[
DFw
Dr[
DBw
DFw
Dd[
Dr[
Dd[
Dr[
Dr[
Dr{
```

Figure 13: Example run of *expand*

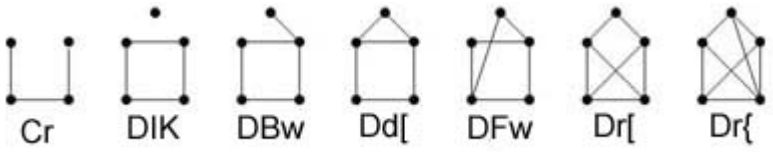


Figure 14: Key of graphs in the example run of *expand*

Subgraphs are created by shifting the rows and columns up and left as necessary. The last row and column are then zeroed to guarantee that their information is not harmful. Figures 15 and 16 show an example run and

the corresponding graph key respectively. Once again isomorphism is not a concern at this point so all graphs are output. Also, the original graph is output. Multiple graphs can be processed by this program. By including the original graph in the output data relating to that graph can be easily identified.

```
bash-2.05$ shrink
DFw
Cr
Cr
Cr
CF
CF
```

Figure 15: Example run of *shrink*

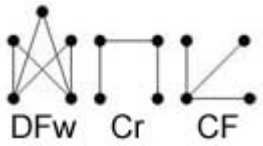


Figure 16: Key of graphs in the example run of *shrink*

At this point isomorphism is not a concern since these graphs are not the actual reconstructions. Even though isomorphism is not a concern, it can be used to help reduce file sizes. To help reduce the sizes I created a program called *makeGraph* for the purpose of pruning the output files of *expand* and *shrink*. This program needed to know the size of the original graph and the computed graphs in order to calculate how many lines need to be read before a tally is output. Of all the computed graphs read in, each unique graph is only output once. Along with each unique graph a count of how many times it was read is also output. Also, the original graph is output as before. Figure 17 shows and example run where the input is the same as the output in figure 13.

Now that the information is readily available, the process of computing the reconstruction numbers can be approached. This process consists of 2

```
bash-2.05$ makeGraph 4 5 g e.out
Cr DIK 1
Cr DBw 4
Cr Dd[ 4
Cr DFw 2
Cr Dr[ 4
Cr Dr{ 1
```

Figure 17: Example run of *makeGraph*

parts, computing $\exists rn(G)$ and computing $\forall rn(G)$. It is best to complete the computations in this order otherwise a lot of computation would be repeated. Once $\exists rn(G)$ has been determined the search for $\forall rn(G)$ can begin starting with a deck size equal to $\exists rn(G)$. The actual algorithms to compute these two numbers are quite similar. This is a result of their similarity in meaning. Figures 18 and 19 detail the algorithms used for computing $\exists rn(G)$ and $\forall rn(G)$. Before these algorithms are used though, the decks of the graph to be reconstructed and the extension graphs need to be created. Figure 20 describes how this is accomplished. Figure 21 shows an example run of the *reconstruct* program. This program determines the values of both reconstruction numbers, $\exists rn(G)$ and $\forall rn(G)$. The arguments required by the program are the names of the files that contain the extension graphs and subgraphs needed to for reconstruction. If the reconstruction numbers for multiple graphs are being computed in a single run, the extension graphs for all of those graphs must be contained in a single extension graph file. The same is true for the subgraphs of all of the graphs to be examined. Each line of input is a graph for which to compute the reconstruction numbers.

For subdeck_size = 2 to $n-1$
  For each unique subdeck of that size
    Verify whether or not any deck of any extension graph contains the current subdeck
    If no other deck contains this subdeck, break
  end
end

The value of $\exists rn(G)$ is the current subdeck size and the current subdeck reconstructs G up to isomorphism.

Figure 18: Algorithm for determining $\exists rn(G)$

For subdeck_size = $\exists rn(G)$ to $n-1$
  Verify whether or not any decks of any extension graphs contain any of the subdeck of this size
  If none of the decks of any of the extension graphs contain any of the subdecks of this size break.

The value of $\forall rn(G)$ is the current subdeck size.

Figure 19: Algorithm for determining $\forall rn(G)$

1. Read in a graph G
2. Find its subgraphs in the provided subgraph file and store them in an array.
3. For each subgraph:
   Find its expansions in the provided expansion file and store them in a common array.
4. For each extension graph:
   Find its subgraphs in the provided subgraph file
   In a matrix row for each unique subgraph mark the number of copies that G's deck also contains with a 1

Figure 20: Description of preliminary steps for computing reconstruction numbers

```
bash-2.05$ reconstruct extensionGraphs subgraphs
Please enter a graph in graph6 format, ^D to exit
E`Kw
E`Kw: DJ[ DJ[ D`K D`K D`K D`K
ern = 3 using cards C1 C2 C3 arn = 5

Please enter a graph in graph6 format, ^D to exit
```

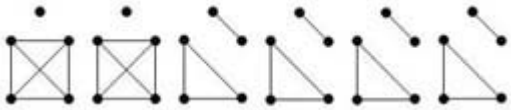
Figure 21: Example run of *reconstruct*

To better understand how the reconstruction numbers are computed I've drawn up an example that goes through the process step by step. It is detailed below.

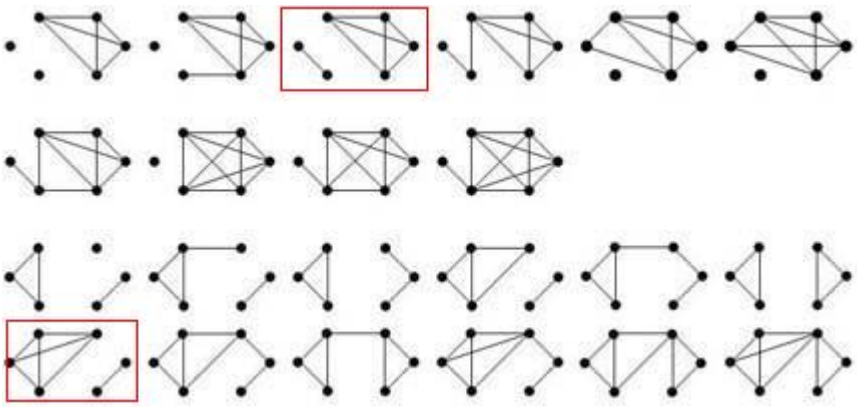
Step 0 Input graph G

$$G = K_2 \cup K_4 = \text{E`Kw} =$$ 

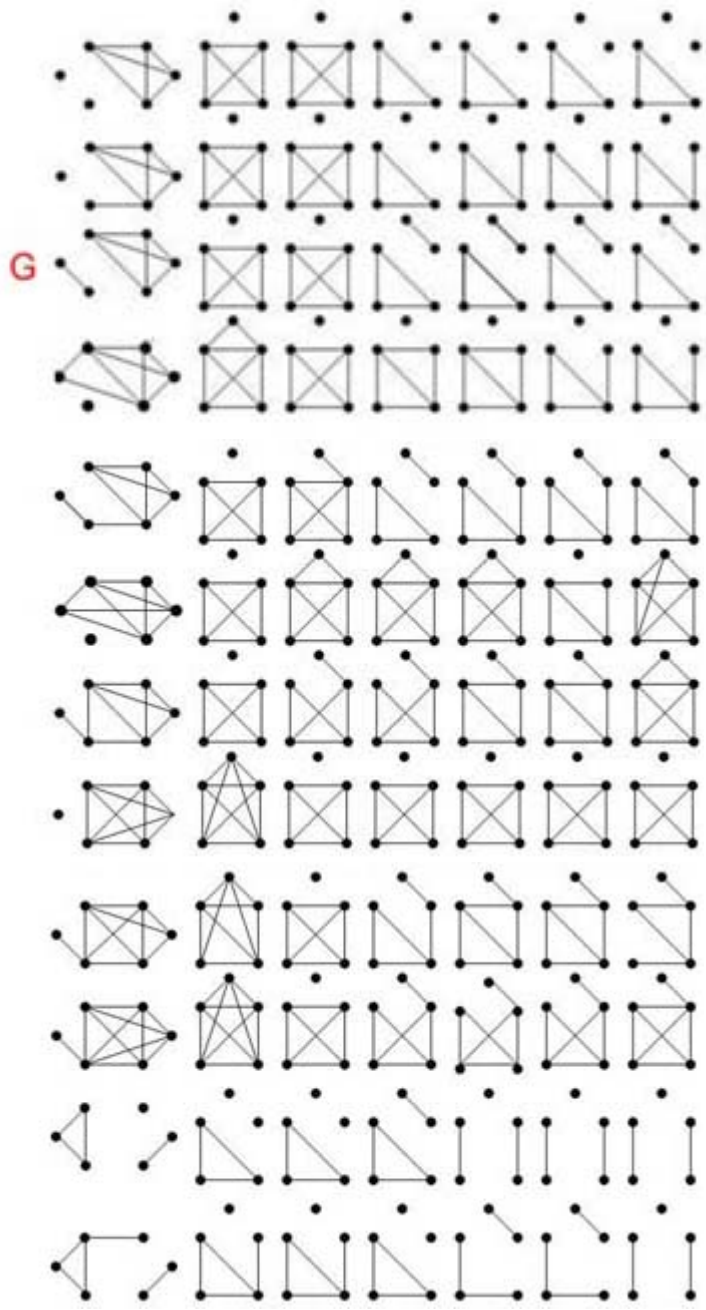
Step 1 Construct $D(G)$, the deck of G

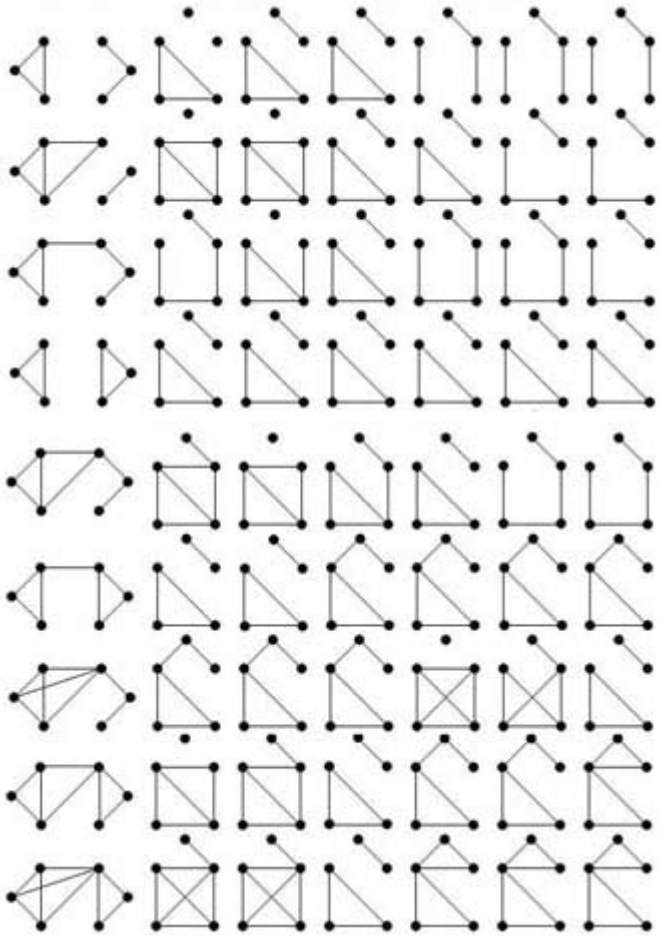


Step 2 Determine what other graphs may be recreated by the cards in $D(G)$, i.e., determine which graphs also contain at least one of these cards in their deck. These graphs can be determined by constructing the extension graphs of each card. The first two rows detail the graphs that are extensions of the first unique card. The last two rows detain the graphs that are extension of the second unique card. The original graph, G, is outlined in red within the sets of extension graphs.



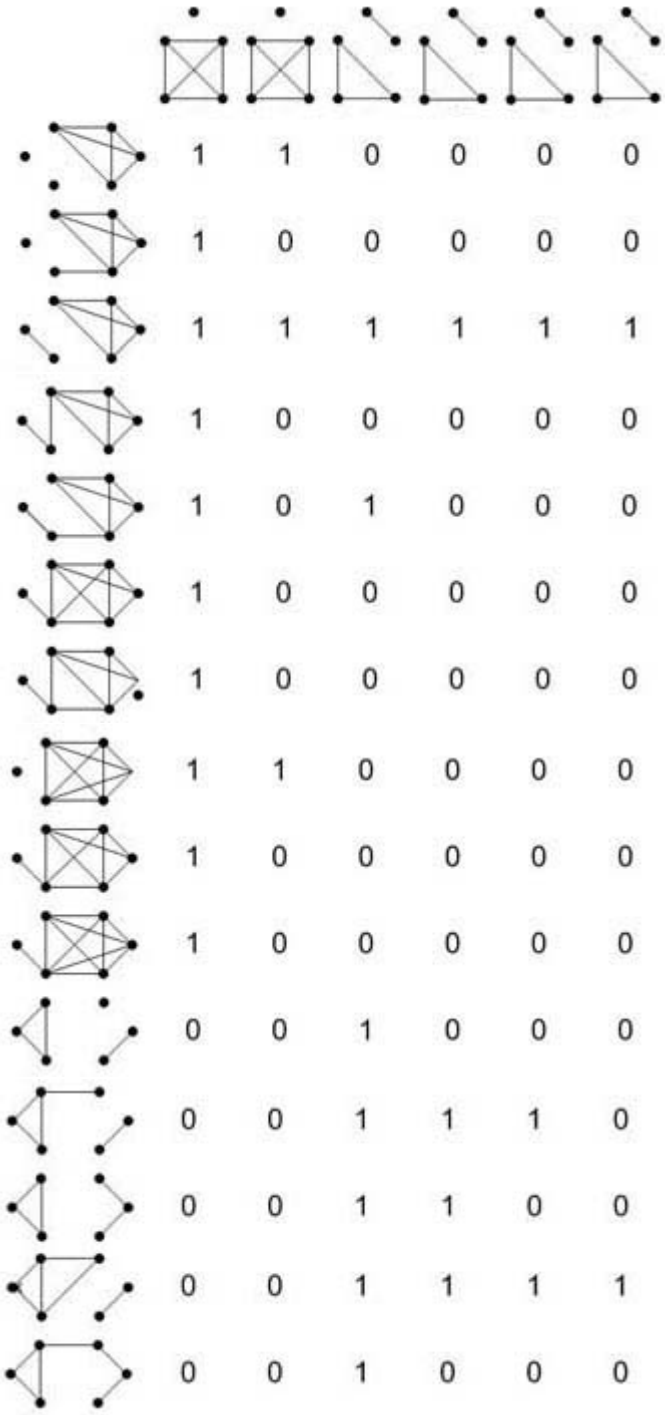
Step 3 Determine the deck of each extension graph. In the following diagram each row consists of an extension graph followed by its deck.



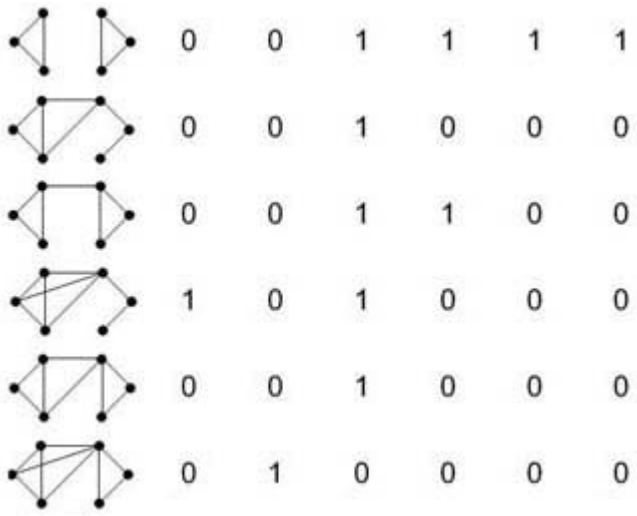


Step 4 Relate each extension graph's deck to G's deck by determining the intersection of the two decks. Add a row to the relation matrix where a 1 represents the fact that the card is in the intersection of the two decks. A 0 represents the fact that the card is not in the intersection.

The rows are left justified. For example, the deck of the graph containing two copies of K_2 is composed of 6 isomorphic cards. The graph being recreated only has 4 copies of this card. As a result the row is: 0 0 1 1 1 1.

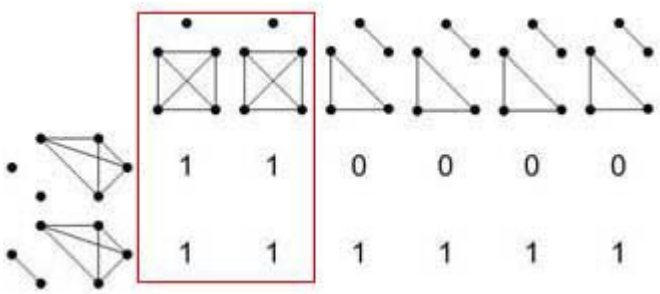


	1	1	0	0	0	0
	1	0	0	0	0	0
	1	1	1	1	1	1
	1	0	0	0	0	0
	1	0	1	0	0	0
	1	0	0	0	0	0
	1	0	0	0	0	0
	1	1	0	0	0	0
	1	0	0	0	0	0
	1	0	0	0	0	0
	0	0	1	0	0	0
	0	0	1	1	1	0
	0	0	1	1	0	0
	0	0	1	1	1	1
	0	0	1	0	0	0



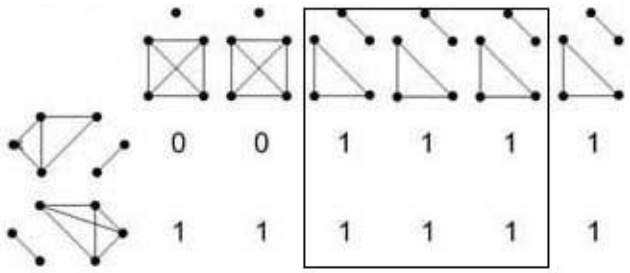
Step 5 For each possible subdeck size and each unique subdeck of that size, verify whether or not the subdeck reconstructs G up to isomorphism. This is equivalent to comparing the columns representing the chosen cards in the subdeck for each row to the same columns in G's row, i.e, a row of all 1's. If none of the rows are equal to the row representing G this subdeck reconstructs G uniquely up to isomorphism. For example, the complete deck of G uniquely reconstructs G, since its row is the only row with all 1's. For this algorithm subdecks are searched in increasing size so the first subdeck that uniquely reconstructs G is the minimum subdeck which does so. As a result the value of $\exists rn(G)$ is the size of the current subdeck.

The following diagram shows an example of a subdeck that does not reconstruct G uniquely up to isomorphism. The top row is a reference to what $D(G)$ is. The first graph of rows two and three is the graph whose deck is being compared to G's deck. The subdeck being verified is outlined in red.



Step 6 For each possible subdeck size greater than or equal to $\exists rn(G)$ verify whether or not all unique subdecks of that size reconstruct G uniquely up to isomorphism. Again check the rows of the relation matrix as was done in step 4. If none of the rows matches G's row, then $\forall rn(G)$ is the current subdeck size.

The value of $\forall rn(G)$ is 4. It cannot be 3 since there exists at least one subdeck of that size that does not reconstruct G uniquely up to isomorphism as shown below.



4 Results

To determine the correctness of all of the programs created, random reconstructions were validated by hand. Before the reconstruction program was even run though, *expand* and *shrink* were vigorously tested. For the smaller order graphs (orders 3-5) all expansion graphs and subgraphs were validated by hand. At that point the programs were deemed to be correct. Graphs were chosen at random from certain classes of results. The results were looked at based on each reconstruction number alone and also paired together. Graphs with high and low values from each group were chosen for validation.

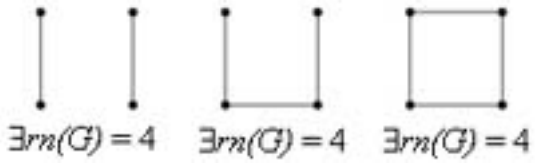


Figure 22: All graphs of order 4 with $\exists rn(G) = 4$

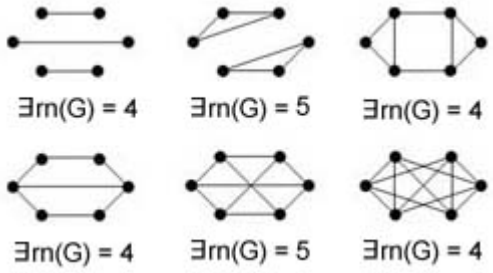


Figure 23: All graphs of order 6 with $\exists rn(G) > 3$

Tables 1-4 list the distributions of the reconstruction numbers produced by my program. As stated in [5] the value of $\exists rn(G)$ for all graphs of order 3, 5, and 7 is 3. In the same paper Harary states that three graphs of order 4 and six graphs of order 6 do not have $\exists rn(G) = 3$. My program verified these results. Figure 22 shows the graphs of order 4 with $\exists rn(G) > 3$. Figure 23 shows the graphs of order 6. Although Harary did not determine the reconstruction numbers of graphs of order greater than 7, it is interesting to note that only 12 graphs of order 8 do not have $\exists rn(G) = 3$. Figure 24 identifies what these graphs are.

As can be seen in figure 24, there are two graphs with $\exists rn(G) = 6$. Earlier a conjecture by Harary and Plantholt, [11, 12] about the upper bound of $\exists rn(G)$ was referenced. This conjecture stated that the upper bound for $\exists rn(G)$ is $\frac{n}{2} + 1$. The two graphs of order 8 with $\exists rn(G) = 6$ provide two counterexamples to this conjecture.

As predicted by Bollobàs, the dominant value of $\exists rn(G)$ is 3. What is surprising is that the dominant value of $\forall rn(G)$ is so close to 3. In fact it seems to be 4. What this means is that if a person were to choose a subdeck of size greater than 4 at random to attempt a reconstruction, the chance that

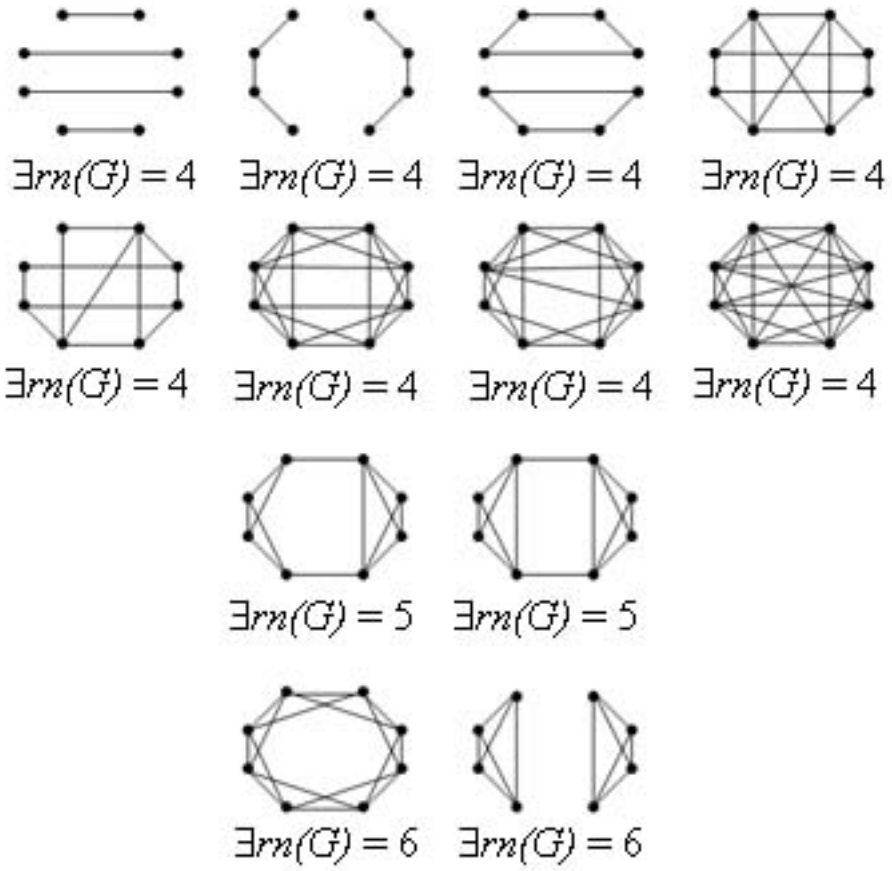


Figure 24: All graphs of order 8 with $\exists rn(G) > 3$

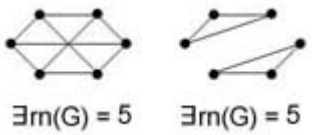


Figure 25: All graphs of order 6 with $\exists rn(G) = \forall rn(G) = 5$

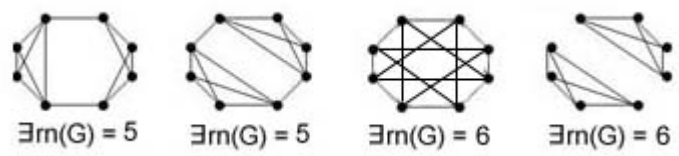


Figure 26: All graphs of order 8 with $\forall rn(G) = 5$ or 6

the chosen subdeck will reconstruct G is very high. This helps validate the opinion that most graphs are easy to reconstruct. Figures 25 and 26 identify graphs with $\forall rn(G) > 4$ and also gives their corresponding $\exists rn(G)$ value.

$\exists rn$	n = 3	n = 4	n = 5	n = 6	n = 7	n = 8	%Totals
3	4	8	34	150	1044	12334	98.74
4		3		4		8	0.11
5				2		2	0.03
6						2	0.02
7							
8							
9							

Table 1: $\exists rn(G)$ counts

$\forall rn$	n = 3	n = 4	n = 5	n = 6	n = 7	n = 8	%Totals
3	4	2	7	8	16	266	2.23
4		9	19	56	496	8209	64.65
5			8	90	520	3576	30.85
6				2	12	292	2.25
7						3	0.02
8							
9							

Table 2: $\forall rn(G)$ counts

$\forall rn$	n = 3	n = 4	n = 5	n = 6	n = 7	n = 8
3	100	18.18	20.59	5.13	1.53	2.15
4		81.82	55.88	35.90	47.51	66.49
5			23.53	57.69	49.81	28.96
6				1.28	1.15	2.37
7						0.03
8						
9						

Table 3: Percentages of $\forall rn(G)$ as a function of n

$\exists rn$	$\forall rn$	n = 3	n = 4	n = 5	n = 6	n = 7	n = 8	%Totals
3	3	4	2	7	8	16	266	2.22
	4		6	19	54	496	8205	64.57
	5			8	86	520	3574	30.80
	6				2	12	286	2.21
	7						3	0.02
	8							
	9							
4	4		3		2		4	0.07
	5				2		2	0.07
	6						2	0.01
	7							
	8							
	9							
5	5				2			0.01
	6						2	0.01
	7							
	8							
	9							
6	6						2	0.01
	7							
	8							
	9							
7	7							
	8							
	9							
8	8							
	9							
9	9							

Table 4: Counts of $\exists rn(G)$ and $\forall rn(G)$ pairs

Table 5 shows the distribution of reconstruction numbers among a few classes of graphs. The distributions for bipartite graphs and regular graphs further validate previously proposed ideas. The distributions for dense graphs, graphs with the number of edges close to the maximum possible number of edges, and sparse graphs, graphs with the number of edges much less than the possible number of edges, illustrates the theorem that $\exists rn(G) = \exists rn(\overline{G})$ [3, 5].

Although not all graphs of order 9 were analyzed, the information that has been produced is interesting. At this time all graphs have $\exists rn(G)$ values of 3. This data represents only about 10% of all graphs of order 9, but it is interesting to note that so far all graphs of odd order have $\exists rn(G)$ values of 3. It will be interesting to see whether or not this pattern continues for the rest of the graphs of order 9. Table 6 illustrates the $(\exists rn(G), \forall rn(G))$ pair values of the data collected so far as well as the current percentages.

Graphs of order 3-7 were very quick to compute the reconstruction numbers for. As a result the reconstruction numbers for all graphs of a single order were determined in one run of my program. The number of graphs of order 8 and 9 made doing something like this a timely task. Instead the number of graphs processed was broken up into many files. For graphs of order 8 50 graphs were processed per run. For graphs of order 9 a single graph was processed per run. To help run all of these processes Condor was used [4]. Condor is an execution program that processes as many jobs as possible in a given time period. The program is installed on 153 computers at the school so I was able to have 153 processes running in parallel. The overall time it took to gather all information about all graphs of a certain order was greatly reduced. The one hindrance was that there was no dedicated resource available for my project. As a result my Condor processes were sometimes evicted making the time until which results were received longer. Overall Condor was a very useful resource. Without the use of Condor graphs of size 8 and 9 would have had to have been run consecutively and the data would not have been as readily available. The use of Condor allowed the data to be computed more quickly.

5 Future Work

There are two major areas of this project that I would like to improve. The first thing I would like to do is reduce the amount of time spent performing IO

	$\exists rn(G)$	$\forall rn(G)$	Bipartite	Regular	Dense	Sparse
n = 4	3	3	4	1	1	1
		4	3	0	1	1
	Total # of Graphs		7	1	2	2
n = 5	3	3	3	3	1	1
		4	6	5	2	2
		5	4	4	1	1
	Total # of Graphs		13	12	4	4
n = 6	3	3	4	2	1	1
		4	6	4	2	2
		5	22	8	5	5
	4	4	2	0	1	1
	Total # of Graphs		35	15	9	9
n = 7	3	3	11	2	1	1
		4	18	6	1	1
		5	57	17	7	7
		6	2	0	0	0
	Total # of Graphs		88	25	9	9
n = 8	3	3	6	3	1	1
		4	71	6	0	0
		5	170	27	5	5
		6	52	4	2	2
	4	4	1	0	2	1
		6	1	0	0	0
	6	6	1	0	0	0
	Total # of Graphs		303	41	9	9

Table 5: Count of reconstruction numbers within classes of graphs

$\exists rn(G)$	$\forall rn(G)$	n = 9	% Total
	3	1897	7.17%
	4	19821	74.95%
3	5	4442	16.80%
	6	282	1.07%
	7	4	0.01%

Table 6: Count of $(\exists rn(G), \forall rn(G))$ Pairs for 26,446 out of 274,668 Graphs of Order 9

functions. If an extension input file is given which includes the extensions of all graphs of order 8, the file will be very large. The way the program is coded, this file will be read each time a new graph is processed. For example, if the reconstruction numbers of 9 graphs are being processed, this large extension file will be read 9 times. This is very cumbersome. In the future I would like to reduce the dependence of this program on file IO. The other area of the project that I would like to improve is the internal storage of a graph. At the present time adjacency matrices are being used. Although they are quite easy to understand, they require a lot of space in memory to be stored. One possible replacement is the structure used within the *nauty* program. The structure represents an adjacency matrix but the data is stored is stored in a variable of type long. Masks are used to manipulate the correct area of data.

I would also like to be able to finish the analysis of graphs of order 9 as well as look at graphs of larger order. Given that of the data collected so far for order 9, the $\exists rn(G)$ value is always 3 it will be interesting to determine for which graph order the value will at some point be greater than 3.

References

[1] L. Babai. *Automorphism groups, isomorphism, reconstruction (Chapter 27 of the Handbook of Combinatorics)*. Elsevier, 1995.

[2] B. Bollobás. Almost every graph has reconstruction number three. *Journal of Graph Theory*, 14(1):1–4, 1990.

[3] J. A. Bondy. A graph reconstructor's manual. *Surveys in Combinatorics*, 1991.

[4] Condor Team, University of Wisconsin-Madison. *Condor Version 6.6.5 Manual*, May 2004.

[5] F. Harary. The graph reconstruction number. *Journal of Graph Theory*, 9:451–454, 1985.

[6] E. Hemaspaandra, L. Hemaspaandra, S. Radziszowski, and R. Tripathi. Complexity results in graph reconstruction. In *Proceedings of the 29th International Symposium on Mathematical Foundations of Computer Science*. to appear.

[7] P. J. Kelly. *On Isometric Transformations*. PhD thesis, University of Wisconsin, 1942.

[8] J. Lauri and R. Scapellato. *Topics in Graph Automorphisms and Reconstruction*. Cambridge University Press, 2003.

[9] B. McKay. Small graphs are reconstructible. *Australasian Journal of Combinatorics*, pages 123–126, 1997.

[10] B. D. McKay. nauty *User's Guide (Version 2.2)*. Computer Science Department, Australian National University, bdm@cs.anu.edu.au. Webpage: http://cs.anu.edu.au/~bdm/nauty/.

[11] R. Molina. Correction of a proof on the ally-reconstruction number of a disconnected graph. *Ars Combinatoria*, pages 59–64, 1995.

[12] W. Myrvold. The ally-reconstruction number of a disconnected graph. *Ars Combinatoria*, pages 123–127, 1989.

[13] C. S. J. A. Nash-Williams. The reconstruction problem. *Selected Topics in Graph Theory*, pages 205–236, 1978.

[14] S. M. Ulam. *A collection of Mathematical Problems*. New York : Interscience Publishers, 1960.