

# The Clique Problem

A Polynomial Time and Non-Heuristic Solution

(P=NP)

John Archie Gillis

September 7, 2018

# Table of contents

- 1 Introduction to Clique
- 2 Description of the New Approach
- 3 Finding the largest clique (Maximum Clique)
- 4 Listing all maximal cliques (cliques that cannot be enlarged)
- 5 Solving the decision problem of testing whether a graph contains a clique larger than a given size.
- 6 Finding cliques of a selected size, particularly largest cliques.
- 7 Clique Finding Functions

## Appendix

- A – The Traveling Salesman Reduction
- B – The Sudoku Reduction
- C – Further Background
- D - About the Author
- E – References

# 1 Introduction to Clique

In computer science, the clique problem is the computational problem of finding cliques (subsets of vertices, all adjacent to each other, also called complete subgraphs) in a graph. It has several different formulations depending on which cliques, and what information about the cliques, should be found. Common formulations of the clique problem include finding a maximum clique (a clique with the largest possible number of vertices), finding a maximum weight clique in a weighted graph, listing all maximal cliques (cliques that cannot be enlarged), and solving the decision problem of testing whether a graph contains a clique larger than a given size.

The clique problem arises in the following real-world setting. Consider a social network, where the graph's vertices represent people, and the graph's edges represent mutual acquaintance. Then a clique represents a subset of people who all know each other, and algorithms for finding cliques can be used to discover these groups of mutual friends. Along with its applications in social networks, the clique problem also has many applications in bioinformatics and computational chemistry.

Most versions of the clique problem are hard. The clique decision problem is NP-complete (one of Karp's 21 NP-complete problems). The problem of finding the maximum clique is stated to be both fixed-parameter intractable and hard to approximate. Most experts in the field have concluded that listing all maximal cliques may require exponential time as there exist graphs with exponentially many maximal cliques.

Most, if not all experts in the field agree that to find a maximum clique, one can systematically inspect all subsets, but this sort of brute-force search is too time-consuming to be practical for networks comprising more than a few dozen vertices. Although (prior to the present paper), no polynomial time algorithm is known for this problem, more efficient algorithms than the brute-force search are known. For instance, the Bron–Kerbosch algorithm can be used to list all maximal cliques in worst-case optimal time, and it is also possible to list them in polynomial time per clique.

While a method for computing the solutions to NP-complete problems using a reasonable amount of time has remained undiscovered (until the publication of the present paper), computer scientists and programmers (whom have not studied the present paper), still frequently encounter NP-

complete problems. NP-complete problems are often addressed by using heuristic methods and approximation algorithms. This is a very poor methodology and strategy. In many instances, it could take decades to make any real progress with these methods, especially when the polynomial solution that is presented in this paper will have enormous impacts on human health, such as assisting with discoveries that could aid in treatments for Alzheimer's, Cancer and many other illnesses. We will make enormous headway with determining how a protein's amino acid sequence dictates its three-dimensional atomic structure. Unfortunately, buy in from the "experts" in this field will be required before any of these systems stand a chance of being developed, and thus anybody hoping to benefit from these more sophisticated therapies, will need to wait until the academic community can find the time in their busy schedules to read the present paper.

The ability to compute solutions for problems such as clique (and many others) has been deemed the holy grail of computational complexity theory and is one of the Millennium Prize Problems. The Millennium Prize Problems are seven problems in mathematics that were stated by the Clay Mathematics Institute in 2000 for which a solution to any individual problem would award the solver \$1,000,000.

## **2 Description of the New Approach**

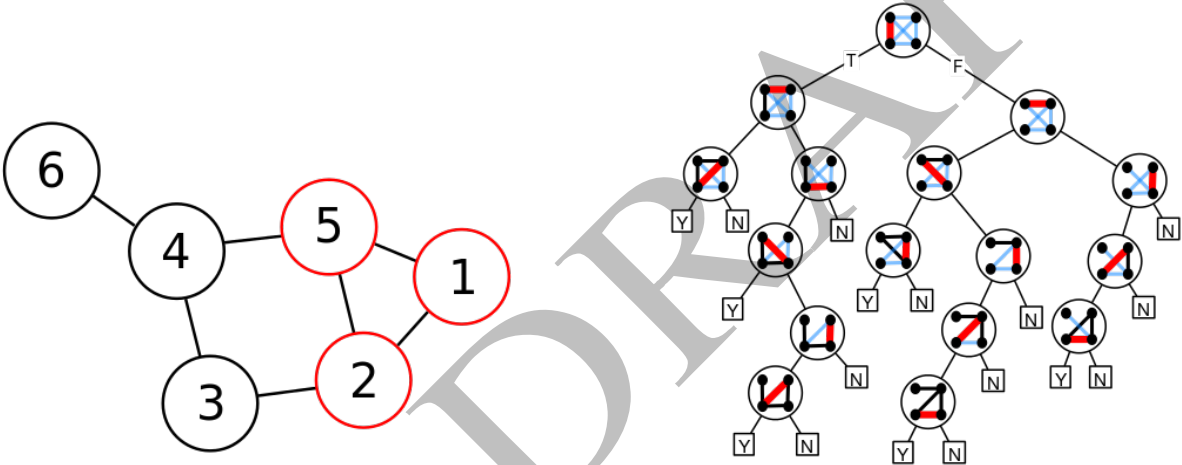
The present paper provides a novel approach to solving the clique problem(s). The author provides a means for greatly reducing the time that it will take a computer (or human) to solve for:

1. Maximum clique (a clique with the largest possible number of vertices),
2. Listing all maximal cliques (cliques that cannot be enlarged), and
3. Solving the decision problem of testing whether a graph contains a clique larger than a given size.
4. Finding cliques of a selected size, particularly largest cliques.

To solve the clique problem, we must discard previous graphing methods completely and start from scratch with a new strategy. We already know that traditional methods have not worked. Many have tried for the last 40 years with no success. Many of these individuals were/are very intelligent. The people are not the problem, the systems are.

Traditional data input methods simply will not work for sorting data into cliques. The present novel method requires that the data and each of its variables be completely converted into a binary system so that each and every permutation can be accounted for and compared to each other in a perfectly and elegantly logical way. In our new method each variable will take on both a single binary place value, such as 1, 2, 4, 8, 16, 32, 64 etc. and also a numerical value that shows the single variables relationship to all other variables in the data set.

By organizing our data in this way we can account for EVERY possible permutation that might occur, no matter what the size of the input. This being stated, the present methods run much deeper than being a simple conversion to a Boolean Incidence Matrix.



Present methods of working with graphs as seen above, are one of the problems why finding items, such as a largest clique, or clique(s) of a fixed size is a real problem for computational devices. Attempting to solve a clique problem of any significant size by utilizing these methods seems a silly (often impossible) endeavor.

I will use a social network to explain how the present system provides a better method for communicating with computational devices for the purpose of solving this difficult problem. A system must be employed that will allow a computer to sort and search for a requested output. Some algorithms for certain questions and inputs (particularly NP-Class problems) can in many instances (with many variables) take longer than the age of the universe to run. This is not acceptable.

Mathematicians generally order elements of clique in an ordered progression 1, 2, 3, 4, 5, 6, etc., but herein lies the problem when trying to find specific size cliques or groups within the larger structure. The present invention matches a binary place valued number to the variables, which in this example are names prior to computation.

- As an example;
- John becomes 1
- Sue becomes 2
- Bob becomes 4
- Jenn becomes 8
- Colin becomes 16
- Maggy becomes 32
- Jim becomes 64
- Kelly becomes 128

TABLE 1

CLIQUE		Kelly	Jim	Maggy	Colin	Jenn	Bob	Sue	John	
		128	64	32	16	8	4	2	1	
1	John	1	1	1	1	1	1	1	1	<b>255</b>
2	Sue	1	1	1	1	1	1	1	1	<b>255</b>
4	Bob	0	0	1	1	1	1	1	1	<b>63</b>
8	Jenn	0	0	1	1	1	1	1	1	<b>63</b>
16	Colin	0	0	0	1	1	1	1	1	<b>31</b>
32	Maggy	0	0	1	0	1	1	1	1	<b>47</b>
64	Jim	0	1	0	0	0	0	1	1	<b>67</b>
128	Kelly	1	0	0	0	0	0	1	1	<b>131</b>
		<b>131</b>	<b>67</b>	<b>47</b>	<b>31</b>	<b>63</b>	<b>63</b>	<b>255</b>	<b>255</b>	

The above table provides an example for only one instance of clique. It is a very simple version of clique with a very small input of only eight individuals. Most instances will have many more variables to be sorted, but for simplicity we will use this very basic example.

In the provided example of Table 1; John has been assigned the binary variable 1 and Sue 2. John and Sue are then further assigned the number 255. 255 shows their relationship to themselves, each other and everybody else in the data set.

By following this logic, we can also see that IF an individual is assigned a number 254, 253, 251, 247, 239, 223, 191 or 127, then they will be friends with six people, themselves and enemies with one. We can see this clearly in the binary representations. 11111110, 1111101, 11111011, 11110111, 11101111, 11011111, 10111111, 01111111. These binary numbers represent both themselves and all their friendships within the data set. A high or low number doesn't mean that you have more or less friends, but it instead tells us exactly who your friends are. This will be useful when we begin sorting and filtering for requested outputs.

Although it is a bit confusing, we must sort the binary numbers into nine specific groupings or levels for the system to work. The groups are:

1. Binary numbers with eight 1's and zero 0's or Level 8 of 8.
2. Binary numbers with seven 1's and one 0 or Level 7 of 8.
3. Binary numbers with six 1's and two 0's or Level 6 of 8.
4. Binary numbers with five 1's and three 0's or Level 5 of 8.
5. Binary numbers with four 1's and four 0's or Level 4 of 8.
6. Binary numbers with three 1's and five 0's or Level 3 of 8.
7. Binary numbers with two 1's and six 0's or Level 2 of 8.
8. Binary numbers with one 1's and seven 0's or Level 1 of 8.
9. Binary numbers with zero 1's and eight 0's or Level 0 of 8.

TABLE 2

8 on/0 off	7 on/1 off	6 on/2 off		5 on/3 off			4 on/4 off			
255	127	63	237	31	124	211	15	85	149	204
	191	95	238	47	143	213	23	86	150	209
	223	111	243	55	151	214	27	89	153	210
	239	119	245	59	155	217	29	90	154	212
	247	123	246	61	157	218	30	92	156	216
	251	125	249	62	158	220	39	99	163	225
	253	126	250	79	167	227	43	101	165	226
	254	159	252	87	171	229	45	102	166	228
		175		91	173	230	46	105	169	232
		183		93	174	233	51	106	170	240
		187		94	179	234	53	108	172	
		189		103	181	236	54	113	177	
		190		107	182	241	57	114	178	
		207		109	185	242	58	116	180	
		215		110	186	244	60	120	184	
		219		115	188	248	71	135	195	
		221		117	199		75	139	197	
		222		118	203		77	141	198	
		231		121	205		78	141	201	
		235		122	206		83	147	202	

3 on/5 off			2 on/6 off		1 on/7 off	0 on/8 off
7	67	140	3	96	1	0
11	69	145	5	129	2	
13	70	146	6	130	4	
14	73	148	9	132	8	
19	74	152	10	136	16	
21	76	161	12	144	32	
22	81	162	17	160	64	
25	82	164	18	192	128	
26	84	168	20			
28	88	176	24			
35	97	193	33			
37	98	194	34			
38	100	196	36			
41	104	200	40			
42	112	208	48			
44	131	224	65			
49	133		66			
50	134		68			
52	137		72			
56	138		80			





For simplicity, we will break the binary system into what we will call “**Levels # of #**”.

As per the TABLE 2 and 3, the **first** column with the number 255 will be classified as a **Level 8 of 8** number. The 8 simply represents the number of “1’s or ON’s” that the output number will contain, when represented in its binary form. We could also call it an “8 ON/0 OFF” number. This means that any individual with this number will belong to every clique, be friends with everybody and if they share this number with another they will all have the same exact number of friends, be friends with themselves and each other. It also means that they will belong to any grouping of smaller cliques as well. If we are to look for groupings or cliques of three for example, anyone with the number 255 will need to be automatically included.

The **second** column will be classified as **Level 7 of 8** numbers. There are 8 of these numbers (127, 191, 223, 239, 247, 251, 253, 254). This means that anybody with one of these numbers is friends with six people, themselves (one) and non-friends or enemies with one. Each specific number actually tells us exactly whom their friends and enemies are.

The **third** column will be classified as Level 6 of 8 numbers. There are 28 of them (63-252).

The **fourth** column will be classified as Level 5 of 8 numbers. There are 56 of them (31-248).

The **fifth** column will be classified as Level 4 of 8 numbers. There are 70 of them (15-240).

The **sixth** column will be classified as Level 3 of 8 numbers. There are 56 of them (7-224).

The **seventh** column will be classified as Level 2 of 8 numbers. There are 28 of them (3-192).

The **eighth** column will be classified as Level 1 of 8 numbers. There are 8 of them (8-128).

The **ninth** column will be classified as a Level 0 of 8 number. There is 1 of them (0).

Searching out various cliques gets more and more complicated as we get closer and closer to a 50/50% distribution. For example, in a data set of eight people, if everyone were to have 4 different friends and four different enemies, then we would be at our maximum complexity with 70 numbers at the level 4 of 4. Level 5 of 8, 6 of 8, 7 of 8 and 8 of 8 might be complicated, but we can alternatively look for enemies rather than friends if we need to, which may simplify these levels. As an example; 3 or 0000 0011 will be a sub-level number to any binary digit with two ones at the end 0000 00**11**. Some examples are 7 (0000 0**111**), 67 (0100 00**11**), 15 (0000 1**111**), 127 (0111 1**111**), 256 (1111 1**111**) and many more!

An individual that has four friends occur (3 friends + him/herself) would be in a LEVEL 4 of 8. Any LEVEL 8 of 8 individuals would obviously be friends with the exact same four people, because logically LEVEL 8 of 8 individuals are friends with everybody. Again this would hold true for the other levels as well, but becomes slightly more complex.

The provided grouping examples will be the same for any eight variable clique, but if the data set for input is larger or smaller than 8 people we obviously need to adjust the numbers to fit the sizes to be input. If ten people are to be sorted into various groups for example, we will use numbers ranging from 0-512. For example; 1, 2, 4, 8, 16, 32, 64, 128, 256, and 512. Each individual is assigned one of these binary numbers and in the case of a larger data set binary numbers with nine 1's, ten 1's, etc., will be required.

Additional numbers will obviously need to be added for organizing larger data sets. No matter what size the data set, we will want to break up the data into groupings of binary numbers with equal numbers of 1's (or use a function to find them). Thus we may have groupings called Level 10 of 10 (11 1111 1111) or Level 8 of 10 (00 1111 1111), (11 1111 0101), (11 10011 1111) etc. We might even have massive data sets and levels such as Level 82 of 1000 or Level 762 of 2,300,000,000 (number of Facebook users), etc.

We then create a grid which assigns a 1 for friendships and a 0 for enemies/non-friends. We can then create a binary representation of each individual's relationship to every other individual, which provides us with all the possible friend permutations, both large and small. Essentially an additional number is assigned to each person that includes their position in the group! We can then use these numbers to sort our data into whatever sized clique(s) that we desire. It also becomes easy to find the largest clique, by simply sorting numbers rather than resorting to a brute force methodology.

### **3 Finding the largest clique (Maximum Clique)**

We will order the following questions into a computer algorithm:

1. **What is the largest number in the Level 8 of 8 number set?**

In the provided example our largest number is 255.

Note:

(It is the only number in that set)

**2. Is there more than one identical largest number output?**

In the provided example this answer is Yes.

**3. How many identical largest numbers are there?**

In the provided example there are two outputs of 255.

For this to have been our largest clique, we would have needed eight 255's. We only have two and thus this cannot be the largest clique. We did discover that John and Sue had the most friends in common, but that was not what we were looking to find.

**4. Does one or more of the output numbers belong to the "7 ON/1 OFF" (which is the same as the Level 7 of 8) category?** In the present example the answer is no. (If even just one of these numbers were to be an output and six 255, numbers, we would have found a largest clique of seven. This is NOT the case in this example). You can start to see however how our system works.

**5. Does one or more of the output numbers belong to the "6 ON/2 OFF" (which is the same as the Level 6 of 8) category?** We can see that we have an output of two of these numbers. 63 (C)-Bob and 63 (D)-Jenn fall into this category.

**6. Are there six of these numbers OR in the present scenario are there two of these numbers (i.e. 63) and four "8 ON/0 OFF" numbers?** If there were, we would have a clique of six. This is not the case. There are only two numbers in the "8 ON/0 OFF" classification, thus the largest clique that we would be able to create would be of four people (63, 63, 255 & 255) or John, Sue, Bob & Jenn. We must thus continue the search, but we can now see that our largest clique will consist of at least four people and thus there will be no need to search any lower numbered cliques. (Four or five will be the answer in the present scenario).

(Note: Even if there were NO “6 on/2 off” numbers, but there were 6 numbers of 255 we would still have a max clique of six, even though there are NO individuals that perfectly fit into a clique of six. It will likely be very rare to find a person wherein they and all their friends fit perfectly into a clique, wherein they have no friends that are excluded).

7. **Does one or more of the output numbers belong to the “5 ON/3 OFF” (which is the same as the Level 5 of 8) category?**

Yes, 31 (E) Colin (00011111) and 47 (F) Maggy (00101111) both belong to this category.

8. **Are there five of these numbers?** No

9. **Are there one or more of these numbers that share at least five of their binary 1's or (ON's) with another output number?** Yes! In the present scenario wherein there is only one of each number in the “5 on/3 off” category there are four other numbers in the categories of numbers with 6 or more 1's that share at least the same placement of 1's as 31 and 47. In our present scenario we see that both 63 (00111111) and 255 (11111111) both share at least the same number of 1's (or ON's) as 63 and 255. Because there are two 63's and two 255's we have our required 5 numbers for a five clique! This ALSO means that there is not just one largest clique, but that we have a tie for the largest clique. We have TWO!

#### **4 Listing all maximal cliques (cliques that cannot be enlarged)**

1. **Find the maximal clique(s).** (See above)
2. **Ask does one or more of the output numbers belong to the “4 ON/4 OFF” category?**  
In the present example the answer is no and so this tells us that there is no maximal clique of four. Essentially any four clique can be expanded to a five clique.
3. **Ask does one or more of the output numbers belong to the “3 ON/5 OFF” category?**  
In the present example the answer is yes.
4. **What are they?**

In the present example we have two numbers in the “3 ON/5 OFF” category.

They are (67) Jim and (131) Kelly. WAIT! We need at least three numbers to make a three clique!

5. **Do any of the output “3 ON/5 OFF” category numbers share the same three “ON’s or 1’s” as any of the other output numbers.**

In the present example they do!

We have two cliques that do not fall into sub groups of our larger four or five cliques

1. 131 (Kelly), 255 (Sue) and 255 (John)
2. 67 (Jim), 255 (Sue) and 255 (John)

Note:

If we wanted discover all of the three cliques that are built within the other output numbers, we would have ten that include Maggy and ten that include Colin. Four would cancel out as repeated and so we would have sixteen three-person cliques and the two non-expandable maximal cliques, for a total of 18 three-person cliques.

$$\begin{aligned}
 C(n, r) &= C(5, 3) \\
 &= \frac{5!}{(3!(5-3)!)} \\
 &= 10
 \end{aligned}$$

8. 47 Maggy, 63 Jenn, and 63 Bob,
  9. 47 Maggy, 63 Jenn, and 255 Sue,
  10. 47 Maggy, 63 Bob and 255 Sue,
  11. 63 Jenn, 63 Bob and 255 Sue,
  12. 47 Maggy, 63 Jenn and 255 John
  13. 47 Maggy, 63 Bob, and 255 John
  14. 63 Jenn, 63 Bob and 255 John
  15. 47 Maggy, 255 Sue and 255 John
  16. 63 Jenn, 255 Sue and 255 John
  17. 63 Bob, 255 Sue and 255 John
- 
1. 31 Colin, 63 Jenn and 63 Bob
  2. 31 Colin, 63 Jenn and 255 Sue
  3. 31 Colin, 63 Bob and 255 Sue
  4. 31 Colin, 63 Jenn and 255 John
  5. 31 Colin, 63 Bob, and 255 John
  6. 31 Colin, 255 Sue and 255 John
  7. ~~63 Jenn, 255 Sue and 255 John~~

- ~~8. 63 Bob, 255 Sue and 255 John~~
- ~~9. 63 Jenn, 63 Bob and 255 Sue~~
- ~~10. 63 Jenn, 63 Bob and 255 John~~

There are a number of easy ways to achieve our output, but for simplicity I simply found all the three combinations of the higher categorical groupings and deleted those that were duplicate. Thus we have 16 higher categorical groupings and two lower categorical groupings for a complete list of 18 total three cliques.

Here are the 18:

1. 47 Maggy, 63 Jenn, and 63 Bob
2. 47 Maggy, 63 Jenn, and 255 Sue
3. 47 Maggy, 63 Bob and 255 Sue
4. 63 Jenn, 63 Bob and 255 Sue
5. 47 Maggy, 63 Jenn and 255 John
6. 47 Maggy, 63 Bob, and 255 John
7. 63 Jenn, 63 Bob and 255 John
8. 47 Maggy, 255 Sue and 255 John
9. 63 Jenn, 255 Sue and 255 John
10. 63 Bob, 255 Sue and 255 John
11. 31 Colin, 63 Jenn and 63 Bob
12. 31 Colin, 63 Jenn and 255 Sue
13. 31 Colin, 63 Bob and 255 Sue
14. 31 Colin, 63 Jenn and 255 John
15. 31 Colin, 63 Bob, and 255 John
16. 31 Colin, 255 Sue and 255 John
17. 131 Kelly, 255 Sue and 255 John (**Maximal**)
18. 67 Jim, 255 Sue and 255 John (**Maximal**)

5. **Ask does one or more of the output numbers belong to the “2 ON/7 OFF” category?**

In the present example the answer is no and so we know that there are no maximal two-person cliques.

6. **Ask does one of the output numbers belong to the “1 ON/8 OFF” category?**

In the present example the answer is no and so we know that there are no maximal one-person cliques. A one-clique, would be a person with no friends, would just be friends with themselves and thus cannot really be called a clique.

## 5 Solving the decision problem of testing whether a graph contains a clique larger than a given size.

We will use the example question of: Is there a clique larger than four?

1. **Ask does one or more of the output numbers belong to the “5 ON/3 OFF” category?**

In the present example the answer is yes. 31 Colin and 47 Maggy belong to this category.

2. **Do any of the output “5 ON/3 OFF” category numbers share the same three “ON’s or 1’s” as any of the other output numbers.**

As Colin is 00011111- 31 we would need to search for only seven possible numbers

They would consist of:

- 10011111- 159
- 01011111- 95
- **00111111- 63 (we find two of these)**
- 11011111- 223
- 10111111- 191
- 01111111- 127
- **11111111- 255 (we find two of these)**

As Maggy is 00101111- 47 we would need to search for only seven possible numbers

They would consist of:

- 10101111- 95
- 01101111- 111
- **00111111- 63 (we find two of these)**
- 11101111- 239
- 10111111- 191
- 01111111- 127
- **11111111- 255 (we find two of these)**

3. **Ask, when we add the numbers that fall into these categories, are there five or more of them?**



In the present example the answer is Yes. This means that the two numbers that were output in the “**5 ON/3 OFF**” category each belong clique of five.

## 6 Finding cliques of a selected size, particularly largest cliques.

If we were to request outputs from a data set of 50 people, we would have 562,949,953,421,312 different possible numbers to represent each individual’s relationship to themselves and everybody else in the data set. If these 50 people all lived in the same town for example, the chances of us finding a clique of 30 might be pretty good. If we wanted to check if there was a 30 clique, we would need to do the following. Let’s also pretend that everybody has at least 30 friends or more.

1. **Ask does one or more of the output numbers belong to the Level 30 of 50 numbers category?**

If the number 1,073,741,823 or (00 0000 0000 0000 0000 0011 1111 1111 1111 1111 1111 1111 1111 1111) in binary showed up for example, we would have a yes. Let’s pretend that there were only one Level 30 of 50 numbers discovered.

2. **We would then need to ask; Do any of the other output numbers share the same 30 binary “ON or 1” values as 1,073,741,823?**

Let’s pretend the answer is yes. An example is 2,147,483,647.

3. **Are there at least 29 of these other numbers?**

In this example we would only need to compare our 29 numbers to a list of 524,288 potential possibilities. This is much more practical than trying all 50! (or  $3.0414093e+64$ ) combinations. Let’s pretend the answer is YES. We have found our clique of 30.

4. **WAIT! But are there other cliques of 30?**

There very well could be other cliques of 30, but not ones that include Mr. 1,073,741,823 (Let’s call him Jake). He only had 30 friends and so could ONLY belong to this specific 30 clique. No other 30 clique could possibly include Jake.

Note:

If we wanted to discover ALL of the 30 cliques, we would use a different method that would include looking for 30 cliques within our Level 31 of 50, Level 32 of 50, Level 33 of 50 categories etc. Alternatively, we could work back from 50 down to 30.

## 7 Clique Finding Functions (Finding cliques of a selected size)

To take a slightly different approach than referring to a list, we can create specific computational functions. There are many possibilities, but to provide a simple example, (again with our single Level 30 of 50 number in the above example of 1,073,741,823), if the binary digit furthest right is “ON or 1” (such as it is in 1,073,741,823, because it is obviously an odd number, then all even numbers can be discounted and do not need to be considered when trying to determine if they are part of this specific clique. We know that only numbers that are odd will belong to our clique. This immediately cuts our search effort in half. So in summary, if any of the 50 output numbers are even, they do not even need to be compared and can be deleted. (If more than 20 were deleted we would deduce that there is no 30 clique.

To provide yet another example of the function system, if the binary place value for 2 is on we can delete another 50% of the complete set (they will be overlap from the odd number deletion however).

The function would look like:

IF (from the output list) the **second** digit (from the right in the base-10 number) is zero or an even number AND the first digit is 2, 3, 6 or 7 OR IF the second digit is odd AND the first digit is 0, 1, 4, 5, 8, 9 THEN do not delete these numbers from the 50 numbers in our list (looking for a 30 clique in the group of 30). If a number does not fit one of these requirements then DELETE it, it is not part of the clique. (i.e. 72 and 73 would be deleted. Thus, we can see how rules can start to be created to narrow our 50 number list down, without the requirement of comparing it to all of the remaining possible outputs from a list. (524,288 in our 30 of 50 example).

TABLE 4

<b>2</b>	10	<b>22</b>	10110	<b>42</b>	101010	<b>62</b>	111110
<b>3</b>	11	<b>23</b>	10111	<b>43</b>	101011	<b>63</b>	111111
<b>6</b>	110	<b>26</b>	11010	<b>46</b>	101110	<b>66</b>	1000010
<b>7</b>	111	<b>27</b>	11011	<b>47</b>	101111	<b>67</b>	1000011
<b>10</b>	1010	<b>30</b>	11110	<b>50</b>	110010	<b>70</b>	1000110
<b>11</b>	1011	<b>31</b>	11111	<b>51</b>	110011	<b>71</b>	1000111
<b>14</b>	1110	<b>34</b>	100010	<b>54</b>	110110	<b>74</b>	1001010
<b>15</b>	1111	<b>35</b>	100011	<b>55</b>	110111	<b>75</b>	1001011
<b>18</b>	10010	<b>38</b>	100110	<b>58</b>	111010	<b>78</b>	1001110
<b>19</b>	10011	<b>39</b>	100111	<b>59</b>	111011	<b>79</b>	1001111
							etc.

Below provides an example if the binary 1 and binary 2 place values were required to be on in unison. The logic might look like this:

IF digit two is odd AND digit one is 1, 5, or 9 OR IF digit two is even AND digit one is 3 or 7 THEN delete all numbers that do not have these attributes. All the numbers below DO have these attributes.

TABLE 5

<b>3</b>	11	<b>67</b>	1000011	<b>131</b>	10000011	<b>195</b>	11000011
<b>7</b>	111	<b>71</b>	1000111	<b>135</b>	10000111	<b>199</b>	11000111
<b>11</b>	1011	<b>75</b>	1001011	<b>139</b>	10001011	<b>203</b>	11001011
<b>15</b>	1111	<b>79</b>	1001111	<b>143</b>	10001111	<b>207</b>	11001111
<b>19</b>	10011	<b>83</b>	1010011	<b>147</b>	10010011	<b>211</b>	11010011
<b>23</b>	10111	<b>87</b>	1010111	<b>151</b>	10010111	<b>215</b>	11010111
<b>27</b>	11011	<b>91</b>	1011011	<b>155</b>	10011011	<b>219</b>	11011011
<b>31</b>	11111	<b>95</b>	1011111	<b>159</b>	10011111	<b>223</b>	11011111
<b>35</b>	100011	<b>99</b>	1100011	<b>163</b>	10100011	<b>227</b>	11100011
<b>39</b>	100111	<b>103</b>	1100111	<b>167</b>	10100111	<b>231</b>	11100111
<b>43</b>	101011	<b>107</b>	1101011	<b>171</b>	10101011	<b>235</b>	11101011
<b>47</b>	101111	<b>111</b>	1101111	<b>175</b>	10101111	<b>239</b>	11101111
<b>51</b>	110011	<b>115</b>	1110011	<b>179</b>	10110011	<b>243</b>	11110011
<b>55</b>	110111	<b>119</b>	1110111	<b>183</b>	10110111	<b>247</b>	11110111
<b>59</b>	111011	<b>123</b>	1111011	<b>187</b>	10111011	<b>251</b>	11111011
<b>63</b>	111111	<b>127</b>	1111111	<b>191</b>	10111111	<b>255</b>	11111111

Going back to our eight-person example, we can see that with our novel method of finding the largest clique we only had to find one or more numbers in the “Level 5 of 8” category, (wherein we found two). After that, all we had to do was to search seven additional numbers of higher categories that share at least the same five binary places as the two “Level 5 of 8” category numbers. With traditional methods of trial and error we would have needed to make at least  $1+8+28+56=93$  comparisons to achieve the same result. The present method for this example is a significant 86.72% better.

In a larger example to find a clique of 8 out of 16 wherein there is only one person of the 16 with exactly only 8 friends but that fits into an 8 clique, we would likely have to make  $1+16+120+560+1820+4368+8008+11440=26,333$  (Level 1 numbers+ Level 2 numbers+ Level 3 numbers+ Level 4 numbers+ Level 5 numbers+ Level 6 numbers+ Level 7 numbers+ Level 8 numbers) at least 26,333 comparisons out of the 65,536 possible to find the correct answer as trial and error would not be required for anybody with less than 8 friends.

Our new method requires that 16 numbers be output to find an 8 clique. If only a single person is found with 8 friends (0000 0000 1111 1111 =255), we will need to compare their output number with the other 15 numbers to determine if there are in fact seven other numbers that (in addition to our 0000 0000 1111 1111 = 255), will create an 8 clique. If there are seven numbers that share the same eight binary ON digits, then we have our clique. By utilizing the present methods there are only 255 numbers that will work and thus we simply have to see if seven of our 15 numbers from the output list match any of the 255 acceptable numbers list. Examples of these numbers include (0000 0001 1111 1111), (0000 0011 1111 1111) or (1000 1111 1111 1111), etc. Comparing these 15 numbers to a list of 255 is much faster and more efficient than comparing the 15 numbers to 26,333 or 65,536 numbers (if everybody were to have 8+ friends). Lightning speeds faster than any trial and error method. It is also a scalable solution.

Again we can have a computer search for these numbers from a chart or turn our search into a computational function.

I will now provide some more in depth examples from the data set of TABLE 1, which consists of 8 people in total. It is small and simple and thus should be easy to visualize, particularly the way that it is set up.

What if we wanted to find if/how many clique(s) of 5 friends exist in this group of 8 people?

We will begin by searching our list for numbers with five ones. 31, 47, 55, 59, 61 and 62 are all examples of binary numbers with five ones. We then look for duplicates. In the below example, there are no duplicates for a person with 4 friends + themselves (5) as a friend.

What we find is that any binary number that shares the same five ones, must also be included in our 5 clique. Thus if a 63, 191 or 255 happen to be in our list (which they are), then they will be part of the 5 clique too! See TABLE 6.

TABLE 6

<b>255</b>	1111 1111	<b>191</b>	1011 1111	<b>63</b>	0011 1111	<b>31</b>	0001 1111
						<b>47</b>	0010 1111
						<b>55</b>	0011 0111
						<b>59</b>	0011 1011
						<b>61</b>	0011 1101
						<b>62</b>	0011 1110

Notice that 47 and 31 are the only binary numbers with five ones in our TABLE 1 example. We can now easily see that they are both sub-groups of 63 (six ones wherein 5 are shared) and 255 (eight ones wherein 5 are shared), which are also found in our TABLE 1 list, thus we have two cliques of five.

Note: Five is also the largest size clique.

- 1) 47 Maggy, 63 Jenn, 63 Bob, 255 Sue, and 255 John, and
  - 2) 31 Colin, 63 Jenn, 63 Bob, 255 Sue, and 255 John
- are both our two largest cliques and consist of 5 friends.



diagram below shows how these numbers all share five 1's. In this simplified scenario they are the five 1's justified furthest to the right.

TABLE 8

<b>255</b>	1111 1111	<b>191</b>	1011 1111	<b>63</b>	0011 1111	<b>31</b>	0001 1111
						<b>47</b>	0010 1111
						<b>55</b>	0011 0111
						<b>59</b>	0011 1011
						<b>61</b>	0011 1101
						<b>62</b>	0011 1110

...review the next 7 pages (TABLE 8 extended) to view the connection between each category level. LEVEL 4 of 8, LEVEL 3 of 8, LEVEL 2 of 8, LEVEL 1 of 8 and LEVEL 0 of 8 are not shown, due to space limitations. Notice how six of every LEVEL 5 of 8 numbers shares five of the same six "1's or ON's" as their corresponding LEVEL 6 of 8 numbers. Our LEVEL 6 of 8 numbers share six of seven "1's or ON's" and so on.

2nd DRAFT

LEVEL 8 of 8		LEVEL 7 of 8		LEVEL 6 of 8		LEVEL 5 of 8				
255	1111 1111	127	0111 1111	63	0011 1111	31	0001	1111		
				47	0010	1111				
				55	0011	0111				
				59	0011	1011				
				61	0011	1101				
				62	0011	1110				
				95	0101 1111	31	0001	1111		
				79	0100	1111				
				87	0101	0111				
				91	0101	1011				
				93	0101	1101				
				94	0101	1110				
				111	0110 1111	47	0010	1111		
				79	0100	1111				
				103	0110	0111				
				107	0110	1011				
				109	0110	1101				
				110	0110	1110				
				119	0111 0111	55	0011	0111		
				87	0101	0111				
				103	0110	0111				
				115	0111	0011				
				117	0111	0101				
				118	0111	0110				
				123	0111 1011	59	0011	1011		
				91	0101	1011				
				107	0110	1011				
				115	0111	0011				
				121	0111	1001				
				122	0111	1010				
				125	0111 1101	61	0011	1101		
				93	0101	1101				
				109	0110	1101				
				117	0111	0101				
				121	0111	1001				
				124	0111	1100				
				126	0111 1110	62	0011	1110		
				94	0101	1110				
				110	0110	1110				
				118	0111	0110				
				122	0111	1010				
				124	0111	1100				
				191	1011 1111	63	0011 1111	31	0001	1111
				47	0010	1111				
				55	0011	0111				
				59	0011	1011				
				61	0011	1101				
				62	0011	1110				



		159	1001 1111	31	0001	1111			
				143	1000	1111			
				151	1001	0111			
				155	1001	1011			
				157	1001	1101			
				158	1001	1110			
				175	1010 1111	47	0010	1111	
						143	1000	1111	
						167	1010	0111	
						171	1010	1011	
						173	1010	1101	
						174	1010	1110	
				183	1011 0111	55	0011	0111	
						151	1001	0111	
						167	1010	0111	
						179	1011	0011	
						181	1011	0101	
						182	1011	0110	
				187	1011 1011	59	0011	1011	
						155	1001	1011	
						171	1010	1011	
						179	1011	0011	
						185	1011	1001	
						186	1011	1010	
				189	1011 1101	61	0011	1101	
						157	1001	1101	
						173	1010	1101	
						181	1011	0101	
						185	1011	1001	
						188	1011	1100	
				190	1011 1110	62	0011	1110	
						158	1001	1110	
						174	1010	1110	
						182	1011	0110	
						186	1011	1010	
						188	1011	1100	
			223	1101 1111	95	0101 1111	31	0001	1111
							79	0100	1111
							87	0101	0111
							91	0101	1011
							93	0101	1101
							94	0101	1110
					159	1001 1111	31	0001	1111
							143	1000	1111
							151	1001	0111
							155	1001	1011
							157	1001	1101
							158	1001	1110

		<b>207</b>	<b>1100 1111</b>	<b>79</b>	<b>0100</b>	<b>1111</b>			
				<b>143</b>	<b>1000</b>	<b>1111</b>			
				<b>199</b>	<b>1100</b>	<b>0111</b>			
				<b>203</b>	<b>1100</b>	<b>1011</b>			
				<b>205</b>	<b>1100</b>	<b>1101</b>			
				<b>206</b>	<b>1100</b>	<b>1110</b>			
				<b>215</b>	<b>1101 0111</b>	<b>87</b>	<b>0101</b>	<b>0111</b>	
						<b>151</b>	<b>1001</b>	<b>0111</b>	
						<b>199</b>	<b>1100</b>	<b>0111</b>	
						<b>211</b>	<b>1101</b>	<b>0011</b>	
						<b>213</b>	<b>1101</b>	<b>0101</b>	
						<b>214</b>	<b>1101</b>	<b>0110</b>	
				<b>219</b>	<b>1101 1011</b>	<b>91</b>	<b>0101</b>	<b>1011</b>	
						<b>155</b>	<b>1001</b>	<b>1011</b>	
						<b>203</b>	<b>1100</b>	<b>1011</b>	
						<b>211</b>	<b>1101</b>	<b>0011</b>	
						<b>217</b>	<b>1101</b>	<b>1001</b>	
						<b>218</b>	<b>1101</b>	<b>1010</b>	
				<b>221</b>	<b>1101 1101</b>	<b>93</b>	<b>0101</b>	<b>1101</b>	
						<b>157</b>	<b>1001</b>	<b>1101</b>	
						<b>205</b>	<b>1100</b>	<b>1101</b>	
						<b>213</b>	<b>1101</b>	<b>0101</b>	
						<b>217</b>	<b>1101</b>	<b>1001</b>	
						<b>220</b>	<b>1101</b>	<b>1100</b>	
				<b>222</b>	<b>1101 1110</b>	<b>94</b>	<b>0101</b>	<b>1110</b>	
						<b>158</b>	<b>1001</b>	<b>1110</b>	
						<b>206</b>	<b>1100</b>	<b>1110</b>	
						<b>214</b>	<b>1101</b>	<b>0110</b>	
						<b>218</b>	<b>1101</b>	<b>1010</b>	
						<b>220</b>	<b>1101</b>	<b>1100</b>	
			<b>239</b>	<b>1110 1111</b>	<b>111</b>	<b>0110 1111</b>	<b>47</b>	<b>0010</b>	<b>1111</b>
							<b>79</b>	<b>0100</b>	<b>1111</b>
							<b>103</b>	<b>0110</b>	<b>0111</b>
							<b>107</b>	<b>0110</b>	<b>1011</b>
							<b>109</b>	<b>0110</b>	<b>1101</b>
							<b>110</b>	<b>0110</b>	<b>1110</b>
					<b>175</b>	<b>1010 1111</b>	<b>47</b>	<b>0010</b>	<b>1111</b>
							<b>143</b>	<b>1000</b>	<b>1111</b>
							<b>167</b>	<b>1010</b>	<b>0111</b>
							<b>171</b>	<b>1010</b>	<b>1011</b>
							<b>173</b>	<b>1010</b>	<b>1101</b>
							<b>174</b>	<b>1010</b>	<b>1110</b>
					<b>207</b>	<b>1100 1111</b>	<b>79</b>	<b>0100</b>	<b>1111</b>
							<b>143</b>	<b>1000</b>	<b>1111</b>
							<b>199</b>	<b>1100</b>	<b>0111</b>
							<b>203</b>	<b>1100</b>	<b>1011</b>
							<b>205</b>	<b>1100</b>	<b>1101</b>
							<b>206</b>	<b>1100</b>	<b>1110</b>
					<b>231</b>	<b>1110 0111</b>	<b>103</b>	<b>0110</b>	<b>0111</b>
							<b>167</b>	<b>1010</b>	<b>0111</b>
					<b>199</b>	<b>1100</b>	<b>0111</b>		
					<b>227</b>	<b>1110</b>	<b>0011</b>		
					<b>229</b>	<b>1110</b>	<b>0101</b>		
					<b>230</b>	<b>1110</b>	<b>0110</b>		

		<b>235</b> 1110 1011	<b>107</b>	0110	1011	
			<b>171</b>	1010	1011	
			<b>203</b>	1100	1011	
			<b>227</b>	1110	0011	
			<b>233</b>	1110	1001	
			<b>234</b>	1110	1010	
			<b>237</b> 1110 1101	<b>109</b>	0110	1101
				<b>173</b>	1010	1101
				<b>205</b>	1100	1101
				<b>229</b>	1110	0101
				<b>233</b>	1110	1001
				<b>236</b>	1110	1100
			<b>238</b> 1110 1110	<b>110</b>	0110	1110
				<b>174</b>	1010	1110
				<b>206</b>	1100	1110
			<b>230</b>	1110	0110	
			<b>234</b>	1110	1010	
			<b>236</b>	1110	1100	
		<b>247</b> 1111 0111	<b>119</b> 0111 0111	<b>55</b>	0011	0111
				<b>87</b>	0101	0111
				<b>103</b>	0110	0111
				<b>115</b>	0111	0011
				<b>117</b>	0111	0101
				<b>118</b>	0111	0110
			<b>183</b> 1011 0111	<b>55</b>	0011	0111
				<b>151</b>	1001	0111
				<b>167</b>	1010	0111
				<b>179</b>	1011	0011
				<b>181</b>	1011	0101
				<b>182</b>	1011	0110
			<b>215</b> 1101 0111	<b>87</b>	0101	0111
				<b>151</b>	1001	0111
				<b>199</b>	1100	0111
				<b>211</b>	1101	0011
				<b>213</b>	1101	0101
				<b>214</b>	1101	0110
		<b>231</b> 1110 0111	<b>103</b>	0110	0111	
			<b>167</b>	1010	0111	
			<b>199</b>	1100	0111	
			<b>227</b>	1110	0011	
			<b>229</b>	1110	0101	
			<b>230</b>	1110	0110	
		<b>243</b> 1111 0011	<b>115</b>	0111	0011	
			<b>179</b>	1011	0011	
			<b>211</b>	1101	0011	
			<b>227</b>	1110	0011	
			<b>241</b>	1111	0001	
			<b>242</b>	1111	0010	

		<b>245</b> 1111 0101	<b>117</b> 0111 0101
			<b>181</b> 1011 0101
			<b>213</b> 1101 0101
			<b>229</b> 1110 0101
			<b>241</b> 1111 0001
			<b>244</b> 1111 0100
		<b>246</b> 1111 0110	<b>118</b> 0111 0110
			<b>182</b> 1011 0110
			<b>214</b> 1101 0110
			<b>230</b> 1110 0110
		<b>242</b> 1111 0010	
		<b>244</b> 1111 0100	
	<b>251</b> 1111 1011	<b>123</b> 0111 1011	
		<b>59</b> 0011 1011	
		<b>91</b> 0101 1011	
		<b>107</b> 0110 1011	
		<b>115</b> 0111 0011	
		<b>121</b> 0111 1001	
		<b>122</b> 0111 1010	
		<b>187</b> 1011 1011	
		<b>59</b> 0011 1011	
		<b>155</b> 1001 1011	
		<b>171</b> 1010 1011	
		<b>179</b> 1011 0011	
		<b>185</b> 1011 1001	
		<b>186</b> 1011 1010	
		<b>219</b> 1101 1011	
		<b>91</b> 0101 1011	
		<b>155</b> 1001 1011	
		<b>203</b> 1100 1011	
		<b>211</b> 1101 0011	
		<b>217</b> 1101 1001	
	<b>218</b> 1101 1010		
	<b>235</b> 1110 1011		
	<b>107</b> 0110 1011		
	<b>171</b> 1010 1011		
	<b>203</b> 1100 1011		
	<b>227</b> 1110 0011		
	<b>233</b> 1110 1001		
	<b>234</b> 1110 1010		
	<b>243</b> 1111 0011		
	<b>115</b> 0111 0011		
	<b>179</b> 1011 0011		
	<b>211</b> 1101 0011		
	<b>227</b> 1110 0011		
	<b>241</b> 1111 0001		
	<b>242</b> 1111 0010		
	<b>249</b> 1111 1001		
	<b>121</b> 0111 1001		
	<b>185</b> 1011 1001		
	<b>217</b> 1101 1001		
	<b>233</b> 1110 1001		
	<b>241</b> 1111 0001		
	<b>248</b> 1111 1000		

		250	1111 1010	122	0111	1010	
				186	1011	1010	
				218	1101	1010	
				234	1110	1010	
				242	1111	0010	
				248	1111	1000	
	253	1111 1101	125	0111 1101	61	0011	1101
					93	0101	1101
					109	0110	1101
					117	0111	0101
					121	0111	1001
					124	0111	1100
			189	1011 1101	61	0011	1101
					157	1001	1101
					173	1010	1101
					181	1011	0101
					185	1011	1001
					188	1011	1100
			221	1101 1101	93	0101	1101
					157	1001	1101
					205	1100	1101
					213	1101	0101
					217	1101	1001
					220	1101	1100
			237	1110 1101	109	0110	1101
					173	1010	1101
					205	1100	1101
					229	1110	0101
					233	1110	1001
					236	1110	1100
			245	1111 0101	117	0111	0101
				181	1011	0101	
				213	1101	0101	
			229	1110	0101		
			241	1111	0001		
			244	1111	0100		
	249	1111 1001	121	0111	1001		
			185	1011	1001		
			217	1101	1001		
			233	1110	1001		
			241	1111	0001		
			248	1111	1000		
	252	1111 1100	124	0111	1100		
			188	1011	1100		
			220	1101	1100		
			236	1110	1100		
			244	1111	0100		
			248	1111	1000		
254	1111 1110	126	0111 1110	62	0011	1110	
				94	0101	1110	
				110	0110	1110	
				118	0111	0110	
				122	0111	1010	
				124	0111	1100	

		<b>190</b> 1011 1110	<b>62</b>	0011	1110
			<b>158</b>	1001	1110
			<b>174</b>	1010	1110
			<b>182</b>	1011	0110
			<b>186</b>	1011	1010
			<b>188</b>	1011	1100
		<b>222</b> 1101 1110	<b>94</b>	0101	1110
			<b>158</b>	1001	1110
			<b>174</b>	1010	1110
			<b>182</b>	1011	0110
	<b>186</b>	1011	1010		
	<b>188</b>	1011	1100		
<b>238</b> 1110 1110	<b>110</b>	0110	1110		
	<b>174</b>	1010	1110		
	<b>206</b>	1100	1110		
	<b>230</b>	1110	0110		
	<b>234</b>	1110	1010		
	<b>236</b>	1110	1100		
<b>246</b> 1111 0110	<b>118</b>	0111	0110		
	<b>182</b>	1011	0110		
	<b>214</b>	1101	0110		
	<b>230</b>	1110	0110		
	<b>242</b>	1111	0010		
	<b>244</b>	1111	0100		
<b>250</b> 1111 1010	<b>122</b>	0111	1010		
	<b>186</b>	1011	1010		
	<b>218</b>	1101	1010		
	<b>234</b>	1110	1010		
	<b>242</b>	1111	0010		
	<b>248</b>	1111	1000		
<b>252</b> 1111 1100	<b>124</b>	0111	1100		
	<b>188</b>	1011	1100		
	<b>220</b>	1101	1100		
	<b>236</b>	1110	1100		
	<b>244</b>	1111	0100		
	<b>248</b>	1111	1000		

Table 8 shows how our data system will work. For example, if one or more higher level numbers are found in a request/search, but there is also a lower level number that shares ALL the same 1's of the higher level then we will also want the computer system to include the lower level number(s) in the clique. There are a number of ways to approach this, but we demonstrate a small portion of an example reference table in Table 8 extended. Table 8 extended also provides an example showing how 31, 47, 55, 59, 61 and 62 (Level 5 numbers) would all have 4 out of 5 friends in common with 63 (a Level 6 of 8 number).

Realizing that certain numbers such as 1, 2, 4 or 128 will likely not be included in a clique, (unless we are looking for cliques of 1), it is still important to include them in the process as their inclusion will be required to eliminate any letters/people that are only friends with themselves. As an example, if a woman named Gertrud signed up to Facebook, but decided to not become friends with anybody before she abandoned the platform then she would show up as a LEVEL 1 number. LEVEL 1 numbers are not compatible with anybody except for themselves and thus will not belong to a clique. LEVEL 0 numbers (which can't really exist as they are 0), don't even like themselves, so it is quite a sad state of affairs for them and these poor people may need to seek counselling.

In an alternative embodiment of the present invention we may find it more beneficial to flip all the zeros to ones and ones to zeros. If the majority of people were to have very few friends we may find numbering the NOT FRIENDS as 1 a more economical method of organization rather than numbering the WE ARE FRIENDS method. Again, if the distribution is relatively equal, then it will not matter much as to how we organize the 0's and 1's.

<https://jamesmccaffrey.wordpress.com/2011/06/24/the-maximum-clique-problem/>

“It turns out that finding the maximum clique for graphs of even moderate size is one of the most challenging problems in computer science. The problem is NP-complete which means, roughly, that every possible answer must be examined. Suppose we have a graph with six nodes. First we'd try to see if all six nodes form a clique. There is Choose  $(6,6) = 1$  way to do this. Next we'd examine all groups of five nodes at a time; Choose  $(6,5) = 6$  ways. And so on, checking Choose  $(6,4) = 15$ , Choose  $(6,3) = 20$ , Choose  $(6,2) = 15$ , and Choose  $(6,1) = 6$  possible solutions for a total of 63 checks. (For the maximum clique problem we can stop when we find the largest clique so let's assume that on average we'd have to go through about one-half of the checks).

The total number of checks increases very quickly as the size of the graph,  $n$ , increases. For  $n = 10$  there are 1,023 total combinations. For  $n = 20$  there are 1,048,575 combinations. But for  $n = 1,000$  there are

10,715,086,071,862,673,209,484,250,490,600,018,105,614,048,117,055,336,074,437,  
503,883,703,510,511,249,361,224,931,983,788,156,958,581,275,946,729,175,531,468,  
251,871,452,856,923,140,435,984,577,574,698,574,803,934,567,774,824,230,985,421,  
074,605,062,371,141,877,954,182,153,046,474,983,581,941,267,398,767,559,165,543,  
946,077,062,914,571,196,477,686,542,167,660,429,831,652,624,386,837,205,668,069,

combinations. Even if you could perform one trillion checks per second it would take you  $3.4 \times 10^{281}$  years which is insanely longer than the estimated age of the universe (about  $1.0 \times 10^{10} = 14$  billion years).”

- James McCaffrey

*Note:*

*In appendix A, I will provide a strategy for using smaller numbers for our methods as working with very large numbers can be difficult for computers with limited resources.*

## **Appendix**

### **A – The Traveling Salesman Reduction (Optimization)**

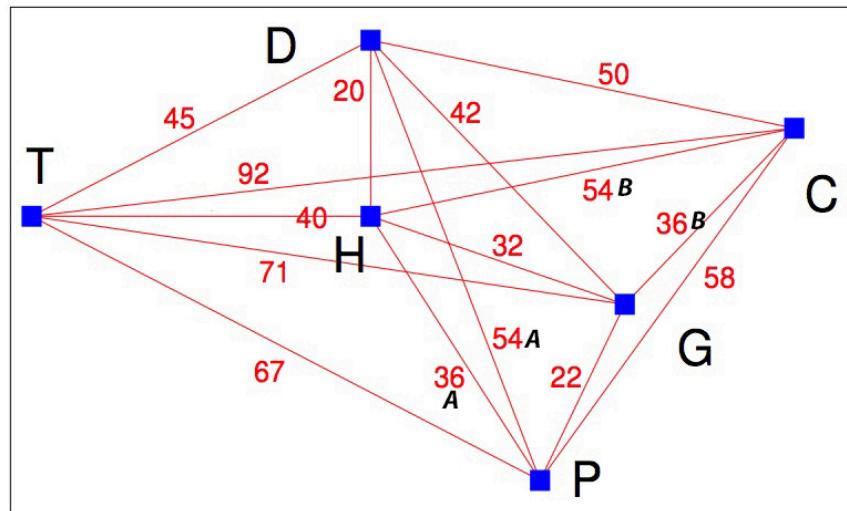
The travelling salesman problem (TSP) asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?" It is an NP-hard problem in combinatorial optimization, important in operations research and theoretical computer science.

The problem was first formulated in 1930 and is one of the most intensively studied problems in optimization. It is used as a benchmark for many optimization methods. Even though the problem is computationally difficult, a large number of heuristics and exact algorithms are known, so that some instances with tens of thousands of cities can be solved completely and even problems with millions of cities can be approximated within a small fraction of 1%.

The TSP has several applications even in its purest formulation, such as planning, logistics, and the manufacture of microchips. Slightly modified, it appears as a sub-problem in many areas, such as DNA sequencing. In these applications, the concept city represents, for example, customers, soldering points, or DNA fragments, and the concept distance represents travelling times or cost, or a similarity measure between DNA fragments. The TSP also appears in astronomy, as astronomers observing many sources will want to minimize the time spent moving the telescope between the sources. In many applications, additional constraints such as limited resources or time windows may be imposed.



TABLE 9



Note:

The above TSP example is from:

<http://people.ku.edu/~jlmartin/courses/math105-F11/Lectures/chapter6-part3.pdf>

Sabrina has the following list of errands and needs to find the shortest route:

**(H) Home** (The start and finish of her route).

(P) Pet store

(G) Greenhouse

(C) Cleaners

(D) Drugstore

(T) Target Store

The present invention can attack the TSP in a variety of ways! I will describe a couple of examples out of many possibilities with the present system. I begin by making sure that if there is a distance that is the same as another distance, that I can distinguish them. I do this by adding A or B etc., to the number. In the example provided we will see 36<sub>A</sub> and 36<sub>B</sub> and 54<sub>A</sub> and 54<sub>B</sub>. This is a critical step.

Each unique value is then assigned a binary number based on its size. The shortest path element receives the smallest binary number and vice versa. In the example provided we see that length 20 (the shortest) is linked to binary digit 1 and that length 92 (the longest) is linked to binary digit 16384.

We then need to create a situation wherein we can produce a number of logical “NOT” scenarios. To do this (in this example) we will use two or more variables together (collaborative variable). For example, 22 and 32 are connected at G and thus can become a single larger variable with a number of NOT’s. We can also call this HGP as these are the nodes that they are connected to. HPG can also double as PGH. We know that H stands for HOME and thus it will be the beginning and the end point of the journey. This tells us that in the necessary six-point journey (travel to each place once from home and also end at home via the shortest route), that we will require two different collaborative variables from TABLE 10 and one from TABLE 11. All three will work together to create our shortest path(s). (Not in the present example, but in some instances there could potentially be two different shortest paths of the same length).

Using this logic, we can start taking notes of our logical NOT’s. Looking at our first collaborative variable we can see that it is comprised of nodes HGP (32 & 22). Of course HGP reversed is also PGH (22 & 32). Being the same, neither can include 36A in its path from or to home because it would make the person on the journey visit the same place twice, which is not allowed in the TSP problem. If we look at the G node in this first collaborative variable, we can see that 32 and 22 are already used and 71, 36B and 42 can NOT be used because G has to go to P and H and thus these values are scratched or deleted as possibilities. Any value from P (22 already being used), is also scratched (67, 36A, 54A and 58), because logic states that we will need to use a number from Table 11 (with no H) for our middle collaborative variable and because any number from Table 10 would want to go straight home in two steps, which is NOT allowed.

TABLE 10

	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1		
	92	71	67	58	54B	54A	50	45	42	40	36B	36A	32	22	20		
H G P	1				1		1	1		1			1	1	1	6	17825
H P G	1				1		1	1		1		1		1	1	10	17825
H G C			1			1		1		1	1	1	1		1	20	4777
H D G	1		1	1	1				1	1		1			1	65	23592
H G D	1		1	1	1				1	1		1	1			68	23592
H D T				1	1			1			1	1	1	1	1	129	3102
H T D				1	1			1		1	1	1	1	1		160	3102
H D C		1	1				1			1		1	1	1	1	257	12334
H D P	1	1			1	1				1	1		1		1	513	25652
H P D	1	1			1	1				1	1	1	1			520	25652
H C G			1		1	1		1		1	1	1			1	1040	4777
H C D		1	1		1		1			1		1	1			1280	12334
H P C		1						1	1	1		1	1		1	2056	8421
H C P		1		1	1			1	1	1			1		1	3072	8421
H P T			1		1		1		1		1	1	1		1	4104	1365
H T P			1		1		1		1	1	1		1		1	4128	1365
H G T		1		1	1	1	1					1	1		1	8196	3849
H T G		1		1	1	1	1			1		1			1	8224	3849
H T C	1					1			1	1		1	1	1	1	16416	591
H C T	1				1	1			1			1	1	1	1	17408	591

We know that the green number 6 (the top number of the 2<sup>nd</sup> column from the right) of TABLE 10 must be 4 and 2 as binary numbers 4 and 2 are the only ones that will work to make 6. We also know that the white numbers are the ONLY compatible areas where the last two numbers for the journey can be found. TABLE 10 needs to provide both the first two and last two steps of the journey, due to the fact that the journey is six steps long. The middle two steps (two collaborative or added variables) will be provided from TABLE 11. In the present example we see that 6 and 129 are compatible and add up to **135**. 6 (a green number composed of two binary numbers 4 and 2) is a subgroup of 3102 (a white number composed of six binary numbers that also includes 4 and 2) and thus there is no conflict. They are compatible. Perhaps we can call them friends, like in the clique problem.

A computer program can easily determine this by consulting a list or program with the necessary elements. We can for example provide the software program with a list of compatible numbers. Rather than brute force however, the computer can simply look at the one number sums provided by our system to do its sorting, which will be much more efficient. Although it may seem like (and it is) a lot of work when dealing with only 6 travel points, we must remember that our system does not get

much more complicated as we begin to add new data points, whereas the presently used brute force method gets exponentially more difficult. For a small number of cities, the present methods may seem slower than brute force, but as the number of cities or data points grows larger, our methods become dramatically better and faster than other available methods.

For example:

- 6 = 4, 2
- 7 = 4, 2, 1
- 14 = 4, 2, 8
- 22 = 4, 2, 16
- 15 = 4, 2, 8, 1
- 31 = 4, 2, 1, 8, 16
- 3102 = 4, 2, 2048, 1024, 16, 8

...all belong to a family of binary numbers that contain 4 and 2. This will dramatically assist a computing device with its sorting and searching methods.

6 and 160 are also compatible, which add up to **166**. 10 and 129 are compatible and add up to **139**. 10 and 160 are also compatible and add up to **170**. We will need to find their compatibility with Table 11 before we can make a determination of which will be the shortest path however.

...go to next page...





know our starting point. As described below we can see that the standard brute force method would require checking 120 different paths for 6 cities.

<http://www.businessinsider.com/p-vs-np-millennium-prize-problems-2014-9>

“If we have a Traveling Salesman Problem with five cities, we have  $4 \times 3 \times 2 \times 1 = 24$  paths to look at. If we have six cities, we have  $5 \times 4 \times 3 \times 2 \times 1 = 120$  paths.

As we can already see with these small numbers of cities, the number of paths grows extremely quickly as we add more cities. While it's still easy to take a given path and find its length, the sheer number of possible paths makes our brute-force approach untenable. By the time we have 30 cities, the number of possible paths is about a 9 followed by 30 zeros. A computer that could check a trillion paths per second would take about 280 billion years to check every path, about 20 times the current age of the universe.

There are algorithms for the Traveling Salesman Problem that are much more efficient than this brute-force approach, but they all either provide some kind of approximate "good enough" solution that might not be the actual shortest path, or still have the number of needed calculations grow exponentially with the number of cities, taking an unacceptably long time for large numbers of cities. There is no efficient, polynomial time algorithm known for the problem.”

---Andy Kiersz

The present invention initially cuts this in half thus requiring only 60 variables to be analyzed, sorted and ordered into paths that satisfy the needed constraints of the TSP problem. For these small numbers the brute force system is likely better and faster, but as these numbers grow we can see how the present system becomes dramatically superior.

TABLE 14

Number of Cities	Brute Force Paths to Check	Variables in the present system
6	120	5 x 4 x 3 EQUALS 60
7	720	6 x 5 x 4 EQUALS 120
8	5,040	7 x 6 x 5 EQUALS 210
9	40,320	8 x 7 x 6 EQUALS 336
10	362,880	9 x 8 x 7 EQUALS 504
11	3,628,800	10 x 9 x 8 EQUALS 720
12	39,916,800	11 x 10 x 9 EQUALS 990
13	479,001,600	12 x 11 x 10 EQUALS 1320
14	6,227,020,800	13 x 12 x 11 EQUALS 1716
15	87,178,291,200	14 x 13 x 12 EQUALS 2184
16	1,307,674,368,000	15 x 14 x 13 EQUALS 2730
17	20,922,789,888,000	16 x 15 x 14 EQUALS 3360
18	355,687,428,096,000	17 x 16 x 15 EQUALS 4080
19	6,402,373,705,728,000	18 x 17 x 16 EQUALS 4896
20	121,645,100,408,832,000	19 x 18 x 17 EQUALS 5814
etc...	etc...	etc...
30	8,841,761,993,739,700,000,000,000,000	30 x 29x 28 EQUALS 24,360

Whereas the numbers in the brute force approach get exponentially larger, our numbers actually get comparatively smaller. As the example in the chart above shows; for 6 cities we require 120 groupings, but for 7, we only require 210 and so on, which is less than double. The number of variables does obviously grow, but in smaller and smaller ratios. Here is the pattern from 6 cities. From 6 to 7 cities we 2x our number. From 7 cities to 8 cities we 1.75x our number and from 8 to 9 we 1.6x, etc. Notice how the ratios get smaller and smaller! How awesome is that!

TABLE 15

Cities	Variables	x factors
6	60	2.0000
7	120	1.7500
8	210	1.6000
9	336	1.5000
10	504	1.4286



11	720	1.3750
12	990	1.3333
13	1320	1.3000
14	1716	1.2727
15	2184	1.2500
16	2730	1.2308
17	3360	1.2143
18	4080	1.2000
19	4896	1.1875
		etc...

For the TSP problem we will need to check a number of the satisfied lower numbers as the smallest number may not be the shortest path. A low number is likely to be one of the satisfied lower paths, but recognize from the example provided that the second shortest path was actually the third lowest number.

As soon as one of our low numbered satisfied paths add up to a number lower than higher numbered potential paths (or potential partial paths), we then know that we can eliminate the upper numbers from our search! This can literally save billions of years of searching! For example; As 442 gave us a nice short path that was satisfied, then any number above this need not be investigated any further for our shortest path query. Only seven numbers were smaller than 442 in our example. Twenty-three partial paths were longer than our completely satisfied full path. Thus we can exclude these from the search, which is awesome because the computer can save an enormous amount of time.

Sorting the partial paths from smallest to largest was the key this success! The other essential and novel element of the present invention is that rather than trying to determine the shortest path by analyzing the system via its individual parts, (for which in this example there are 15), we instead combine two or more single elements, which then provide us with a number of impossible options for each selected path. This creates a NOT in logic terms, that we would not have had access to, if dealing with our variables individually. The system is made larger, but this seems like a necessity to create the necessary constrained environment.

50 partial paths (or collaborative variables), each with two variables (connected to three letters) are required if the start and end points are known. 60 partial paths are required if no starting and ending point are known. Our Sudoku example also benefited from this strategy.

### **SOME OTHER NP-COMPLETE PROBLEMS**

Karp's 21 problems are shown below, many with their original names. The nesting indicates the direction of the reductions used. For example, Knapsack was shown to be NP-complete by reducing Exact cover to Knapsack. The methods of the present invention can be modified and transposed to solve all of these problems (and many more) in a much shorter time than is presently available.

**Satisfiability:** The Boolean satisfiability problem for formulas in conjunctive normal form (often referred to as SAT)

**0–1 integer programming** (A variation in which only the restrictions must be satisfied, with no optimization)

**Clique** (see also independent set problem)

**Set packing**

**Vertex cover**

**Set covering**

**Feedback node set**

**Feedback arc set**

**Directed Hamilton circuit** (Karp's name, now usually called Directed Hamiltonian cycle)

**Undirected Hamilton circuit** (Karp's name, now usually called Undirected Hamiltonian cycle)

**Satisfiability with at most 3 literals per clause** (equivalent to 3-SAT)

**Chromatic number** (also called the Graph Coloring Problem)

**Clique cover**

**Exact cover**

**Hitting set**

**Steiner tree**

**3-dimensional matching**

**Knapsack** (Karp's definition of Knapsack is closer to Subset sum)

**Job sequencing**

**Partition**

**Factoring**

**Max cut**

...and many more

[https://en.wikipedia.org/wiki/List\\_of\\_NP-complete\\_problems](https://en.wikipedia.org/wiki/List_of_NP-complete_problems)

## **B - SUDOKU**

Sudoku is a logic-based, combinatorial number-placement puzzle. The objective is to fill a  $9 \times 9$  grid with digits so that each column, each row, and each of the nine  $3 \times 3$  sub grids that compose the grid (also called "boxes", "blocks", or "regions") contains all of the digits from 1 to 9. The puzzle setter provides a partially completed grid, which for a well-posed puzzle has a single solution.

Completed games are always a type of Latin square with an additional constraint on the contents of individual regions. For example, the same single integer may not appear twice in the same row, column, or any of the nine  $3 \times 3$  sub regions of the  $9 \times 9$  playing board.

French newspapers featured variations of the puzzles in the 19th century, and the puzzle has appeared since 1979 in puzzle books under the name Number Place. However, the modern Sudoku only started to become mainstream in 1986 by the Japanese puzzle company Nikoli, under the name Sudoku, meaning "single number". It first appeared in a US newspaper and then The Times (London) in 2004, from the efforts of Wayne Gould, who devised a computer program to rapidly produce distinct puzzles.

In computational complexity theory, the Cook–Levin theorem, also known as Cook's theorem, states that the Boolean satisfiability problem is NP-complete. That is, any problem in NP can be reduced in polynomial time by a deterministic Turing machine to the problem of determining whether a Boolean formula is satisfiable. The theorem is named after Stephen Cook and Leonid Levin.

An important consequence of this theorem is that if there exists a deterministic polynomial time algorithm for solving a problem that can be reduced to Boolean satisfiability, then every NP problem can be solved by a deterministic polynomial time algorithm. The question of whether such an algorithm exists to solve a problem that has been reduced to Boolean satisfiability in deterministic

polynomial time is thus equivalent to the P versus NP problem, which is widely considered the most important unsolved (until the publication of this paper) problem in theoretical computer science.

We will now provide an example of Sudoku, but this is in no way meant to limit the present methods to this or any single problem. Any NP-Complete problem may be addressed with variations of the present invention as can be seen in the Cook-Levine papers or the proof based on the one given by Garey and Johnson[8]. Every NP problem can be reduced to an instance of a SAT problem by a polynomial-time many-one reduction.

First we will take an example of a standard 9x9 Sudoku puzzle.

TABLE 5

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Transposing our clique solving process for Sudoku will require that we transform the Sudoku numbers into binary values. Sudoku numbers 1-2-3-4-5-6-7-8-9 will now become 1-2-4-8-16-32-64-128-256. This will better allow for yes/no answers for our collaborative variables and simplify our solving process. The present method will SEEM QUITE COMPLICATED, but the process will be much more efficient then presently available methods of brute force or single variable constraint programming.

Why?

Because rather than having 81 individual variables, that need to be considered, we have created a system wherein two sets of 27 variables (54 variables in total) can be used instead. In a 16x16 grid with 256 variables, the present system will only require two sets of 64 variables (128 in total). As chart 1 below shows, traditional methods grow extremely rapidly, wherein the present method (although the variables DO grow), the variables grow at a much smaller rate! The variables of the present invention in the 100x100 grid have only one-fifth the variables of traditional constraint methods.

Grid Size	Traditional Number of Variables	Variables of the present invention		Sum
9x9	81	27	27	54
16x16	256	64	64	128
25x25	625	125	125	250
36x36	1296	216	216	432
49x49	2401	343	343	686
100x100	10000	1000	1000	2000

As we will see in the upcoming explanation of how to solve TSP we will be using what I call collaborative constraints for Sudoku. If we now focus on A1 we will notice that it will be provided with constraining elements from A2 and A3 (the horizontal constraints) and also constraints from A4 and A7 (the 3x3 constraint). Any numbers (or collaborative sum) in any of these areas will affect the available options for numbers which can be input into A1. If A1 already has a number in its area, this will also limit available options.

It is essential that we keep in mind that A1, A4 and A7 share the same 3x3 block as D1, D2 and D3. This is one of the elements that will allow our logic to work. They share a commonality, but also are provided with differing constraints. This holds true for all the other 3x3 blocks as well. For example, B2, B5 and B8 share many constraints with E4, E5 and E6, but each also have unique ones that make our system function.

Thus, if we create a number of logic statements, we can solve the Sudoku. For example, in the Sudoku example provided in Table 5, **IF** E1 is a three-variable number (which it is) and **IF** I know that F2, B2 and B9 all contain a variable that contains an identical value (which it does), (32 in binary and 6 in Sudoku), **THEN** B4 and E3 must contain that same number. (For visualization purposes **B4Z is equal to E3Y**).

TABLE 6

A1	A2	A3
A4	A5	A6
A7	A8	A9
B1	B2	B3
B4	B5	B6
B7	B8	B9
C1	C2	C3
C4	C5	C6
C7	C8	C9

D1	D2	D3	D4	D5	D6	A3	D8	D9
E1	E2	E3	E4	E5	E6	E7	E8	E9
F1	F2	F3	F4	F5	F6	F7	F8	F9

TABLE 7 (not really necessary)

X	Y	Z	X	Y	Z	X	Y	Z
X	Y	Z	X	Y	Z	X	Y	Z
X	Y	Z	X	Y	Z	X	Y	Z
X	Y	Z	X	Y	Z	X	Y	Z
X	Y	Z	X	Y	Z	X	Y	Z
X	Y	Z	X	Y	Z	X	Y	Z
X	Y	Z	X	Y	Z	X	Y	Z
X	Y	Z	X	Y	Z	X	Y	Z
X	Y	Z	X	Y	Z	X	Y	Z
X	Y	Z	X	Y	Z	X	Y	Z

X	X	X	X	X	X	X	X	X
Y	Y	Y	Y	Y	Y	Y	Y	Y
Z	Z	Z	Z	Z	Z	Z	Z	Z
X	X	X	X	X	X	X	X	X
Y	Y	Y	Y	Y	Y	Y	Y	Y
Z	Z	Z	Z	Z	Z	Z	Z	Z
X	X	X	X	X	X	X	X	X
Y	Y	Y	Y	Y	Y	Y	Y	Y
Z	Z	Z	Z	Z	Z	Z	Z	Z

Table 7 shows further breakdown into xyz zones, but realistically we may not require these in the preferred embodiment as for example, if we were to say that A1 and D1 shared a number, the only possible place that they can share that number is in the top space of D1 or furthest left of A1. No other location is possible.

After we name our independent sections we turn the Sudoku numbers 1-2-3-4-5-6-7-8-9 into binary place values 1-2-4-8-16-32-64-128-256. This will make everything much easier for a computational device to comprehend. An essential part and novel aspect of the present solution that has been

missing from the art is the ability to assist a computational device with fully understanding what variables do NOT need to be looked at during a searching effort. This will hold true in our next example (TSP) as well. The searching effort seems to be what takes up the computational devices time. The more possibilities that can be eliminated from the searching efforts the faster the output or answers will be found.

TABLE 8

16	4		64			
32			1	256	16	
	256	128				32
128			32			4
8		128	4			1
64			2			32
	32				2	128
			8	1	256	16
			128		64	256

16	4		64			
32			1	256	16	
	256	128				32
128			32			4
8		128	4			1
64			2			32
	32				2	128
			8	1	256	16
			128		64	256

We will need to list what I will call our levels for Sudoku. There are three variations. Level 1 numbers are single variables and will include 1, 2, 4, 8, 32, 64, 128, and 256. Once the Sudoku has been solved by our algorithm three of these unique numbers will add up to a larger collaborative variable (Level 3 number) and will reside in one of the 3x1 or 1x3 sections (i.e. A1, A2, D1, D2, etc.). Level 2 numbers are collaborative variables and will be the sum of any two different level one numbers. Again, Level 3 numbers are also collaborative variables and will be the sum of any three different level 1 numbers. This strategy better provides a computer with the ability to answer yes or no questions, as is a requirement in Boolean logic. For example, if A1=7 then we will logically know that A1 consists of a 4, 2 and a 1. By utilizing a binary number system, we allow only the one possibility, no matter what the numbers. This also tells us that the sum of A2 and A3 must be 249.

This is not the case when we use 1-9 numbering. If for example, we were to use 1-9 integer values and A1 equaled 15. This would not really tell us ANYTHING! The 15 in this scenario could be comprised of 456, 951, 942, 861, 852, 834, 762, or 753. By Utilizing binary digits rather than 1-9

integers we can dramatically reduce the time required to find a solution to the Sudoku via faster and better method of constraint programming that have not prior to the present invention been utilized.

**LEVEL 1 (one variable numbers)** = 1, 2, 4, 8, 16, 32, 64, 128, or 256

**LEVEL 2 (two variable numbers)** = Any one of --- 3, 5, 6, 9, 10, 12, 17, 18, 20, 24, 33, 34, 36, 40, 48, 65, 66, 68, 72, 80, 96, 129, 130, 132, 136, 144, 160, 192, 256, 257, 258, 260, 264, 272, 288, 320, or 384

**LEVEL 3 (three variable numbers)** = Any one of --- 7, 11, 13, 14, 19, 21, 22, 25, 26, 28, 35, 37, 38, 41, 42, 44, 49, 50, 52, 56, 67, 69, 70, 73, 74, 76, 81, 82, 84, 88, 97, 98, 100, 104, 112, 131, 133, 134, 137, 138, 140, 145, 146, 148, 152, 161, 162, 164, 168, 176, 193, 194, 196, 200, 208, 224, 259, 261, 262, 265, 266, 268, 273, 274, 276, 280, 289, 290, 292, 296, 304, 321, 322,, 324, 328, 336, 352, 385, 386, 388, 392, 400, 416, or 448

We can now simply run a program with a number of logic steps again and again until the Sudoku is fully solved. It can be modelled as a much simpler and faster constraint satisfaction problem that would likely use many less lines of code.

Here are some examples of logic steps that may be applied **recursively** as numbers get filled in and the constraints increase. This type of program should be effective at solving any Sudoku that would be input into it without the requirement of backtracking or brute force methods.

- **IF** A1 and A2 both = a Level 3 (3 variables added) number **AND IF** A3 = a Level 2 number **THEN** the one empty box (the x, y **or** z) of A3 must = 256 minus the sum of A1, A2 and A3. A3 must then be modified from a Level 2 number to a Level 3 number and the new addition placed in the one empty x, y or z space that is available. The full horizontal line would now in this scenario be complete.
- **IF** E1 contains a Level 3 (3 variables added) number **AND IF** (F2 **OR** D2) **AND** (B2 **AND** B9) contain the same Level 1 number **THEN** E3 and B4 should share that number (*in the only place they can, which is the z location*).



- We can also be more specific and even use the xyz variables if desired. **IF** (D3 and E1) and (C3 and C8) contain the sum 128 (a Level 1 variable), or the sum 129, 130, 132, 136, 144, 160, 192, or 384 (a Level 2) or 131, 133, 134, 137, 138, 140, 145, 146, 148, 152, 161, 162, 164, 168, 176, 193, 194, 196, 200, 208, 224, 385, 386, 388, 392, 400, 416 or 448 (a Level 3), **THEN** C4(Y) is 128 (8 in Sudoku).
- We can also make generalized family systems. For example, **IF** B3 and B5 contain a same number (such as binary 4) or contain a higher level number that includes binary 4, (by binary necessity to acquire its sum such as 5 (4,1) or 13 (1, 8, 4), **AND** (E1 and E3) contain Level 3 numbers **THEN** E2(Z) should be filled with that number. The (example 4) number is then converted to the Sudoku number 3. (*Note: the pictured examples, do not provide this example*).
- MANY more logic operators can be created and will guarantee a solution to even the most complicated Sudoku. Brute force methods will no longer be required to solve these types of problems, but they may be used in collaboration. We can design a complete system utilizing only a portion of all possible logic operators. The simpler the Sudoku, the less operators that will likely be required.

The key to the Sudoku solution seems to be in breaking the puzzle down into a variety of smaller units, each of which share two unique logic systems. By logic systems I am referring to our 3x1 units which each belong to both a 9x1 horizontal and a 3x3 group **and** our 1x3 units which belong to 1x9 vertical and 3x3 groups. When logic operators make these two groups talk to each other, every blank in a satisfiable Sudoku can easily be found. Each Sudoku number is assigned a binary digit (1, 2, 4, 8, 16, 32, 64, 128 or 256). We can apply the logic operators recursively (again and again as more blank spaces are filled with numbers) to fill in the puzzle. Once it is complete and everything is satisfied logically, we can convert our numbers back to 1-9 and our puzzle is solved in polynomial time.

This method can also be utilized for larger puzzles, we simple need to increase our numbers. A 16x16 Sudoku would require 1x4 and 4x4 logic groupings and our binary numbers would go to 65536.

*Note:*

*A similar approach can be used to solve the "Eight Queens Puzzle". If the chess board has an even length (i.e. 8x8) it is easy to visualize, but if it is an odd length (i.e. 9x9), we need to adjust the strategy wherein we have two 5x5's with some crossover between each sub-group...but that's for another paper.*

## C – Further Background

The present invention takes a novel approach to solving NP-Complete problems and provides steps that a computational device and software program can follow to accurately solve NP-class problems without the use of heuristics or brute force methods. The present invention (methods and computational devices) provides methods that are fast and accurate if utilized properly.

The present inventor states that the solution to solving the “**Does P=NP?**” question (and our ability to design algorithms to solve such problems efficiently) lies in a novel method presented for searching, filtering, combining and structuring data. Those skilled in the art seem to have failed to recognize the power of representing data variables as specific unique binary place values and/or their sums (collaborative variables), prior to searching or filtering selected data sets. By binary place values, the present inventor is referring to the selected place values for 1 or 0 in the binary numbering system. The numbers 1, 2, 4, 8, 16, 32, 64, etc., are all examples.

What the inventor refers to when stating “**their summations**” is the unique value that each grouping of two or more of these numbers can create when added together. For instance, if we take the number 7 and state that it must consist of unique binary numbers, then there is only one possible outcome. The numbers to create a 7, must be 1, 2 and 4. If we take the number 255 and state that it must consist of unique binary numbers, then there is also only one possible outcome. If (again) no number is allowed to repeat, then 255 must be created from 1, 2, 4, 8, 16, 32, 64 and 128.

By constraining the binary system to **unique** values (meaning that no number can be used or added more than once in any collaborative variable) a number such as 255 can store a tremendous amount of information. In the described system 255 will thus state that there are eight different numbers that are required to create the sum 255 and that the eight numbers required to fulfill its creation successfully are required to be 1, 2, 4, 8, 16, 32, 64 and 128. It can also indirectly provide us with a number of sub-groups that will be described in more detail, in the Click Problem example provided further on in this paper. As another example, if we are specifically looking for two-digit variables, the numbers 3, 5, 6, 7, 9 etc., will all be options, but 1, 2, 4 and 8 will not.

Note: Please remember that the number 5 MUST consist of (4 and 1) as (2, 2 and 1) are not allowed, because the 2 would be duplicated. The avoidance of duplicated variables is a key to the success of the present system as these duplicate variables seem to be one of the difficulties that make certain questions in computational complexity theory very difficult to answer.

The present invention also describes a novel method for breaking specific problems into logical groupings that the present inventor (John Archie Gillis) has defined as collaborative variables. They utilize novel binary representations/conversions, so that one can more easily and quickly determine selected and desired informational outputs. In a number of instances for the present system to work, we must organize two or more variables into larger variables (collaborative), so that we can determine logical NOT's. It seems that in many cases that finding constraints by looking at a problem as single variable data points is insufficient. This seems specifically true as will be seen in the TSP problem, as constraints or NOT's are difficult to come by unless two or more variables are joined as one. We can then utilize a system of permutations, logic and searching to find fast and accurate answers.

Numbers are organized into various groups, levels and hierarchies as will be shown in the three different NP-Complete examples provided below. In the present system the higher value of each number does not necessarily mean that it has prominence over any other number. The numbers are instead listed into values that assist with sorting specific things into selected and specific groupings.

The present invention provides a number of methods for solving NP-class problems. NP-class problems include many pattern-matching and optimization problems that are of great practical interest, such as determining the optimal arrangement of transistors on a silicon chip, developing accurate financial-forecasting models, or analyzing protein-folding behavior in a cell. Since all the NP-complete optimization problems become easy with the present methods, everything will be much more efficient. Transportation of all forms can now also be scheduled optimally to move people and goods around quicker and cheaper. Manufacturers can improve their production to increase speed and create less waste.

Developments in vision recognition, language comprehension, translation and many other learning tasks will now become much simpler. The present inventor feels that by utilizing the systems of the

present invention in numerous fields, that the invention will have profound implications for mathematics, cryptography, algorithm research, artificial intelligence, game theory, internet packet routing, multimedia processing, philosophy, economics and many other fields.

## D - About the Author

John Archie Gillis is from Halifax, NS, Canada. This is his solution to the P versus NP problem. He is presently working on a number of interesting inventions that he believes many people could benefit from provided the concepts can be brought to light. He is particularly interested in applying the present strategies to biological systems, genetic therapies and cancer research. A number of his other inventions can be found via a Google Patent search or at <https://archiegillis.com/inventions/>

If you have any questions/comments or are interested in the companies I am building that will host this IP, please don't hesitate to contact me at [johnarchiegillis@gmail.com](mailto:johnarchiegillis@gmail.com)

## E - References

1. Cook, S.A. (1971). "The complexity of theorem proving procedures". Proceedings, Third Annual ACM Symposium on the Theory of Computing, ACM, New York. pp. 151–158. doi:10.1145/800157.805047.
2. Wikipedia contributors. (2018, June 19). Clique problem. In *Wikipedia, The Free Encyclopedia*. Retrieved 13:57, June 28, 2018, from [https://en.wikipedia.org/w/index.php?title=Clique\\_problem&oldid=846513850](https://en.wikipedia.org/w/index.php?title=Clique_problem&oldid=846513850)
3. Wikipedia contributors. (2018, May 14). NP-completeness. In *Wikipedia, The Free Encyclopedia*. Retrieved 14:00, June 28, 2018, from <https://en.wikipedia.org/w/index.php?title=NP-completeness&oldid=841292328>
4. Wikipedia contributors. (2018, June 20). Travelling salesman problem. In *Wikipedia, The Free Encyclopedia*. Retrieved 14:01, June 28, 2018, from [https://en.wikipedia.org/w/index.php?title=Travelling\\_salesman\\_problem&oldid=846797001](https://en.wikipedia.org/w/index.php?title=Travelling_salesman_problem&oldid=846797001)
5. Wikipedia contributors. (2018, June 22). Sudoku. In *Wikipedia, The Free Encyclopedia*. Retrieved 14:02, June 28, 2018, from <https://en.wikipedia.org/w/index.php?title=Sudoku&oldid=847081582>

6. The Traveling Saleswitch Problem. <http://people.ku.edu/~jlmartin/courses/math105-F11/Lectures/chapter6-part3.pdf>
7. Gödel's Lost Letter and P=NP <https://rjlipton.wordpress.com/the-gdel-letter/>
8. Garey, Michael R.; David S. Johnson (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman. ISBN 0-7167-1045-5

2nd DRAFT