3SAT

Brute Force,
Heuristic,
& Mappings

Name: Elijah J. Yohannan
Email: ejyohannan@crimson.ua.edu
CWID: 12064449
Date: April 28, 2023

Introduction

For my project, I wrote a brute force and heuristic method that aim to find solutions to inputs to the 3SAT problem. Rather than merely addressing the decision problem of whether or not the whole formula can be satisfied, both methods seek an optimized solution, attempting to satisfy as many clauses as possible.

Brute Force

My brute force method works as follows:

1. Start with all variable values set to FALSE.

   For instance, if we have variables x1, x2, x3, and x4, start with FFFF.

   Determine how many clauses are satisfied by this combination of values.
2. Flip the latter-most value to TRUE, as if we were counting in binary, giving us FFFT.
   a. **IF** this set of values is satisfies a greater number of clauses than our previous max, update the maximum number of clauses satisfied and the store that combination of values.
   b. **Else** continue to the next combination of values (in this case, FFTF).
3. Continue this process until either all clauses are satisfied, or until all combinations of values are tested.

Note that, unless a combination satisfying all clauses is reached first, a total of $\mathbf{2^n}$ clauses will be tested, where **n** is the number of variables. So, for an input with variables x1 to x10 and 30 clauses, a total 2^{10}=1024 combinations could potentially be tested.

Heuristic

My heuristic method works as follows:

1. Count the number of occurrences of each variable in the formula, where positive and negative versions are considered distinct from each other.
    a. For instance, if we have the formula,

    ```
    (x1 || x2 || x3)
    && (x1 || -x2 || -x4)
    && (x2 || x3 || x4)
    ```

    then we will have the following counts:

    ```
    x1: 2     x2: 2     x3: 2     x4: 1
    -x1: 0    -x2: 1    -x3: 0    -x4: 1
    ```

2. Choose the variable with the highest count and remove all clauses that are satisfied by its assignment. Appropriately reduce the counts of the variables that were in those clauses, and remove the variable that was just assigned from future consideration.
    a. Continuing the previous example, we would select x1=T, since it has the highest count (2), and is first in the order depicted above. Removing the corresponding clauses and decreasing the appropriate counts would give us the following:

    ```
    (x2 || x3 || x4)
    x1: R     x2: 1     x3: 1     x4: 1
    -x1: R    -x2: 0    -x3: 0    -x4: 0
    ```

    Note that x1 and -x1 now have their counts set to “R”, meaning they have been *removed* from consideration.

3. Repeat the process until no further improvement can be made.
    a. In this case, x2 will be assigned a TRUE value, satisfying the final clause. This will give us the final result: **`TTXX`**, meaning

       `x1=T,  x2=T,  x3=X,  x4=X`

       'X', in this case, means that it does not matter what value is assigned to that variable.

Note that, unlike the brute force method, the maximum number of iterations is **n**, where **n** is the number of variables to be selected from (in this case, 4). So, whereas the brute force is executed in **exponential** time, the heuristic is executed in **linear** time.

Hard Problem

Though my brute force and heuristic often satisfy the same number of clauses, especially with smaller inputs, I came up with the following small input which exploits the weakness of greediness in my heuristic algorithm (the equivalent formula is denoted to the right):

```
n=9
1 4 5                    (1 || 4 || 5)
1 -4 5                   && (1 || -4 || 5)
1 4 -5                   && (1 || 4 || -5)
2 6 7                    && (2 || 6 || 7)
2 -6 7                   && (2 || -6 || 7)
2 6 -7                   && (2 || 6 || -7)
3 8 9                    && (3 || 8 || 9)
3 -8 9                   && (3 || -8 || 9)
3 8 -9                   && (3 || 8 || -9)
-1 -2 -3                 && (-1 || -2 || -3)
$
```

For the purpose of comparison, note that if you were to run the brute force method on this input, you would get the following output:

```
Answer:
      Number of clauses satisfied: 10
      Number of iterations: 64
      Values: FFFTTTTTT
```

You can examine the clauses of the input file and see that the values stated here (x1 to x3 set to FALSE and x4 to x9 set to TRUE) do indeed satisfy all 10 clauses, and, by extension, the formula as a whole.

However, if we use heuristic method on this input, we will instead get the following output:

```
Answer:
        Number of clauses satisfied: 9
        Number of iterations: 4
        Values: TTTXXXXXX
```

Note that the number of clauses satisfied here is 9, which is less than 10. What went wrong? Let's walk through the process step-by-step.

When we begin, we have the following counts:

```
Counts (R = removed):
x1:3    x2:3    x3:3    x4:2    x5:2    x6:2    x7:2    x8:2    x9:2
-x1:1   -x2:1   -x3:1   -x4:1   -x5:1   -x6:1   -x7:1   -x8:1   -x9:1
```

Since x1 has the highest count (3) and happens to be first in the order depicted above, it will be removed first by setting its value to TRUE. This removes the first three clauses of the formula, leaving us with the following remaining formula and new set of counts:

```
n=9
2 6 7                   (2 || 6 || 7)
2 -6 7                  && (2 || -6 || 7)
2 6 -7                  && (2 || 6 || -7)
3 8 9                   && (3 || 8 || 9)
3 -8 9                  && (3 || -8 || 9)
3 8 -9                  && (3 || 8 || -9)
-1 -2 -3                && (-1 || -2 || -3)
$
```

```
Counts (R = removed):
x1:R    x2:3    x3:3    x4:0    x5:0    x6:2    x7:2    x8:2    x9:2
-x1:R   -x2:1   -x3:1   -x4:0   -x5:0   -x6:1   -x7:1   -x8:1   -x9:1
```

Note that we will no longer consider the possibility of x1=F, as we have greedily decided to set x1 to TRUE. Now, since x2 has the highest count (3), we will remove it next by setting it to TRUE. This removes the next three clauses of the formula, leaving us with the following remaining formula and new set of counts:

```
n=9
3 8 9                                   (3 || 8 || 9)
3 -8 9                                  && (3 || -8 || 9)
3 8 -9                                  && (3 || 8 || -9)
-1 -2 -3                                && (-1 || -2 || -3)
$

Counts (R = removed):
x1:R    x2:R    x3:3    x4:0    x5:0    x6:0    x7:0    x8:2    x9:2
-x1:R   -x2:R   -x3:1   -x4:0   -x5:0   -x6:0   -x7:0   -x8:1   -x9:1
```

Now both x1 and x2 have been set to TRUE. Of the remaining variables, x3 has the highest count (3), so we likewise set it to TRUE and remove all clauses that it satisfies. This gives us the following remaining formula and counts:

```
n=9
-1 -2 -3                                (-1 || -2 || -3)
$

Counts (R = removed):
x1:R    x2:R    x3:R    x4:0    x5:0    x6:0    x7:0    x8:0    x9:0
-x1:R   -x2:R   -x3:R   -x4:0   -x5:0   -x6:0   -x7:0   -x8:0   -x9:0
```

Note that, even though there is one more clause to satisfy, there is no way to satisfy them, as x1, x2, and x3 have already been set to TRUE; their negative values are not considered.

The program goes through one more iteration, and, upon discovering that no additional clauses have been satisfied by further action, will terminate and display the final result. This example well displays the type of input that causes the heuristic to fail. Failure in this case, means failing to satisfy the maximum number of clauses, as shown by execution of the brute force method.

Intractable Problem

Creating an intractable problem for my brute force algorithm is fairly straightforward. Generally speaking, most inputs with a high number of variables will show themselves to be intractable, as the number of potential combinations to test is $\mathbf{2^n}$, where **n** is the number of variables.

I tested with a file named **`intractable.dat`**, which contains 218 clauses. More important than the number of clauses, however, is the number of variables, which in this case is 50 (x1, x2, ..., x49, x50). Selecting the heuristic method, as we would expect, takes a little under 50 iterations, and produces the following output:

```
Answer:
     Number of clauses satisfied: 215
     Number of iterations: 46
     Values: TTFFTFFXTFFXTFFTFTTTTFTFXFFTTFTXFFFFXTFFTTFTTFTFTF
```

This is pretty good; 215 out of 218 clauses are satisfied.

Running the brute force method on this input, however, has a very different result. Many, many lines are outputted to the terminal, each representing 1000 new combinations of variable-values tested. The output after about 20 seconds will look something like this:

```
------------ 5175000 iterations tested ------------
------------ 5176000 iterations tested ------------
------------ 5177000 iterations tested ------------
------------ 5178000 iterations tested ------------
------------ 5179000 iterations tested ------------
------------ 5180000 iterations tested ------------
------------ 5181000 iterations tested ------------
------------ 5182000 iterations tested ------------
------------ 5183000 iterations tested ------------
------------ 5184000 iterations tested ------------
```

It is necessary the to terminate program with **'control+c'**. Every now and then, we notice a different output quickly pass by, in the following format:

```
------------ 4194000 iterations tested ------------
------------ 4195000 iterations tested ------------

Current max satisfiable: 209
Current max string:
FFFFFFFFFFFFFFFFFFFFFFFFFFTFFFFFFFFFFFTFTFTFFTFFF
x1=F    x2=F    x3=F    x4=F    x5=F    x6=F    x7=F    x8=F
x9=F    x10=F   x11=F   x12=F   x13=F   x14=F   x15=F   x16=F
x17=F   x18=F   x19=F   x20=F   x21=F   x22=F  x23=F    x24=F
x25=F   x26=F   x27=F   x28=T   x29=F   x30=F   x31=F   x32=F
x33=F   x34=F   x35=F   x36=F   x37=F   x38=F   x39=F   x40=T
x41=F   x42=T   x43=F   x44=T   x45=F  x46=F    x47=T   x48=F
x49=F   x50=F
Number of iterations tested: 4195657

------------ 4196000 iterations tested ------------
------------ 4197000 iterations tested ------------
```

These unique parts of the output demonstrate instances where a combination of values is found that satisfies a greater number of clauses than the previous max. However, this problem will go on and on, and will eventually need to be terminated.

Why is this? Recall that the maximum number of iterations is $\mathbf{2^n}$, where n is the number of variables. $\mathbf{2^{50}}$ is equal to **1.1258999e+15.**

(1.1258999e+15 total combinations) / ((5184000 iterations) / (20.37 seconds))

= 4424109009 seconds.

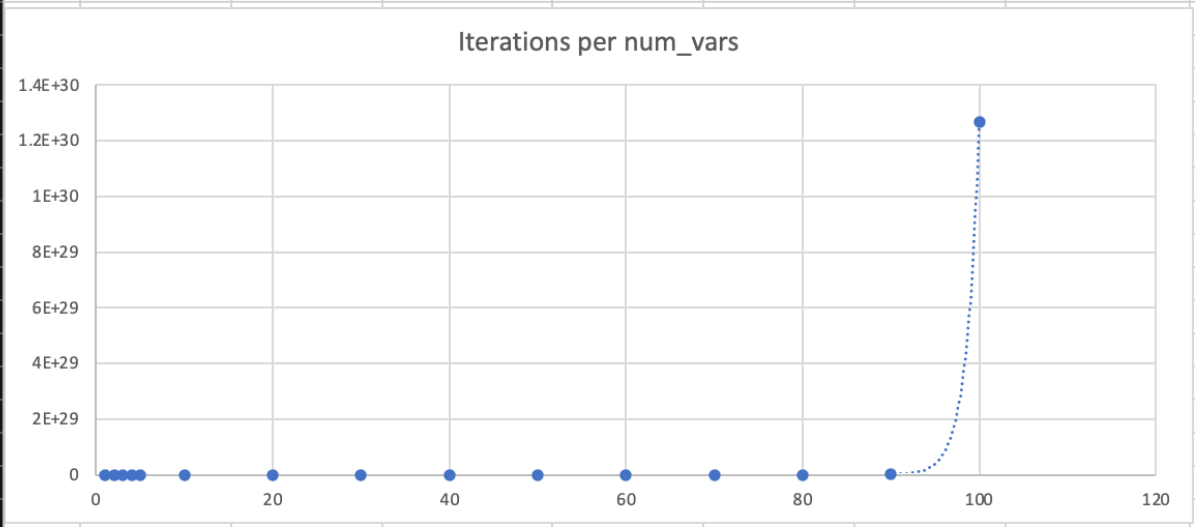
(4424109009 seconds) / (31536000 seconds per year) = 140 years!

So, even though we were able to satisfy 209 clauses in only 20 seconds, the total time it would take to use the brute force method would be, at the maximum (testing all combinations), 140 years!

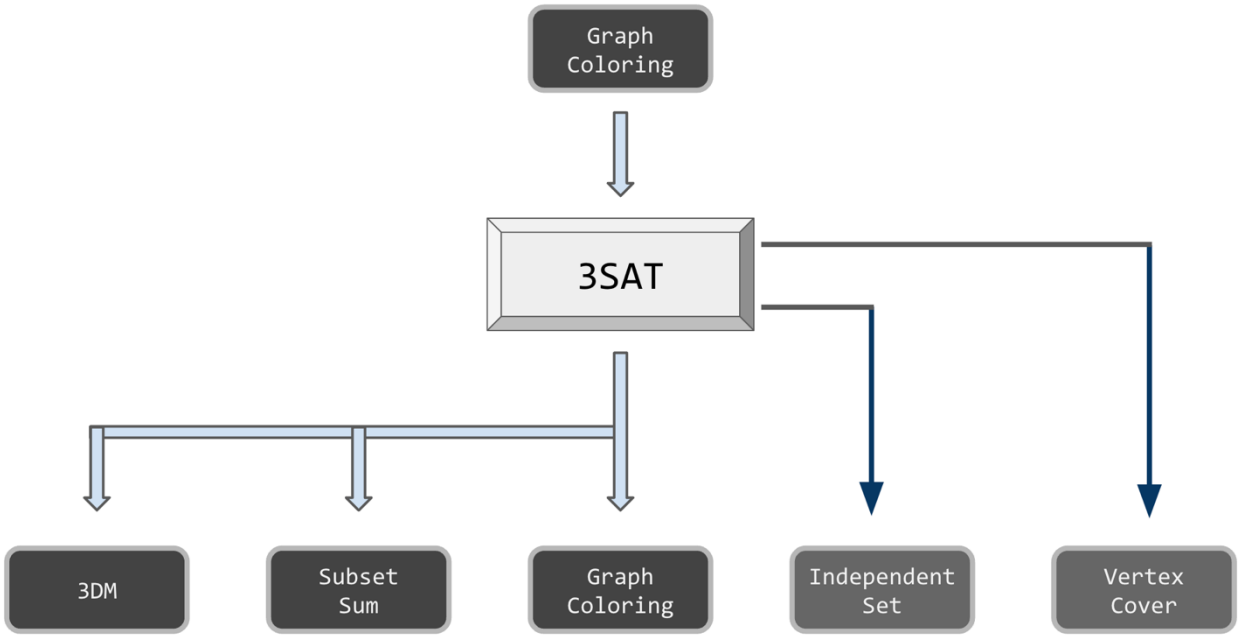
Thus, the input is intractable for the brute force method.

See the chart and graph below to determine the number of iterations it would potentially take to run the brute force method on inputs with different numbers of variables:

	A	B	C	D	E	F	G	H
1	Inputs	Iterations	Time (ms)	Time (seconds)	Time (hours)	Time (Days)	Time (Years)	Time (Millennia)
2	1	2	0	0	0	0	0	0
3	2	4	0	0	0	0	0	0
4	3	8	0	0	0	0	0	0
5	4	16	0.1	0	0	0	0	0
6	5	32	0.1	0	0	0	0	0
7	10	1024	4	0	0	0	0	0
8	20	1048576	4120.3	4.1	0	0	0	0
9	30	1073741824	4219159.1	4219.2	1.2	0	0	0
10	40	1.09951E+12	4320418954	4320419	1200.1	50	0.1	0
11	**50**	**1.1259E+15**	**4.42411E+12**	**4424109009**	**1228919.2**	**51205**	**140.3**	**0.1**
12	60	1.15292E+18	4.53029E+15	4.53029E+12	1258413229	52433884.6	143654.5	143.7
13	70	1.18059E+21	4.63901E+18	4.63901E+15	1.28862E+12	53692297780	147102185.7	147102.2
14	80	1.20893E+24	4.75035E+21	4.75035E+18	1.31954E+15	5.49809E+13	1.50633E+11	150632638.2
15	90	1.23794E+27	4.86436E+24	4.86436E+21	1.35121E+18	5.63005E+16	1.54248E+14	1.54248E+11
16	100	1.26765E+30	4.9811E+27	4.9811E+24	1.38364E+21	5.76517E+19	1.5795E+17	1.5795E+14



Mappings



I personally created code to accomplish the two mappings depicted by the thin, dark-blue arrows above: 3SAT to Independent Set and 3SAT to Vertex Cover; more on this later.

3-D Matching (Sungmin Kim)

I first connected with Sungmin Kim, who had created a program to map from 3SAT to 3-Dimensional Matching. I sent her the intractable and hard problems examined in the previous sections of this paper. She sent me back the following results for the hard problem:

```
Bruteforce: N/A

Heuristic: Size = 53

Note that I can only use the mapping to solve the decision problem
(i.e., whether the expression is satisfiable).
The expression is satisfiable when the number of triples that can be
matched is 6 * (the number of clauses), which in this case is 60.

The heuristic can only be used to confirm, not deny, satisfiability.
The number of triples matched is 53, so the results are inconclusive.

The problem is intractable with my current bruteforce algorithm.
The size of the problem has exploded from N = 10 to N = 1290.
```

In summary, even though we know that the problem is indeed solvable, both our heuristics were only able to find a solution to *most* of the clauses, not all. In her own words:

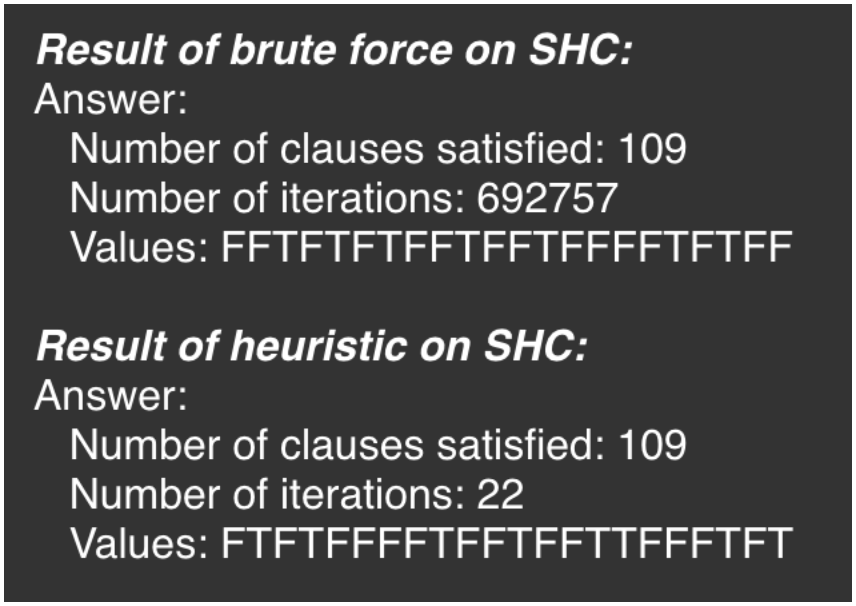
> *"The heuristic works by sorting the triples by the number of triples that they overlap with and then constructing a maximal set by considering each triple in that order. The heuristic fails because each triple overlaps with the same number of triples."*

Interestingly, her brute force approach found this input to be intractable, as the mapping caused the relevant input size to grow over 100-fold. Upon my asking whether she had run the intentionally-intractable file, she informed me that most 3SAT problems are intractable for her brute force program because the mapping "translates a few terms and clauses into a large number of triples." She reported that two variables and three clauses in the 3SAT space are about as much as her program handle when translated over, with certain exceptions.

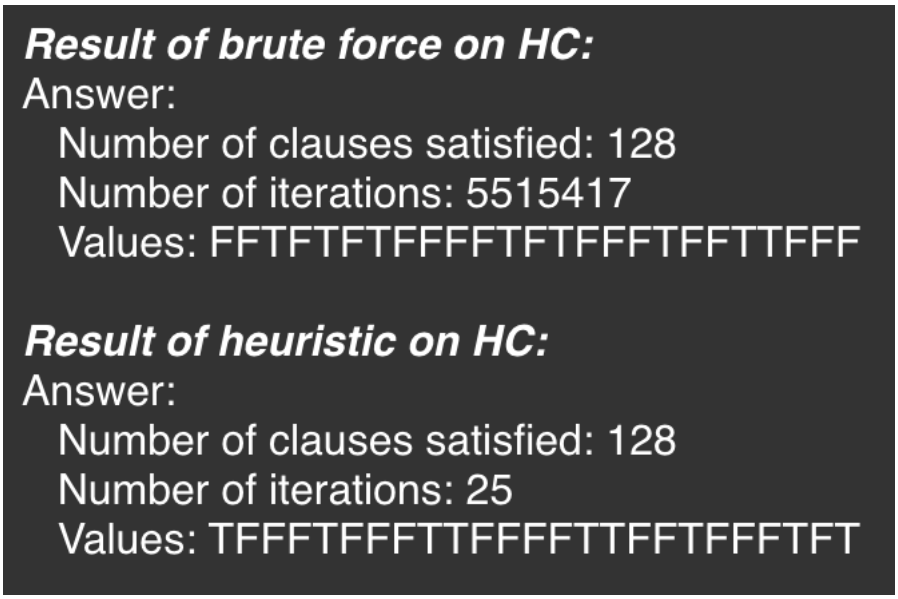
Graph Coloring (Sam Gaines)

I next connected with Sam Gaines, who had completed mappings both *to* and *from* the Graph Coloring problem. He sent me two files, one which he described as "Hard-to-Color," and the other as "Slightly-Hard-to-Color." I will henceforth abbreviate these to "HC" and "SHC." He informed me that he, on his end, had to choose a k-value "to make the graphs *decision* problems for 3SAT." In this case, he chose k=3 for the two files. He reported that while both graphs are indeed 3-colorable, his heuristic only found a valid 4-coloring, ultimately failing to find a valid 3-coloring. Given that the graphs were, in actuality, colorable, success for my heuristic would mean finding a set of boolean values that satisfies all clauses.

I ran both of my solvers on the SHC input, and got the following results:



Given that there were a total 109 clauses in the file he sent, both methods successfully found a solution to the problem. I then ran the two methods on the HC input, and got the following results:



Likewise, given that there were a total 128 clauses in the file he sent, both methods successfully found a solution to the problem.

Next, taking the reverse approach, I sent my hard problem to Sam, who mapped it to both the 3-coloring problem and the 4-coloring problem. He informed me that his heuristic **failed** on both. He explained, “in the 3-coloring problem it suggested a 4-coloring and in the 4-coloring problem it suggested a 5-coloring.” He also ran the problems through his brute force solver. This revealed that a 3-coloring was indeed possible for the 3-coloring mapping. However, he found that running his brute force on the 4-coloring mapping for a few minutes turned up no conclusion; he terminated the program, concluding it to be, for our purposes, intractable.

After, Sam ran his programs on my intractable file which, when mapped to his space, produced a fairly massive file. He said that the input was also intractable for his brute force, due to the size of the graph. Concerning his heuristic, he reported that it “found a 4-coloring, but failed to find a 3-coloring, so equivalently finds that there is not a solution to the 3SAT problem.”

Lastly, Sam sent me a problem that was intractable in his own space, mapped to 3SAT. He sent me two files, one for k=3 and the other for k=4. He informed me that his heuristic had failed to find a solution for k=3, but not k=4. I ran both problems through my heuristic and got the following results:

Results of Heuristic on 3sat-intractable-graph-k3
Greedy Hueristic used
Answer:
  Number of clauses satisfied: 1820
  Number of iterations: 227
  Values:
TTFTFFFFTTFFTFFFFTFFFFFFTFFFTFTFFFTFFTFTFFFFTFTFTFFFTFFT
FFTTFFFFTFTFFTFFFTFTFFFTTFFFFTTFFFFTFFFFTFFTFFFFFFFTFFTFF
TFFTFFTFFTFFTFFTFFFFFFTFTFFFTFFFTFFTFFTFTFTFFFFFFTFFTFFTF
FFTFFTFTFTFFFTFFTFFTFTFFFTFFFTFTFFFTFFTTFFFFTFFFTFFT

Note that there are a total of 1833 clauses in the formula.

Results of Heuristic on 3sat-intractable-graph-k4
Greedy Hueristic used
Answer:
  Number of clauses satisfied: 2779
  Number of iterations: 368
  Values:
TTFFTFFTFFFTTFFFTFFFFFFTFFTFFFTFTFFFFTFFFFFTTFFFFFFFTFFFT
FFFTFFFFFTFFTFFFFTFTFTFFFTFFFFTFTFFFTFFFTFFFTFFFFFTTFFFFT
FFTFFFFFTFFFFFFTFFFFFFFFFFFFFTFFTFFFFTTFFFFFFTFFTFFFFTFFT
FFTFFTFFFTFFFTFFFTTFTFFTFFTFTFFFFFTFFTTFTFFFTFFFFTTFFTFT
FFTFFFFTFFTFFFTFFFTFFTTFFFTFTFFFFTFFFFTFFFTFTFFFFFTFTFFTT
FFFFTFFFFFTFFTFTFXTFFTTTFFTFTFFFTFTXTTFFFFTFFFTTFTTTTTFTT
TTFFFFXXTFTXFFXXTTTTTXTFFTTFTXFFTT

Note that there are a total of 2794 clauses in the formula.

So we see that, in both cases, my heuristic failed to find a solution that satisfies all clauses. This implies that, at least in terms of accuracy, Sam's heuristic outperformed mine for the k=4 problem. Both problems were intractable for my brute force. This is to be expected, as the

number of variables found in the two problems were 226 and 376. Given that it would take 140 years to solve an input with 50 variables, these problems could perhaps be deemed *extremely* intractable for my brute force.

Subset Sum (Rachel Qualls)

I also connected with Rachel Qualls, who had created a mapping from 3SAT to Subset Sum. Both her heuristic and brute force found a solution to my hard problem. Their outputs are shown below:

```
brute force:
        [10000000001000000000, 1000000001010000000, 100000000001000000,
10000000001010000, 1000000000000110, 100000000001010, 10001110000000,
1000001110000, 110000000000, 10000000000, 10000000000, 100000000, 100000000,
10000000, 100000, 100000, 10000, 1000, 1000, 100, 100, 10]
huristic:
        [10000000001000000000, 1000000001010000000, 100000000001000000,
10000000001010000, 1000000000000110, 100000000001010, 10001110000000,
1000001110000, 110000000000, 10000000000, 10000000000, 100000000, 100000000,
10000000, 100000, 100000, 10000, 1000, 1000, 100, 100, 10]
```

She also ran her programs on my intractable input. She reported that her brute force also found the input to be intractable, and her heuristic was only able to find an approximation, shown below:

```
        we approximate sum to:
11111111111111111111111111111111111111111111111111111333333333333
33333333333333333333333333333332103110121022321330301031102122212
11012123301302211223232121222313221112122021310211222212022232132
12212301222123231010223302231221221332212121022121021130122221312
2311312331211130
```

Independent Set

Now to explain my own mappings. The first one I did was 3SAT to Independent Set. The procedure I followed came from this presentation from University of Illinois Urbana-Champaign:

Site: https://courses.engr.illinois.edu/cs374/fa2020/lec_prerec/23/23_2_0_0.pdf

The following is my summary of the steps:

1. Consider the first clause. For each variable in the clause, create a new vertex. You can give each vertex a unique ID if you wish (since all vertices are distinct from each other), but you must label the vertex with an x-value (such as x1, x2, -x3, etc.).

   Create connections between each of these three clauses, forming an undirected triangle.
2. Repeat this process with every other clause, creating a total 3***c*** vertices, where **c** is the number of clauses.
3. Connect all vertices that have opposite labels, such as x1 and -x1.

See the example below, which I used to test and verify the functioning of my own code:

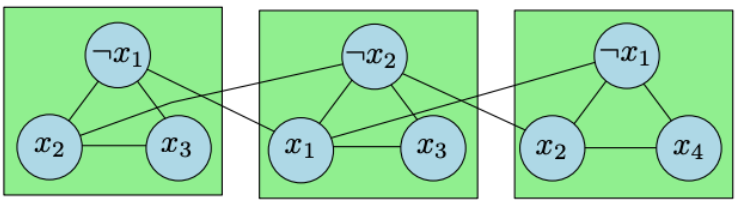


Figure: Graph for $\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

Each of these steps make sense considering the fundamental rule of an Independent Set: no two chosen vertices can be connected by a single edge. Step 1 of the process ensures that at

most one vertex/literal will be selected from each triangle/clause. Step 2 repeats this for each triangle/clause. Step 3 ensures that opposite x-values cannot be chosen. This again makes much sense; if, for example, if we had twelve instances of x1 and one instance of -x1, a heuristic for Independent Set would likely not select -x1 from the mix, as it invalidates so many other potential options. This is just like how my heuristic would *not* pick -x1, favoring x1 instead, which has a higher count.

I reached out to two students who had chosen to work on Independent Set, sending them my hard and intractable problems, but they have yet to get back to me with results. If they do end up reaching out, I will record the results in the following Word document, which anyone with a link should be able to access:

Site (text is small to avoid unintentional whitespace when copy-pasting):

https://bama365-my.sharepoint.com/:w:/r/personal/ejyohannan_crimson_ua_edu/_layouts/15/Doc.aspx?sourcedoc=%7BDA29C840-E332-40F5-BE50-EA9B454343D2%7D&file=IS_and_VC_Mappings.docx&action=default&mobileredirect=true

Vertex Cover

The second mapping that I coded myself was from 3SAT to Vertex Cover. The procedure I followed came from the following presentation from McGill University:

Site: http://cgm.cs.mcgill.ca/~athens/cs507/Projects/2001/CW/npproof.html

The following is my summary of the steps:

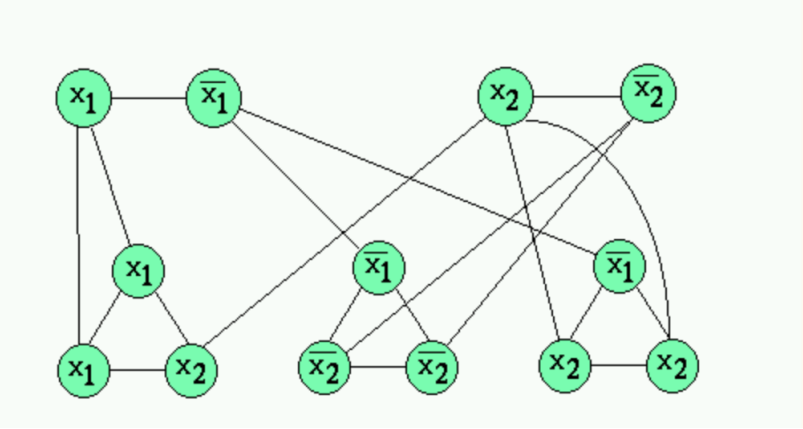
1. Consider the first clause. For each variable in the clause, create a new vertex. You can give each vertex a unique ID if you wish (since all vertices are distinct from each other), but you must label the vertex with an x-value (such as x1, x2, -x3, etc.).

   Create connections between each of these three clauses, forming an undirected triangle.
2. Repeat this process with every other clause, creating a total 3***c*** vertices, where **c** is the number of clauses.
3. Add a pair of vertices (positive and negative) for each x-value that appeared in the input, connected by a single edge. For instance, if we were working with x1 to x3, we would add (x1, -x1), (x2, -x2), and (x3, -x3).

   For each of these new vertices, connect them to all other vertices (in the triangle) that have the same value. For instance, connect the new -x1 to all instances of -x1 among the triangles.

See the example below, which I used to test and verify the functioning of my own code:

Below is a sample construction of $(x_1 \text{ V } x_1 \text{ V } x_2)(!x_1 \text{ V } !x_2 \text{ V } !x_2)(!x_1 \text{ V } x_2 \text{ V } x_2)$ and it's cover in the constructed graph which will yield values of x_1=FALSE and x_2=TRUE:



To determine whether the original 3SAT formula was satisfiable, we will have to check whether there is a vertex cover of size k = 2**c** + ℓ, where **c** is the number of clauses (3 in this case) and **l** is the number of x-variables. This makes sense, as to cover the edge between each positive-

negative pair, we need to select one from that pair (1* ℓ), and to cover each triangle of edges we need to select two vertices from that triangle (2***c**). This limitation ensures that only one vertex from each pair is chosen (otherwise a triangle won't be properly accounted for) and only two vertices from each triangle are chosen (otherwise a pair won't be properly accounted for). If such a cover can be found, the particular selections among the *pairs* will dictate what value for each x-variable should be selected in order to satisfy the original 3SAT formula. If such a cover cannot be found, then the formula is not satisfiable. This setup again makes sense from a greedy heuristic perspective, as the variable that appears more frequently will have a higher number of connections. In the case of my particular input, solving the hard problem would require a vertex cover of size 29, whereas the intractable would require one of size 486.

I reached out a student who had chosen to work on Vertex Cover, sending them my hard and intractable problems, but they have yet to get back to me with results. If they do end up reaching out, I will record the results in the following Word document, which anyone with a link should be able to access:

Site (text is small to avoid unintentional whitespace when copy-pasting):

https://bama365-my.sharepoint.com/:w:/r/personal/ejyohannan_crimson_ua_edu/_layouts/15/Doc.aspx?sourcedoc=%7BDA29C840-E332-40F5-BE50-EA9B454343D2%7D&file=IS_and_VC_Mappings.docx&action=default&mobileredirect=true

I will also add any other mappings to this document if someone else reaches out with a different type of problem.

Conclusion

I hope that all necessary information has been covered in this paper. I am certainly accessible by email if there are any questions, and I will update the provided Word document if any new information becomes available.