JOURNAL OF
EMERGING INVESTIGATORS

Article

Solving a new NP-Complete problem that resembles image pattern recognition using deep learning

Naod Abraham[1], Alan Kwok[1]
[1]Saint Mary Catholic Academy, Toronto, Ontario

SUMMARY

In computer science, non-deterministic polynomial time (NP) denotes the set of problems for which a solution, if it exists, can be checked quickly. We also call the set of problems in NP that are at least as hard as any other NP problem non-deterministic polynomial time complete (NP-complete). Although not known if they can be solved quickly as of writing this paper, NP-complete problems can be converted into one another quickly. A prominent and practical example is the protein folding problem, which is a problem of determining the shape and function of proteins. Some models of the protein folding problem have been shown to be NP-complete. In this paper, we discovered and provided a mathematical proof of a new NP-complete problem, that we named 3-Grid Pattern Decision Problem (3- GPDP). 3-GPDP is a decision problem that asks if a specific kind of pattern exists or not on a grid/matrix of numbers consisting of zeros and ones, much like image recognition. We hypothesized that a neural net would be able to solve 3-GPDP problems with high accuracy and low loss when trained. In support of our hypothesis, our experiment resulted in a neural net that performed with 84% accuracy and 0.14 loss without underfitting or overfitting. Moreover, there was also an increase in accuracy and decrease in loss with more training data. Hence, this further experimentally supported that using neural-networks can be a good heuristic approach to solve many NP-complete problems.

INTRODUCTION

A Turing machine is a mathematical model of a computing machine with a defined set of rules (1). When the Turing machine is given a set of variables as an input, it transforms it into a result or output following its predefined rules. In theory, a Turing machine can do everything that a classical computer can do. Computational complexity theory is a sub-discipline of theoretical computer science that involves placing computational problems into classes according to their resource usage on Turing machines and then tries to relate the different classes with one another (2). One of the important resources to consider is time. If we can do some computation on some input in polynomial time, it means that as the size of our input grows, then the time it takes to do the computation grows as a polynomial function of the input size (3). For example, sorting a given set of numbers from smallest to largest can be done in polynomial time. This is because there exists an algorithm that can solve the problem in a time that is proportional to n^2, where n is the number of numbers in a given set of numbers (4).

In computational complexity theory, two important classes of problems are the P class and the NP class (5). P problems are problems whose solution can be found and verified by a deterministic Turing machine in polynomial time. NP problems are problems whose solution can be verified by a deterministic Turing machine in polynomial time, and the solution can be found on a non-deterministic Turing machine in polynomial time (5). Within the class NP, there is a class called NP-complete problems, which are NP problems that are at least as hard as any other problem in the class NP (5). NP-complete problems are especially important because, if we find a method to solve a single NP-complete problem fast, then we are mathematically guaranteed that the method will also be able to solve other NP problems fast. This is because NP-complete problems can be converted into one another in polynomial time. For example, sudoku, which is just a puzzle, is NP-complete (6). Hence, even though, sudoku is not practically useful, if we can find a fast algorithm for solving nxn sized sudoku, then we can convert an important NP-complete problem into sudoku and solve the sudoku instead. If P=NP, then the two groups of problems are not separate from each other (5). It means that for every problem in NP, there exists at least one algorithm that can solve it in polynomial time on a deterministic Turing machine. If P≠NP, then the two groups of problems are fundamentally separate from each other and there exists at least one problem in NP, such that there is no algorithm that can solve it in polynomial time in a deterministic Turing machine, if one were to try every conceivable algorithm (5).

Numerous studies have been conducted to answer the P vs NP problem, but none have prevailed as of writing this paper (7). That is, we don't know if P=NP or P≠NP. The importance of solving this problem is highlighted by the fact that numerous technological problems are considered NP-complete problems. A prominent example, and perhaps a significant one to our daily lives is the protein folding problem (8). Proteins are one of the building blocks of life. Each protein type is made of a unique sequence of amino acids, linked in a long chain with covalent peptide bonds (9). There are 20 amino acids that a protein can be made up of, and the sequence of amino acids determines the 3-dimensional shape of a protein. The precise 3-dimentional shape of a protein determines its function in a cell (9). However, determining the shape of a protein based on its unique sequence of amino acids is an NP-complete problem (8). Generally, to simulate a given molecule, such as a protein, on a classical computer to find its structure, one would have to account for every quantum state the molecule can have, and the number of these states

grows exponentially the bigger the molecule (10). Hence, this presents a big hurdle to the pharmaceutical companies when it comes to making new drugs. As a result of its immense significance and numerous applications in our world, the Clay mathematical institute announced a US $1,000,000 prize for a solution to the problem in 2000 (7). Hence, The P vs. NP problem is one of the most important open question problems of our time, which is why many scientists have begun to use heuristic approaches to solve problems that fall under the umbrella of NP. In mathematics, heuristic approaches are techniques that give approximate solutions when finding solutions via classical methods are too slow or impossible (11).

One heuristic approach is deep learning. Deep learning is a subfield of machine learning that uses algorithms known as neural nets, abbreviation for neural networks, to learn patterns from data to make predictions (12). Deep learning is used when using an exact algorithm to perform a task is difficult or impossible. Neural net algorithms are fed data to predict solutions by learning from the data. Research has shown the effectiveness of using deep learning to solve NP-complete problems. For example, the Hopfield network was an effective heuristic approach for solving sudoku (13). Hopfield networks allow one to store one or more patterns, and then at a later stage, those patterns can be retrieved by just providing part of a pattern. Hopfield networks are especially good for different patten recognition problems. However, using neural networks as a heuristic approach for solving NP-complete problems could be more effective by choosing NP-complete problems that more closely resemble the type of problems that deep learning is already used for on a daily basis, such as image recognition (12).

In this paper, we created and mathematically proved a new NP-complete problem, that we named 3-Grid Pattern Decision Problem (3-GPDP). 3-GPDP is a decision problem that asks if a specific kind of pattern exists or not on a grid of numbers consisting of zeros and ones. The nature of 3-GPDP resembles an image recognition problem. Hence, since neural nets are algorithms that are especially good at learning and recognizing patterns, especially for images, we hypothesized that a neural net would be able to solve 3-GPDP problems with high accuracy and low loss when trained. Our experiment resulted in a neural net that performed with 84% accuracy and 0.14 loss without underfitting or overfitting. Moreover, there was also an increase in accuracy and decrease in loss with more training data. This supported our hypothesis. By building and training subsequent better neural net models and increasing data size, one can potentially achieve low enough training loss and high enough training accuracy, such that it starts to become applicable to other real-world important NP problems, that we otherwise have no fast algorithms for. Details on how to decrease the loss and increase the accuracy further are discussed in the discussion section.

RESULTS

We created and gave a mathematical definition of a new NP-complete problem that we named Grid Pattern Decision Problem (3-GPDP) (**Appendix A**). 3-GPDP is a decision problem that asks if a specific kind of pattern exists or not on a grid of numbers consisting of zeros and ones – essentially it is an image recognition problem. We also mathematically proved that 3-GPDP is an NP-complete problem (**Appendix B**). We then proceeded to train a neural net to solve 3-GPDP. The loss of the final training batch, final validation batch, and the testing data set was about the same (0.137 average loss), and this showed that the model was not overfitting. Additionally, 0.1397 training loss showed that the model was not underfitting either **(Figure 1A).** The accuracy of the final training batch, final validation batch, and the testing data set was about the same (84.3% average accuracy), and this again showed that the model was not overfitting. Also, 84.03% accuracy in the training data showed that the model was not underfitting either **(Figure 1B).**

Additionally, letting the neural net train on batches of 3-GPDPs instead of all the 3-GPDPs at once was crucial, as it revealed a correlation between the Neural net's performance and the amount of data it was trained on. The slope of line of best fit between training loss and number of batches of training data was negative **(Figure 2A)**. Also, the slope of line of best fit between validation loss and number of batches of validation data was negative **(Figure 2B)**. The slope of line of best fit between training accuracy and number of batches of data was positive **(Figure 2C)**. Also, the slope of line of best fit between validation accuracy and number of batches of validation data was positive **(Figure 2D)**.

DISCUSSION

3-Boolean satisfiability problem (3-SAT) is the problem of deciding if a 3-CNF formula is satisfiable or not (5). In math-

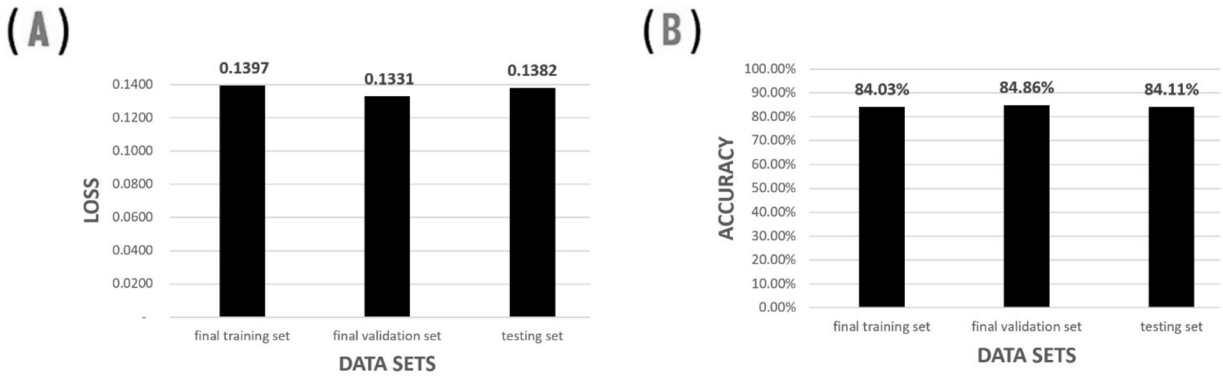

Figure 1: Neural network accuracy and loss. The loss (A) and accuracy (B) of the final training batch, final validation batch, and testing data set.

Figure 2: Correlation between loss and accuracy of the neural net and the number of batches it was trained on. Loss of the neural net and the number of batches it was trained on is correlated; (A) Training data, (B) Validation data. Accuracy of the neural net and the number of batches it was trained on is correlated; (C) Training Data, (D) Validation Data.

ematical logic, a 3-CNF is said to be satisfiable if its literals can be assigned true or false in a way that makes the 3-CNF formula true (14). 3-SAT is an NP-complete problem. Therefore, if 3-GPDP is an element of NP-complete (3-GPDP ∈ NP-complete), a deterministic Turing machine can be used to reduce 3-SAT to 3-GPDP in polynomial time, and if a 3-GPDP formula can be solved in polynomial time, then the corresponding 3-CNF formula can be solved in polynomial time. As a result, if we are able to show 3-GPDP ∈ NP-complete, we would be mathematically guaranteed that whatever "effective" method we develop to solve 3-GPDP, will also be an effective method to solve 3-SAT. The word "effective" in this context refers to being able to solve a problem in polynomial time. It was shown in theorem 1 that 3-GPDP is NP-complete. Hence, whatever effective solving method one comes up with for 3-GPDP should also be effective for 3-SAT. Another NP-complete problem is Boolean satisfiability problem (SAT) (5). Any SAT problem can be turned into a 3-SAT problem. SAT is especially important because SAT was the first NP-complete problem to be discovered, proof given by the Cook-Levin theorem (15). Many NP-complete problems have known algorithms that convert them into a SAT problem in polynomial time. Hence, the close connection of 3-GPDP to 3-SAT makes 3-GPDP of a special interest.

Additionally, in this study our neural net had the same architecture as a shallow autoencoder, and with a small ratio of middle layer to output/input layer nodes. Hence, the small ratio of middle layer to output/input layer nodes and the relatively good effectiveness of our neural net makes us ponder about the possibility of compression of the information in 3-GPDPs, and hence other NP-complete problems as well.

If P≠NP, that means that there will never be a polynomial-time algorithm to solve NP-complete problems. This poses a problem because many NP-complete problem solutions hold great value in technological companies. This issue has prompted the research and application of heuristic approaches. In this study, we hypothesize that since 3-GPDP is a problem of pattern extraction from images, a neural net would result in high accuracy and low loss when solving the 3-GPDP. Despite having a simple architecture, our neural net gave a final training loss of 0.1397 and final training accuracy of 84.03%. Although we observed great initial results, improvements will need to be made to be practically useful on a large scale. This can be acquired for future experiments by providing more training data and changing the Neural net architecture to Convolutional Neural networks (CNN) (12).

This Neural Net gave a final accuracy of 84.03% and 0.1397 final loss after training, but it initially started with a higher loss and lower accuracy **(Figure 2)**. There was a negative correlation between the training loss and number of batches of training data. Also, there was a positive correlation between the training accuracy and number of batches of training data. This was also the case with the validation loss and accuracy **(Figure 2)**. This strongly implied that with more data, the loss and accuracy can decrease and increase respectively even more in future experiments.

Another way we can increase accuracy and decrease loss is by changing the architecture of the Neural net. Some Neural net architectures are better than others in certain tasks. That is, given a certain task where a neural net must learn to make predictions based on the data its given, depending on the nature of the data, some neural network types do better than others. For example, if the task is to learn to recognize images, the preferred type of neural net architecture to use

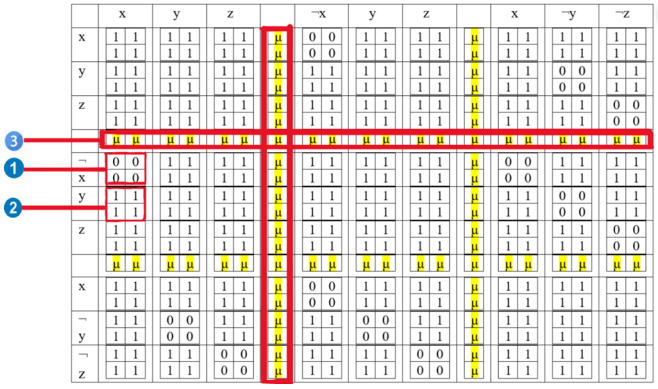

Figure 3: Enhanced version of E_2 for potential use with a CNN. 1) All zeroes in E_2 were transformed into four zeroes arranged in a 2x2 matrices. 2) All ones in E_2 were transformed into four ones arranged in a 2x2 matrices. 3) Rows and columns of 'μ's represent a separator between the row and column clauses. ½ can be substituted instead of 'μ' as it is equally as far away from both zero and one, which helps avoid bias toward either one or zero.

is a convolutional neural network (CNN) (12). Due to computational resource limit, a CNN architecture was not used in this study. However, it would be ideal for 3-GPDP given that 3-GPDP is essentially an image recognition problem. Additionally, one of the other ways higher accuracy and lower loss can be achieved in conjunction with a CNN architecture is by enhancing the training 3-GPDPs and their solutions, without affecting their core content **(Figure 3).** Changing 3-GPDPs in this way, serves to enhance the important features of 3-GPDP.

To conclude, if very low loss and very high accuracy are achieved in future experiments, then neural nets can be used as effective heuristic solvers for large enough 3-GPDP s. Note that, NP-complete problems can be converted into one another in polynomial time, much like 3-SAT and 3-GPDP in this study. Hence, we can convert another NP-complete problem into 3-GPDP via converting to 3-SAT first, solve the 3-GPDP instead and convert the 3-GPDP solution back to the solution of the given problem via the corresponding 3-SAT solution. As a result, this can have various applications when it comes to other NP problems that we otherwise have no approaches that can solve them in a reasonable amount of time.

MATERIALS AND METHODS

To test the performance of a neural net as a heuristic solving approach to 3-GPDP problems, a neural net with the same architecture of a shallow autoencoder was trained on 10000 3-GPDPs. It was also validated and tested on 2500 3-GPDPs and 2504 3-GPDPs, respectively. To create the training, validating, and testing datasets, a total of 15004 50 clause 3-CNF formulas and their solutions were generated. Once they were all generated, they were shuffled and saved. Every one of the formulas was generated specifically to have an overall diversity in terms of difficulty and satisfiability. Then, using the reduction method we discovered, they were all transformed into corresponding 3-GPDPs and 3-GPDP solutions. Finally, the neural net had to fit the 3-GPDPs to their corresponding 3-GPDP solutions. The training and validation data sets were not given to the neural net all at once; instead, the training and validation datasets were divided into ten batches. Finally, the Neural net's performance was evaluated based on the loss and accuracy of the training, validation, and testing data sets. Mean squared error and binary accuracy were used as a loss function and an accuracy metric, respectively.

Generating 3-CNF formulas and their solutions

3-CNF formulas were generated in the Discrete Mathematics and Theoretical Computer Science (DIMACS) format. The DIMACS format is a text-based file format for storing a Boolean expression in a conjunctive normal form (16). In order to generate the 3-CNF formulas, a function arbitrarily named "method_1" was written. The function took in the following inputs: the total number of 3-CNF formulas, the number of clauses in each 3-CNF, and the range of variables of the composing clauses. The function was programmed not to have duplicate literals in one clause; however, a literal and a negation of the same literal in the same clause was allowed. The function returned the requested number of 3-CNF formulas in a DIMACS format with the desired properties.

To avoid a neural net that was biased at the end of the experiment, formulas were generated to have diversity in terms of difficulty, i.e., the probability of being satisfiable. We did this based on another study that investigated the probability of satisfiability of 3-SAT problems (17). The probability of a 3-CNF formula being satisfiable depends on the clause to variable ratio. In our study all the formulas had 50 clauses, a number arbitrarily chosen, but different 3-CNF formulas were allowed to have different variable numbers. This allowed for diversity in terms of difficulty **(Figure 4).** In our data for this study, we only included 3-CNF formulas with clause to variable ratio values ranging from 2/1 to 50/7, as this range already roughly included 0% to 100% probability of being satisfiable **(Figure 4).** Given that, clause to variable ratio = clause/variable, clause = 50, and variable

$$\in \mathbb{N},$$

the feasible variables in the given range of clause-to-variable-ratio (2/1 to 50/7) were, {variable | 7 ≤ variable ≤ 25}.

Determining how many 3-CNF formula of each type (unique clause to variable ratio) was going to be generated was based on a factor of time. Once a random formula with

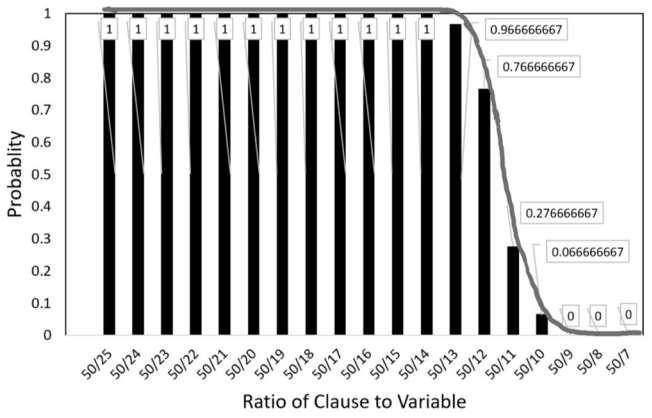

Figure 4: Probability of satisfiability of 50-clause formulas, as a function of the ratio of clauses to variables. The clause to variable ratio range in this experiment was [2/1, 50/7], and since all the formulas had 50 clauses, the variable range was [7, 25]. The line is not a part of the graph, but it helps to show that the graph is consistent with the other study (17).

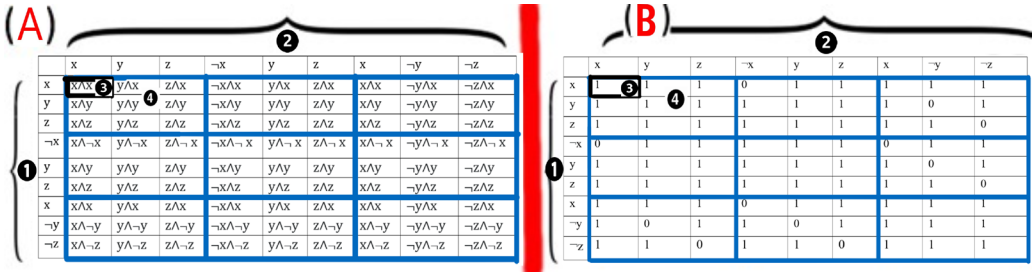

Figure 5: E_1 to E_2 reduction. (A) Intermediate step of E_1 to E_2 reduction. (B) After following the reduction steps in Figure 5A, one ends up with a 3-GPDP equivalent of E_1, i.e., E_2.

the desired properties was generated, it was solved by an online version of Minimalistic Boolean satisfiability problem solver (MiniSat) on the web (18, 19). We created another function arbitrarily named "method-2" to carry out this process. The function's job was to take a 3-CNF formula in a DIMACS format and upload it to an online version of MiniSat on the web using the selenium application programing interface (selenium API) to acquire a solution. If the formula was satisfiable, MiniSat returned which variables must be true for the whole formula to be true. If it is unsatisfiable, MiniSat returned the string, "UNSAT", which stands for unsatisfiable. After method-2 acquired a solution to the given formula, if the formula met the desired criteria in terms of satisfiability (i.e., satisfiable or unsatisfiable), the formula and its solution were sent to a TXT file to be stored for later use. If the formula didn't meet the criteria, another formula was generated by method-1, and the process looped until a formula was found that met the desired criteria.

Variables that had over 0.5 probability of being satisfiable,

$$\{\text{variable} \mid 12 \leq \text{variables} \leq 25 \text{ and variable } \in \mathbb{N}\},$$

were defined as "Type-1". Variables that had lower than 0.5 probability of being satisfiable,

$$\{\text{variable} \mid 7 \leq \text{variables} \leq 11 \mid \text{variables belongs } \mathbb{N}\},$$

were defined as "Type-2". The total training data was composed of 15004 formulas, with 7500 formulas being unsatisfiable and 7504 being satisfiable. To make the process faster, all satisfiable formulas were comprised of only Type-1 formulas, and all the unsatisfiable formulas were comprised of only Type-2 formulas. Note that there are 14 possible Type-1 variable numbers and 5 possible Type-2 variable numbers. Hence, there were 536 formulas of each Type-1 variable numbers and 1500 formulas of each Type-2 variable numbers. After all the data was generated, it was compiled, shuffled, and then saved.

3-SAT to 3-GPDP transformation

An instance of 3-SAT, $E_1 = (x \vee y \vee z) \wedge (\neg x \vee y \vee z) \wedge (x \vee \neg y \vee \neg z)$, can be transformed into an instance of 3-GPDP, E_2, using the following steps **(Figure 5A).** First, write all the literals of the formula in a column in order. Second, write all the literals of the formula in a row in order. Third, select a literal along the row and evaluate the conjunction of the literal with the literals along each column (If it is satisfiable = 1 and If it is unsatisfiable = 0). Fourth, once a grid/matrix is constructed, we can separate it into imaginary non overlapping 3x3 smaller squares.

The 3-GPDP is satisfiable if and only if there is at least one set of satisfiable 1x1 squares, one 1x1 square from each 3x3 square, that align in a way that satisfies a certain symmetry. That symmetry should be as follows. Let the x-axis point in the direction across the columns of the 3-GPDP and let the y-axis point in the direction across the rows of the 3-GPDP. Then, across a given row of 3x3 squares, one should be able to connect the 1x1 squares from that row of 3x3 squares using a straight line parallel to the x-axis. Also, across a given column of 3x3 squares, one should be able to connect the 1x1 squares from that column of 3x3 squares using a straight line parallel to the y-axis. Then, the symmetric pattern formed along the 3x3 square rows must be the same as the symmetric pattern along the 3x3 square columns. This is what makes 3-GPDP a pattern detection problem.

After following the reduction steps above, one ends up with a 3-GPDP equivalent of E_1, E_2 **(Figure 5B).** One thing to note is that, as mentioned in the Generating 3-CNF formulas and their solutions section, if a 3-CNF formula is satisfiable, its solution can be given as the literals that have to be true for the whole formula to be true. On the other hand, if it is unsatisfiable, the solution just says "UNSAT" without returning any variables. If a 3-GPDP is satisfiable, set all ones in the 3-GPDP that are not part of the pattern mentioned above to zeroes. However, if it is unsatisfiable, set all the ones in the 3-GPDP to zeroes.

Based on the transformation of 3-SAT to 3-GPDP, a valid 3-GPDP instance and a satisfiable 3-GPDP instances were given a mathematical definition (Appendix A). Also, based on the transformation of 3-SAT to 3-GPDP and the definitions, 3-GPDP was proven to be NP-complete (Appendix B).

3-SAT solution to 3-GPDP solution transformation

Once all the 3-CNFs and their solution were generated, all the 3-CNFs were reduced into corresponding 3-GPDPs. The 3-CNF solutions were transformed into their corresponding 3-GPDP solutions using a different algorithm. To show how a 3-CNF solution could be used to create a corresponding 3-GPDP solution and vise versa, we evaluated E_1 and used the solution to solve E_2 by following a simple algorithm. One of the satisfying assignments of E_1 is when x=1, y=1, z=1**.** The algorithm replaced the values of the literals in Figure 5A with the satisfying assignments of E_1, that is x=1, y=1, z=1, and then evaluated all the 1x1 squares, (True=1, False=0) **(Figure 6A-B).**

If x=1, y=1, z=1 is a solution for E_1, then $x \wedge y \wedge z = 1$ must be true. This means each literal is going to form a conjunction with the others that evaluates to true, and this looks like a repeating pattern of ones along the rows and columns **(Figure 6C).** Since there is at least one of the literals in each clause,

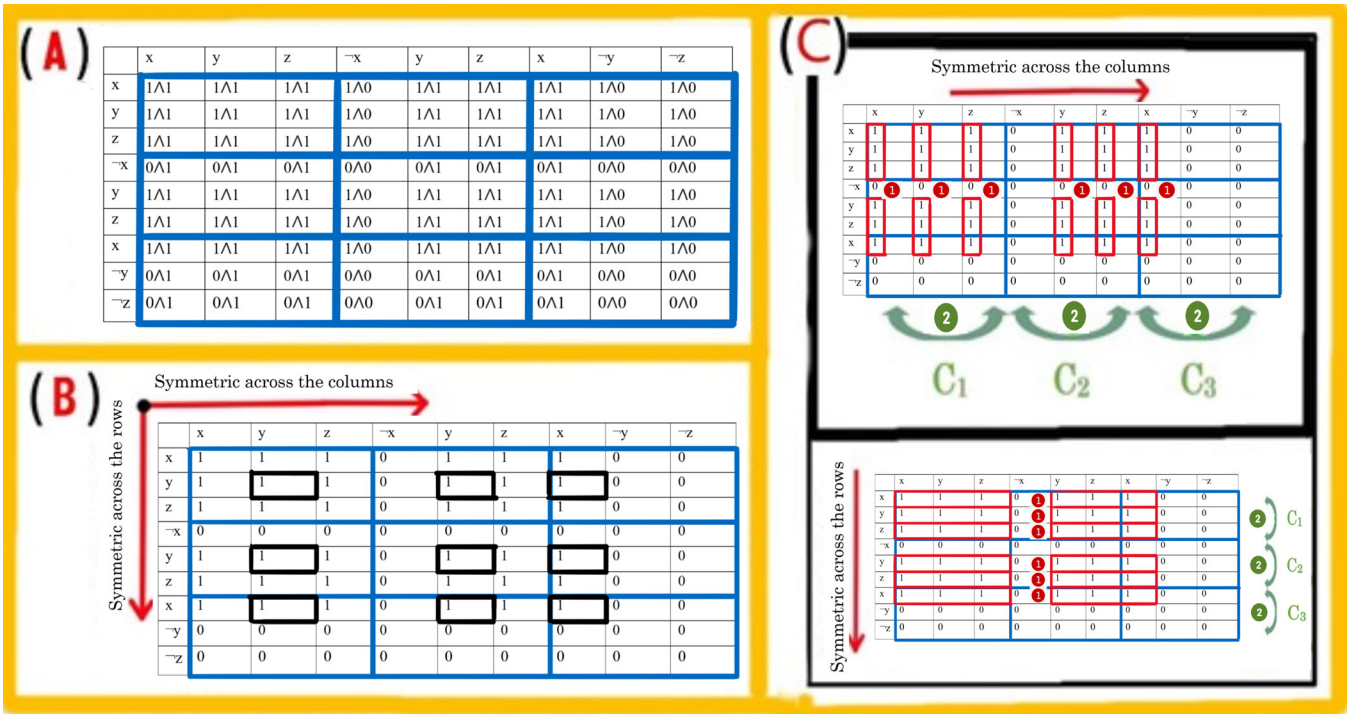

Figure 6: Creating a corresponding solution for E_2 from the solution for E_1. A) The values of literals in figure 5A replaced with the satisfying assignments of E_1, that is x=1, y=1, z=1. B) The result after all the 1x1 squares in figure 6A were evaluated. Notice that there is at least one set of 1x1 squares, at least one 1x1 square from each 3x3 square, that align in a way that is symmetric across the rows and columns. This is one of many other combinations, but since there is at least one, this 3-GPDP is satisfiable, just like its corresponding 3-CNF. C) Relationship between the solutions of E_1 and E_2. 1) Since the solution of E_1, (x=1, y=1, z=1), form a conjunction with each other that evaluates to true, this looks like a repeating pattern along the rows and columns. 2) 'C' stands for clause. There must be at least one group of trues that is part of the repeating patterns of trues mentioned in number one in each clause. This is equivalent to how there must be a least one literal from each clause in a 3-CNF that evaluates to true for the whole formula to be true.

this satisfies the second condition, i.e., one 1x1 square from each 3x3 square needs to be part of the pattern.

If one wants to prepare a solution for a 3-CNF from a corresponding 3-GPDP solution, choose one literal that is parallel to the repeating pattern of ones from each clause in the 3-GPDP solution. If the 3-GPDP solution is made up of only zeroes, then the corresponding 3-CNF is unsatisfiable.

Neural Net

All the neural net code was written using Python and the Keras API from TensorFlow in Google Collaboratory. The neural net was trained using Google Collaboratory's GPU. The architecture chosen was the same as a Shallow auto-encoder's architecture **(Figure 8A).** Like a Shallow auto-encoder's architecture, the input and output layers had the same number of nodes (22500 nodes), and there was only one middle layer with 1000 nodes. The Input and output layer had 22500 nodes because the neural net took in a flattened 50 clause GPDP and 3-GPDP solutions (150x150 = 22500). The number of nodes within the middle layer was arbitrarily chosen. The activation function, loss function, and optimizer were chosen based on their performance in autoencoders

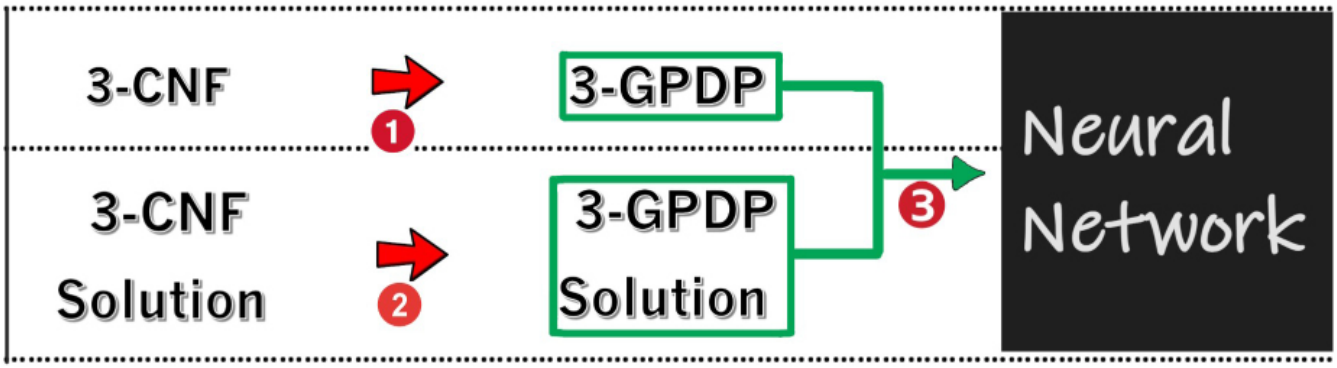

Figure 7: The process of going from 3-CNFs and their solutions to a training a neural net to solve 3-GPDPs. 1) All 15004 3-CNF formulas that were saved in a TXT file were transformed into the corresponding 3-GPDPs. 2) All the 15004 3-CNF solutions for the formulas that were saved in a TXT file were transformed into corresponding 3-GPDP solutions. 3) All the 3-GPDPs and 3-GPDP solutions were flattened from a two-dimensional array into a one-dimensional array, and the neural net had to fit the 3-GPDPs to their 3-GPDP solutions.

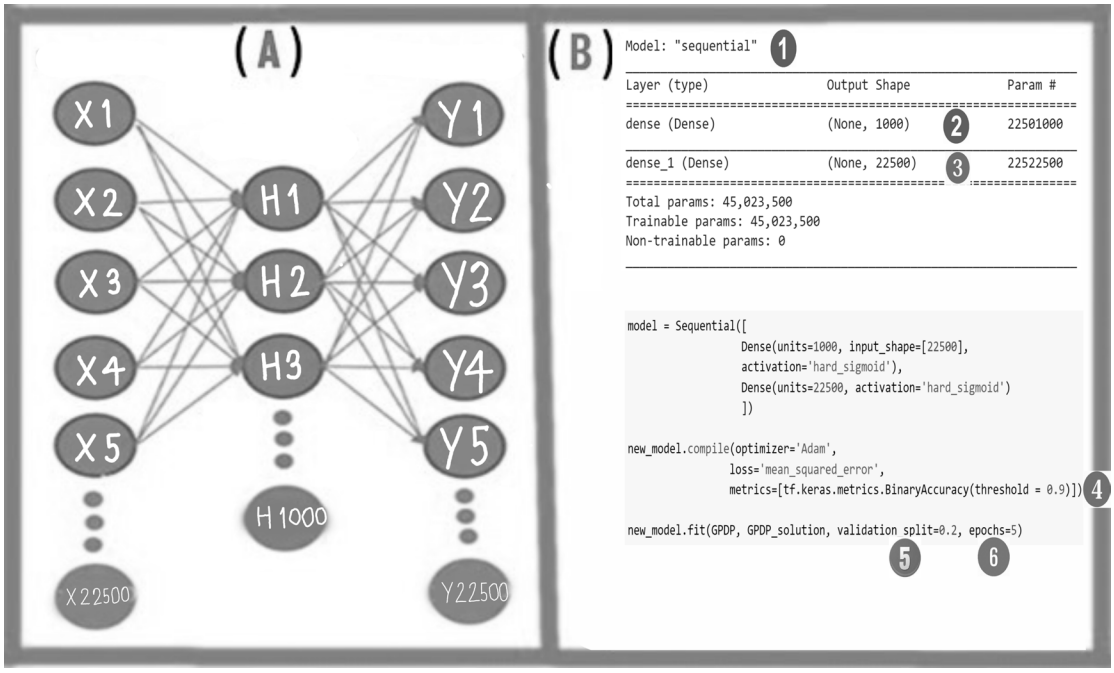

Figure 8: The neural net's architecture. A) The neural net's architecture diagram. B) The neural net's architecture details 1) The Input, Hidden, and Output were densely connected stacks of layers. 2) The Input layer had 22500 nodes and the hidden layer had 1000 nodes. 3) The same as the input layer, the output layer had 22500 nodes 4): Binary Accuracy with a threshold of 0.9 was used as an accuracy metric 5) From each training batch, the last 20% were used as a validation set. 6) The neural net was trained for five epochs every time.

(Figure 8B). Adam was used as an optimizer. Hard Sigmoid was used as an activation function, as it provides an upper and lower boundary, which is one and zero respectively, and it is almost linear which makes it faster to compute. Finally, the performance of the neural net was judged based on its loss and accuracy on the training, validation, and test data sets. Mean squared error was used as a loss function. Also, since the output nodes only contained either 1.0 or 0.0, Binary accuracy with a threshold of 0.9 was used as an accuracy metric.

Once the 3-GPDPs and their solutions were generated, the neural net was trained by trying to fit each 22500 long array of 3-GPDP with its 22500 long array corresponding solution. A common issue with neural nets is overfitting, that is simply memorizing the output of a given input, or in this case the solution of the given instance of 3-GPDP. This is why 2504 3-GPDPs and their solutions were saved to later be used as a test set to see if the neural net performs as well when given data it had never seen before. Another common issue with neural networks is underfitting, that is not being able to perform on the training set, let alone data it had never seen before. One of the ways this can be combated is by increasing the complexity of the neural net or by adding more features to the data. In this case, such features may include the ones shown in Figure 3.

To test if the neural net would improve with more data, 12500 3-GPDPs were fed into the neural net in batches of 1250 3-GPDPs at a time for a total of 10 batches during training. The batch number was chosen under the constraint of ram limit. In each batch, the first 80% or first 1000 3-GPDPs were classified as training data, and the last 20% or 250 3-GPDPs were classified as validation data.

ACKNOWLEDGMENTS

I would like to thank my high school, Saint Mary Catholic Academy. I would also like to thank the Journal of Emerging Investigators for giving me an opportunity to share my work with the world.

Received: December 12, 2020
Accepted: June 15, 2021
Published: January 10, 2023

REFERENCES

1. De Mol, Liesbeth. "Turing Machines." *Stanford Encyclopedia of Philosophy*, Stanford University, 24 Sept. 2018, plato.stanford.edu/entries/turing-machine/.
2. Dean, Walter. "Computational Complexity Theory." *Stanford Encyclopedia of Philosophy*, Stanford University, 20 July 2016, plato.stanford.edu/entries/computational-complexity/.
3. Arora, Sanjeev and Barak, Boaz. "Basic Complexity Classes." *Computational Complexity: A Modern Approach*, Cambridge Univ. Press, 2010, pp. 27–28.
4. Cormen, Thomas, et Al. "Sorting and Order Statistics." *Introduction to Algorithms*, MIT Press, 2009, pp. 150–151.
5. Carlson, James, et al. "The P versus NP Problem." *The Millennium Prize Problems*, The Clay Mathematics Institute, 2006, pp. 87–94.
6. Haythorpe, Michael. "Reducing the Generalised Sudoku Problem to the Hamiltonian Cycle Problem." *AKCE International Journal of Graphs and Combinatorics*, vol. 13, no. 3, 2016, pp. 272–282., doi:10.1016/j.akcej.2016.10.001.
7. "P Vs NP Problem." *Clay Mathematics Institute*, 2020, www.claymath.org/millennium-problems/p-vs-np-prob-

lem.
8. Guyeux, Christophe, et al. “Is Protein Folding Problem Really a NP-Complete One? First Investigations.” *Journal of Bioinformatics and Computational Biology*, vol. 12, no. 01, 2014, p. 1350017., doi:10.1142/s0219720013500170.
9. Alberts, Bruce, et al. “Proteins.” *Molecular Biology of the Cell*, Garland Science, 2015, pp. 109–110.
10. Leontica, Sebastian, et al. “Simulating Molecules on a Cloud-Based 5-Qubit IBM-Q Universal Quantum Computer.” *Communications Physics*, vol. 4, no. 1, 2 June 2021, doi:10.1038/s42005-021-00616-1.
11. Mulder, Patty. “Heuristic Method.” *Toolshero*, 4 Mar. 2022, www.toolshero.com/problem-solving/heuristic-method/.
12. Alzubaidi, Laith, et al. “Review of Deep Learning: Concepts, CNN Architectures, Challenges, Applications, Future Directions.” *Journal of Big Data*, vol. 8, no. 1, 31 Mar. 2021, doi:10.1186/s40537-021-00444-8.
13. Mladenov, Valeri, et al. *Solving Sudoku Puzzles by Using Hopfield Neural Networks*. May 2011, dl.acm.org/doi/10.5555/2001305.2001330.
14. Mendelson, Elliott. “First-Order Logic and Model Theory.” *Introduction to Mathematical Logic*, 2015, pp. 62–62.
15. Cook, Stephen A. “The Complexity of Theorem-Proving Procedures.” *Proceedings of the Third Annual ACM Symposium on Theory of Computing - STOC ‘71*, May 1971, pp. 151–158., doi:10.1145/800157.805047.
16. Maplesoft. “DIMACS CNF (.Cnf) Format.” *Maplesoft*, www.maplesoft.com/support/help/maple/view.aspx?path=Formats%2FCNF.
17. Selman, Bart, et al. “Generating Hard Satisfiability Problems.” *Artificial Intelligence*, vol. 81, no. 1-2, 1996, pp. 17–29., doi:10.1016/0004-3702(95)00045-3.
18. Eén, Niklas, and Niklas Sörensson. “The MiniSat Page.” *Minisat*, minisat.se/.
19. Galenson, Joel. “MiniSat.” *Minisat.js*, 30 Nov. 2014, jgalenson.github.io/research.js/demos/minisat.html

Appendix A

A valid 3-GPDP instance and a satisfiable 3-GPDP instance definitions

Definition 1: A valid 3-GPDP instance is defined as follows:
A valid instance of 3-GPDP is any matrix, M, that satisfies all the following four requirements.

(a) M must be an $n \times n$ square matrix, where $n \in \mathbb{N}$ and $3|n$.
(b) Let S=$\{x: x \in \mathbb{N}, 1 \leq x \leq n\}$. $\forall p \in S,\ M_{PP} = 1$
(c) $M = M^T$, that is M must be a symmetric matrix.
(d) $\forall x, y \in S,\ M_{xy} = 1$ or $M_{xy} = 0$

Definition 2: A satisfiable 3-GPDP instance is defined as follows:
Let the valid 3-GPDP be an $n \times n$ matrix, M, such that $n \in \mathbb{N}$. Also, let the 1x1 matrices that contain the number '1' be true squares, and let the 1x1 matrices that contain the number '0' be false squares. A valid 3-GPDP is satisfiable if and only if it satisfies all of the following four requirements.

(a) The 3-GPDP must be divided into $\frac{n^2}{9} \in \mathbb{N}$ 3x3 matrices that don't overlap each other.
(b) In each 3x3 matrix, there must exist at least one true 1x1 matrix.
(c) Let the x-axis point in the direction across the columns of the 3-GPDP and let the y-axis point in the direction across the rows of the 3-GPDP. There must be at least one set of 1x1 matrices, with only one 1x1 matrix from each 3x3 matrix, that evaluates to true. Additionally, <u>for each such set</u>, the 1x1 matrices must be aligned in a way that is symmetric along the rows and columns. The symmetric pattern should be as follows:
In each row of 3x3 matrices, one should be able to connect the 1x1 matrices that evaluate to true from each 3x3 matrix in that row using a straight line parallel to the x-axis. In each column of 3x3 matrices, one should be able to connect the 1x1 matrices that evaluate to true from each 3x3 matrix in that column using a straight line parallel to the y-axis.
(d) The symmetric pattern across columns created by the true 1x1 matrices in requirement (c) must be the same as the symmetric pattern created by the true 1x1 matrices in requirement (c) across the rows. That is, set every 1x1 matrix that is not part of the symmetric pattern in requirement (c) to false or '0'. Let this new matrix be W. Then $W = W^T$ must be true.

Appendix B

Proof 3-GPDP is NP-Complete

Theorem 1: 3-GPDP is NP-Complete
Proof:
(1). To show that 3-GPDP is in the class NP, lets show that a given solution is verifiable if it is a valid solution or not to a given 3-GPDP in polynomial time on a deterministic Turing machine. One of the ways a 3-GPDP solution can be represented is by leaving one 1x1 matrix from each 3x3 matrix that is part of the symmetric pattern as they are, true or '1', and setting every other 1x1 matrix to false or '0'. This way one can easily check if the remaining true matrices fit the desired symmetric pattern. If no 1x1 matrix is given from even just one 3x3 matrix, then the 3-GPDP is not a valid solution. Hence, a solution can be given by a set containing a pair of numbers, (m, n) , where m is the row number and n is the column number. Those pairs represent the location of the true 1x1 matrices in a given matrix of 3-GPDP solution. Let such set be, $S = \{(m,n)_1, (m,n)_2, (m,n)_3, \dots, (m,n)_p\}$, such that $m, n, p \in \mathbb{N}$. Also, let an arbitrary 3-GPDP whose solution is given by the set be G_1. Now, verifying if a given solution is a valid solution can be done by

executing the following two tests. The first test verifies if the solution has the desired symmetry across its columns in polynomial time on a deterministic Turing machine. The second verifies if the solution has the desired symmetry across its rows, and if the symmetry across the rows is the same as the symmetry across the columns in polynomial time on a deterministic Turing machine.

Test 1:

Step 1: Checking if G_1's solution is size compatible with G_1. Let the column number of G_1 be l. Check if $3 \times \sqrt{|S|} = l$. If it is true, then proceed to step 2, other wise this solution is not valid. This step takes polynomial time on a classical computer to execute.

Step 2: Checking if the spots of the true 1x1 matrices in G_1's solution are also true in G_1. If the spots of the true 1x1 matrices in G_1's solution are also true in G_1, then proceed to step 3, other wise this solution is not valid. This step takes polynomial time on a classical computer to execute.

Step 3: Checking if the matrices in G_1's solution lie in the same row. To verify this, group together the pair (m, n) of the true 1x1 matrices that lie in 3x3 matrices, such that those 3x3 matrices lie side by side horizontally. Hence, given a solution set, there should be $\sqrt{|S|}$ groups, and there should be $\sqrt{|S|}$ pairs in each group. Also, in each group, the pairs must be in order. That is, the pairs representing the left most 3x3 square first and the pairs representing the right most 3x3 square last. Check if all pairs have the same m value in each group. If it is true, then proceed to step 4, otherwise this solution is not valid. This step takes polynomial time on a classical computer to execute.

Step 4: Checking if the matrices in G_1's solution lie in the same column. To verify this, use the groups from step 3. Then, in each group, take only the n value of all the pairs, and put them in a new array in the same order as the pairs. Then, the a^{th} element, such that $a \in \mathbb{N}$, from all the arrays should be equal. If it is true, then proceed to step 5, otherwise this solution is not valid. This step takes polynomial time on a classical computer to execute.

If the solution set passes all four steps, then the solution has at least the desired symmetry across its columns. Since all the steps take polynomial time on a classical computer to execute, the test takes polynomial time on a deterministic Turing machine to execute.

Test 2:

Step 5: Checking symmetry along rows and columns of G_1's solution. Let R be an $n \times n$ matrix, and let the transformation of picking up R, and putting it down be equal to 1. Also, let the transformation of an 180^0 flip along the straight line connecting the top left corner and the bottom right corner of R be equal to f. Due to the way 3-GPDP and its solution were chosen to be constructed, for a 3-GPDP and a valid 3-GPDP solution, $f = 1$. As a result, any arbitrary pattern that exist across the rows also exists along the column, and vise versa, for a 3-GPDP and a valid 3-GPDP solution. Hence, this gives the following two important results. (1) If for a given 3-GPDP solution $f = 1$, and there exists a symmetric pattern across its columns, then there exists a symmetric pattern across its rows and this symmetric pattern is the same as the symmetric pattern across the columns. (2) If for a given 3-GPDP solution $f \neq 1$, then it is not a valid 3-GPDP solution. This means that one can easily verify if there exists the desired symmetry across the rows of a 3-GPDP solution, and if this symmetric pattern is the same as the symmetric pattern across the columns in one go if the solution passes test 1. That is, by just simply verifying if $f = 1$ for the given 3-

GPDP solution. Verifying if $f = 1$ for G_1's solution can be done as follows. Since G_1's solution is constructed out of S, then it is a $3\sqrt{|S|} \times 3\sqrt{|S|}$ matrix. Let's call this matrix M. Then, check if $M = M^T$. If this is true, terminate, otherwise the solution is not valid. This is a polynomial time process on a classical computer.

Passing test 1 and test 2 implies that the solution has the desired symmetry across its columns and rows, and the symmetric pattern along the rows is the same as the symmetric pattern along the columns. Which implies S is a valid solution. Otherwise, not passing test 1 or test 2 implies that S is not a valid solution. Since all the steps take polynomial time on a classical computer to execute, then verifying if a given solution is valid or not always takes polynomial time on a deterministic Turing machine.

(2). Next is to show that 3-GPDP is at least as hard as 3-SAT. To do so, let F_1 be an arbitrary size 3-CNF, so it is in the form of $F_1 = \wedge_{i=1}^{k} \ c_i$, such that $k \in N$ and c_i is a clause. This 3-CNF can be reduced to the corresponding 3-GPDP, let's call it F_2, using the following pseudocode on a classical computer. Hence, it can be done on a deterministic Turing machine as well. The 3_CNF given to the function must be an array of the literals in a 3-CNF formula in order. A literal and its negation must be written in terms of a unique positive integer, and product of that positive integer and (-1), respectively. For example, the literals in $E_1 = (x \vee y \vee z) \wedge (\neg x \vee y \vee z) \wedge (x \vee \neg y \vee \neg z)$ can be represented as:
$x = 1,\ not_x = -1,\ y = 2,\ not_y = -2,\ z = 3,\ not_z = -3$. That is, $E_1 = [1,2,3,-1,2,3,1,-2,-3]$

```
FUNCTION 3_CNF_to_3_GPDP_Converter(3_CNF)
    Declare Integer length= length of 3_CNF
 //the total clause number times 3
    Declare Integer 3_GPDP= Array [0: length, 0: length]
    For x = 0 to length
        For y=0 to length
            If 3_CNF[x]= -3_CNF[y] Then
                3_GPDP[x][y] =0
            Else
                3_GPDP[x][y] =1
            End If
        End For
    End For
```

This is a $O(n^2)$ algorithm on a classical computer, and hence the transformation of 3-SAT to 3-GPDP is a polynomial time process on a deterministic Turing machine. Now suppose the solution of F_1 is given by $S_1 = \{m_1, m_2, m_3, \dots, m_k\}$, such that m-1 represent the index number of the literals, one from each clause, that must be true in the array 3_CNF. Let a solution for F_2 be in the form described in part 1 of this proof, $S_2 = \{(m,n) \ : \ m,n \in S_1\}$. Then notice that, based on how the 3-CNF was transformed to

3-GPDP in the pseudocode above, S_2 passes test 1 and test 2. Hence, S_2 is a valid solution for F_2. To show the converse, suppose the solution of F_2 is given in the form described in part 1 of this proof, $S_3 = \{(m_1, n_1), (m_2, n_2), (m_3, n_3), \dots, (m_p, n_p)\}$, such that $m, n, p \in \mathbb{N}$ and S_3 passes test 1 and test 2. Let another set, $S_4 = \{ m : (m, n) \in S_3 \text{ and } n = n_1 \}$. If we let m-1 represent the index of the literals that must be true in F_1, then notice that S_4 is a valid solution of F_1 based on the pseudo code above. Hence, a 3-SAT is satisfiable if and only if the corresponding 3-GPDP is satisfiable. This implies that 3-GPDP is at least as hard as 3-SAT.

As a result, 3-GPDP is in the class NP and a known NP-Complete problem, 3-SAT, can be reduced to 3-GPDP in polynomial time on a deterministic Turing machine. Hence, we conclude that 3- GPDP is NP-Complete. **Q.E.D.**