Universiteit Leiden
Opleiding Informatica

Multiobjective Pattern Mining

in Bitcoin Data and Genetic Landscapes

Name:	Kin Lok Chow
Date:	31/08/2015
1st supervisor:	Michael Emmerich
2nd supervisor:	Erik Schultes

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

The Bitcoin was introduced in 2009 and is one of the most used cryptocurrencies. This lead to an increased number of Bitcoin miners, which are people who use specialized software to create new bitcoins. At the moment, Bitcoin mining programs use a brute force approach to mine Bitcoins. While the speed of mining increases rapidly, the algorithm stays the same. Nowadays it takes on average about 200 exahashes (10^18 hashes) to mine a block. The current method is to increase the amount of hashes per second. This can be achieved not only by having large Bitcoin mining groups, but also by evolving the hardware. We want to determine if there's a method to decrease the amount of hashes per block. We propose a method of searching patterns by using schemata. To obtain a schema in Bitcoins, we use the amount of leading zeros in the nonces of a Bitcoin block. We want to find the nonces with a high amount of leading zeros. With these schemata we are going to test if the amount of hashes can be reduced. Furthermore we also want to use the same method for a second application. For our second application we want to see if there are patterns in certain gene datasets. To test this approach we will use NK-landscapes. By using these patterns we want to understand the interaction between genes and how it relates to its fitness. Here we are interested in finding high gene fitness sum. In the long term, this analysis can be useful to find out if there is a possible correlation between certain genes and their contribution to diseases. With these two application we are going to test our program which will compute the support vs the length of the strings. With this data we can compare the differences for our applications.

Contents

List of Figures

Chapter 1

Introduction

In this thesis we are going to study two hard combinatorial problems and by using pattern mining we are trying to understand their structure. The first problem involves the Bitcoin mining process. The second one is the NK-landscape maximization, where we can study what happens in the transition from polynomial time solvable problems (K = 1) to NP complete problems (K = 2), and fully random landscapes (K = N-1). To search for these patterns we will use schema finding from test sets and see if there is improvement by using these new patterns.

1.0.1 Bitcoin Introduction

The Bitcoin was invented by a group/person who used the pseudonym Satoshi Nakamoto. He published in 2008 [Nak08] a paper about the Bitcoin protocol on the 'Cryptography Mailing List' and released the Bitcoin software in 2009. The Bitcoin was the first decentralized cryptocurrency (which uses a peer-to-peer network). Unlike centralized money, for example the dollar, no central person/bank will influence the value of the coin. Without a strict generation mechanism, this can lead to major inflation/deflation from one day to the other when the demand of the coin increases/decreases. To maintain a certain amount of supply, the Bitcoin uses a 'Bitcoin generation algorithm'. This means if a bit-block is mined faster, the difficulty of the next block will increase. This way they want to maintain an average of 6 blocks per hour. The max. amount of Bitcoin that can be mined is capped at 21 million Bitcoin.

1.0.2 NK landscapes

The NK-landscapes was introduced by Stuart Kauffman in 1989 [KW89] [Kau93]. Kauffman described it as a "tunably rugged" fitness landscape. Fitness landscape is a way to visualize evolutionary landscapes (like

genes). In fitness landscapes a so called fitness function is considered that maps a genome to a so-called fitness value, which indicated how well the genome is adapted to the environment. Genomes consists of single genes that may contribute to this fitness. It uses N binary numbers, whereas each binary number represents a gene. In addition, a function is defined that maps every genome to a fitness value. With the variable K you can changee the roughness of these landscapes. An advantage of the NK-landscapes is that you can easily tune the size of the landscape and the amount of bumps the model contains by changing the parameters N and K. NK-landscapes are applicable in many fields. The NK-landscapes are widely used in evolutionary biology, as biologists find them a good way to model gene interaction. Nowadays it is also used in other fields e.g. evolution of business [YJ15] or as a general model to study transitions in complex systems [MDSE15]

1.0.3 Schemata

The Holland's schema theorem was proposed by John Holland in 1975 [Hol75]. The theorem describes why the genetic algorithm(GA) works so well. By using schemata and hyperplane sampling we can visualize how the search process of GA works. GA analyzes the inheritance effect of a parent generation to the child generation. The functionality of GA will be explained later. Schemata are generally used to find similarities in certain set of strings.

1.0.4 General problem

The Bitcoin uses a double SHA256 to verify a block. SHA256 is a cryptographic hash function, so it should not be possible to obtain the input data from the hash (output data). Hash functions like this are called preimage resistant. We want to analyse how difficult it is to make some kind of preimage attack on the Bitcoin. The most common way to solve a crypto hash function is by brute forcing through all the nonces. We will try to use patterns to find a faster way to solve it. Nowadays the difficulty of a block is very high. This makes it highly unlikely that you will find a hash that will solve a block by only changing the nonce. In reality you also have to change the timestamp or the set of transactions in the Merkle root. Currently the Bitcoin miners use ASIC (application-specific integrated circuit) machines which can already mine a few TH/s, which is 10^{12} hashes per second. The nonce field only contains 10^9 hashes, so you can brute force 10^3 times through all the nonces. Nonetheless we will use nonce, because they are the most frequently used. For our pattern mining algorithm we will use schema which we described before.

For our second program we want to find out if certain genes contribute to a disease. We will test this by using NK-landscapes. NK-landscape can find local and global optima of the given test-data. While every

gene affects other genes where it is connected with, you can't just sum the fitness of every gene depending if someone has it or not. Most researchers are not necessary interested in the optima of genetic landscapes. In a paper by H.A. Orr [Orr05] he described that most people are interested in other information like average fitness vs time. However we want to check if certain gene or group of genes induce a certain disease or effect. Just like the Bitcoins we will use schemata to parse the genes that are common in high fitness set.

We have designed a program that can plot length against the support(frequency) for the given amount of bit-strings. We are interested in the points that are on the Pareto optimal schema. Like we said earlier, the schema with lower-order length strings have a higher chance to develop well, therefore we want to minimize the length. But we also want reliable schemata, so we maximize the support. These points on the Pareto efficiency front will be stored in a file. This file will contain of pseudo-binary schema strings.

To obtain these data for our plot, we parse the data in binary string. For our Bitcoin case these data would be the nonces which we parse from a Bitcoin miner [Sam14]. We only parse the nonces which can hold a certain given threshold. The best gene existence combination will be parsed from the NK-landscape program [MDSE15]. For the Bitcoin problem we also have to input the given schema in our Bitcoin miner. Here we take the nonce output again but this time we compare it with the brute force algorithm.

1.1 Related work

Many researchers have attempted to speed the process of mining blocks. Examples of these are by Naik [CGN13] who optimized the hashing process by making small tweaks in the SHA-256 mining process. He claimed to have reduced the double SHA-256 to 1.8624 times. Another one is by Heuser [Heu13] who replaced the brute force algorithm by a SAT solver. It uses the SAT solver to find a good nonce. While both in theory have some improvement compared to the actual brute force cases, they do not seem to be used by actual miners. This possibly is because the mining community finds it too big of a hassle to actually implement it. Also because of the ASIC machines, setting the SAT solver up can be slower than the current method. Lastly there is also the method used by Samwel [Sam14] which used Genetic algorithm and simulating annealing to find a nonce. One of my tests is also an extension of this case.

Chapter 2

Definitions

2.1 Bitcoin transaction verification

We earlier described that Bitcoin is a decentralized cryptocurrency, so that means the money is non-existent. Unlike normal currency, we can not just physically give out the money to the other person and verify it by observation. So to prevent people from double spending money, they had to decide on a method to intervene with this problem. For the Bitcoin they chose a method where roughly said, each transaction is stored in a Bitcoin block. So first of all, how does a transaction take place? Each transaction contains an input and an output part. The input part consists of a list of all the output references and a ECDSA(Elliptic Curve Digital Signature Algorithm) signatures. This signature proves that the sender owns this money and has the right to spend it. For the output reference, I will first explain what the output contains. The output lists all the receiver addresses and the amount each one will get. If the output is higher than the input, then this remainder will be used as transaction fee for the Bitcoin miners. Miners aren't obligated to include "the new" transaction in a block, so blocks with a transaction fee are more likely to be included. The reason that the block without a fee eventually will get included, is because there sometimes is no solution without those.

The validity of the transaction will then be checked by full nodes. Full node users have the complete blockchain stored on their hardware to check if the transaction passes certain criteria. E.g. they have to check if the signature is genuine and if the input matches an unspent output transaction from the chain. If it satisfies all criteria, the transaction will be propagated to the miners. They can notice any form of double spending, because they can check the transaction involved with it.

Transactions that are satisfied by multiple nodes will be 'thrown' in a pool where the miners can include them in a block. These raw transaction data will be double hashed using SHA256, which will result in a 64 character string. Every Bitcoin block has a variable called the Merkle root in their header. You can obtain this

variable by first appending all the hashed transaction data pairwise. If you have an odd amount of variables on that level, you append the last one with itself. All the appended variables will be double hashed using SHA256 and this process will continue recursively until we have one variable left. This whole structure is called the Merkle tree. After they have the Merkle root, they can hash the entire block with double SHA256 again, till it meets the target difficulty. The nonce can be incremented each time, which will lead in a different hash every time. When someone finds a hash below the threshold, it will broadcast it to the whole network and they can start on the next block. But first the miners have to remove the transaction that were included in the broadcasted block from their pool. This way a transaction won't be added in multiple blocks. It is possible that a block gets dropped from the blockchain. This can happen when 2 blocks are mined at the same time. Some miners will receive the block that was mined earlier, others might receive a different one. There will be some kind of fork in the blockchain. Depending on which one will become longer first, that one will likely be continued with. The other one will be voided and no Bitcoins will be obtained by the people that mined them. These voided blocks are called extinct blocks or stale blocks. Blocks getting extinct happen on a daily base, but the chances of double spending on a normal occurrence like this are pretty slim. Most people use a Bitcoin wallet, which won't allow action which don't follow the protocol. Secondly, for extra safety measurement, people wait for certain amount of confirmations before they acknowledge a payment. So if they wait for a 6-confirmation, that means that they will wait until there are 6 blocks trailing behind the block with the transaction. Most people find it safe enough then. Satoshi described in his paper that there is only a risk of 0.1% of double spending with a 6-confirmation. A problem that it does not fix is the infamous 51%-attack. It is basically when someone or some group has more than 50% of the total hash power. So the whole process of the miners can be chosen by these attackers. This means that transactions are not safe anymore against double spending. They can choose which blocks they will drop and which transaction they will include. The purpose of a 'puzzle' like this, is to prevent spammers and denial of service attack on the network. Also it regulates the amount of Bitcoin that will circulate.

When a miner finds a block, the full node users will check if the new found block follows the criteria. There is also a second variant of nodes, which are the Simplified Payment Verification (SPV) nodes. SPV nodes do not store the complete blockchain, but only the headers. So they don't have information about the actual transactions in the blocks. They have to obtain this information from full nodes users. They will only check the amount of blocks added after the block for the likeliness that the block is valid. It costs less space and power than full nodes, but SPV can not check if a transaction is double spent. Because SPV nodes request specific transaction from their peers that can reveal the address they possess. To secure this they introduced the Bloom filters for SPV users. These filters let SPV users retrieve a subset of the transaction depending on given search specification.

2.2 Basic principle NK-landscape

Like earlier said in the introduction, the NK-landscape can be used in several occasions. We will explain how it is used with genes. The NK-landscape contains a positive amount genes N and 0 to $N-1$ epistatic genes K, which is the amount of genes interaction each gene has. Each of the genes will have 2^{K+1} fitness values, where fitness is $0 \leq x \leq 1$. The reason for that amount is that for every gene also depends on all the other genes who can influence it and every gene also interact with itself at default. The epistemic genes are commonly chosen between 2 methods, the first one is by using an adjacent neighborhood method. This method connects each gene to it closest K genes. The genes are circular connected, so gene $N-1$ is connected to gene 0. The other one is the random neighborhood. Like the name states, each gene influence K other random chosen genes. Every gene has an allele a, which can be the value 0 or 1 and basically means if the gene exist/activated. It then has 2^N different genotypes(allele combinations) where the total fitness value for the genotype can be obtained by adding the appropriate fitness value of each gene.

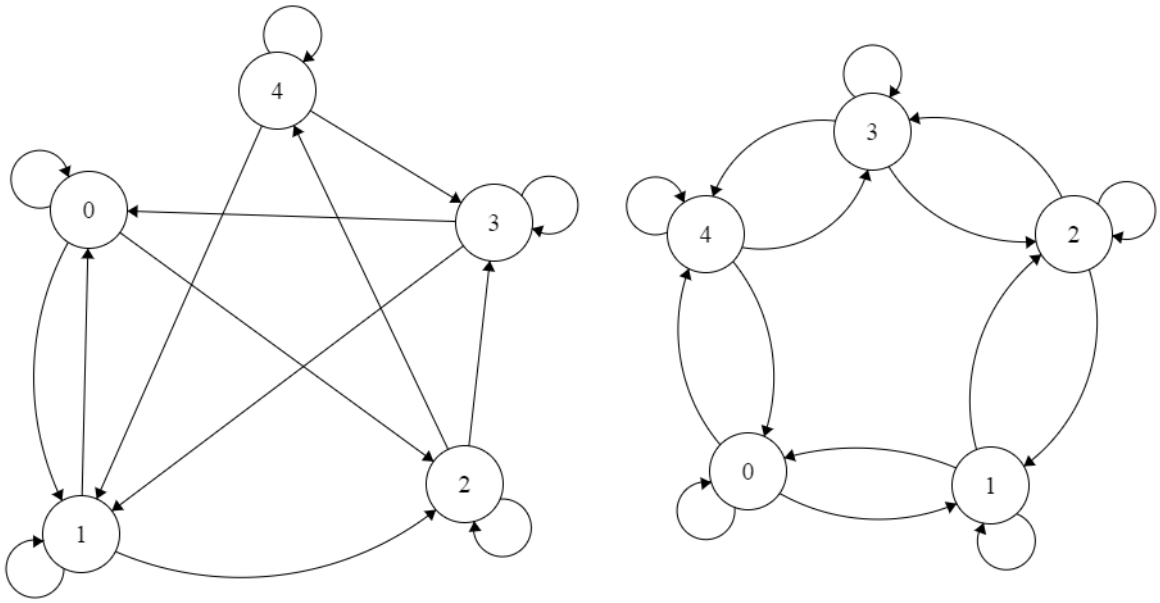


Figure 2.1: Example NK-landscape. Left: Random neighborhood. Right: Adjacent neighborhood

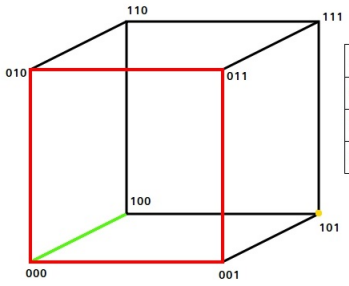
An example of how the landscape N=5 K=2 can be connected is shown in figure 2.1. On the left side we have an example of a random neighborhood interaction could look like and on the right side we have an example with adjacent neighborhood. When we describe the alleles of the landscape we can say 10000, which then means that only allele 0 is 'activated'(or depending what the landscape describes). When looking at the random example you can see that allele 0 has influence on the fitness value of alleles 0,1 and 3.

Weinberger [Wei96] and Wright [WTZ00] explained in their papers that for certain K the time complexity for computing the global maximum of a NK-landscape would change. They divided it in 3 different cases. The first is when K = 0, this means that there is no interaction between any gene. All genes are independent

of each other so we can easily find the optima's by taking the optimal points for each gene individually. So it can be solved in linear time. K = 1 becomes harder while there is now some interaction between the genes. The complexity will increase for each K, because the fitness will change for each genes epistatic gene influencing it. So when at K = 1 the global optimum can still be found in polynomial time, but for K $\geq$ 2 it becomes NP-complete. For K = N-1 the problem is fully random and therefore finding the global optimum has provably exponential time complexity.

2.3 Schema finding

A schema string works with binary numbers, so you have the standard numbers 0 and 1. But on top of that you have a *-symbol. The *-symbol means that we do not care about the value, so it can either be an one or zero. Each schema has potentially $3^L - 1$ (L stands for length of the string) different combinations which each represents a different set of strings. Because a string with only *'s is unnecessary, we ignore that combination. All these different subsets of schemata are also called hyperplanes. In figure 2.2 we have a small example of a 3-bit schema illustrated as a cube (a 3d dimension). Each axis represent a different bit position and the amount of *'s equals the dimension of the valid result. You can see the red square equals all possible numbers of the schema 0**. Two frequently used functions in schema are: the defining length and the order. The distance between the left most number and the right most number in a string is called the defining length($\Delta(H)$). The amount of bits in the string is called the order($o(H)$). So the schema: 1**1* has a length of 3 and order 2. More examples can be seen in figure 2.2.



Schema	Defining length	Order
0**	0	1
*00	1	2
101	2	3

Figure 2.2: Example schema in 2D hypercube

So why does schemata work so well for GA like we earlier explained in the introduction. For this we will first give a small explanation about the GA GA's are a subclass of evolutionary algorithm. It is a search algorithm that uses methods which are seen in natural selection. They are used to find optima of functions while they

can overcome local optima's. There are many different methods and optimizations on GA, but the basic ones mostly contain the following steps. They first initialize a population which contain the individuals of the first generation. Each of these individuals contain a possible random generated solution. Each generation will then execute the following steps:

- first we have to evaluate each of the individuals, they will all receive a fitness value depending on the goal of the GA. These goals can vary for every occasion, some want to find the fastest point from A to B for thte multi-objective shortest path searching, others want to design the best car while still meeting the requirements.

- When each of these individual got evaluated, it selects a few of the best individuals. With these individuals we build the next generation.

- Finally we use crossover and mutation on these chosen individuals. Crossover creates a new individual by taking sections from two or more of the previous selected individuals. For mutation we continue with the newly obtained individuals and modify genes or switch a pair of genes.

Each generation it will run all the steps again and this will continue till it reaches the termination criteria. This criteria is usually when a certain goal is reached or has run a given amount of generations. These termination criteria have to be set, because GA's cannot know when they found the global optima of the problem.

Holland's theorem gives the lower bound of the amount of schemata that for the next generation will be expected. The theorem makes use of the building block hypothesis which states that schema with a short defining length and low order strings with an above average fitness will likely exponentially grow in population. The reason for this is that these type of strings are less likely to cause a disrupt at crossover or mutation. Disruption means that the individual doesn't fit in the given schema anymore. Generations after these aren't bound to this lower bound and should be calculated by using the new current generation.

$$P(H,t+1) \geq P(H,t)\frac{f(H,T)}{\bar{f}}[1-p_c\frac{\Delta(H)}{L-1}(1-P(H,t)\frac{f(H,t)}{\bar{f}})](1-p_m)^{o(H)} \tag{2.1}$$

P(H,t) and P(H,t+1) are the set of all the solution of schema H at generation t and the next generation t+1. f(H,t) is the mean/average fitness of the given schema H and generation t. $\bar{f}$ is the mean/average of the whole population. So $P(H,t)\frac{f(H,T)}{\bar{f}}$ is used to find the above average cases. $[1-p_c\frac{\Delta(H)}{L-1}(1-P(H,t)\frac{f(H,t)}{\bar{f}})]$ is the crossover part, where $\frac{\Delta(H)}{L-1}(1-P(H,t)$ gives the bound for the disruption. The second $\frac{f(H,t)}{\bar{f}}$ is for the other parent that you chose for crossover. We want to know what the chance is that it survives the crossover, so we take $1-$crossover part. Lastly you have the mutation part: $(1-p_m)^{o(H)}$. This depends on the order of the schema H and just as crossover the probability of mutation. [RBK11]

2.4 Establishing the Pareto front

To distinguish the best points from our test sets we use GA. We use a non-dominated sorting GA called NSGA-II(Non-dominated sorting genetic Algorithm) [DAPMoo] which is an algorithm for multi-objective optimization. That means that the algorithm searches for all the points that are not dominated by any other points. A point is a non-dominated point if there is no other point that is better in at least one aspect and better or equal as good in the other aspects. If we gather all the points that satisfies these criteria we have all the points on the Pareto front. NSGA-II differs from other GA's in a few ways. First it uses a tournament selection which is an elitist selection. Elitist means that you keep the best individuals of the population for the next generation. The tournament selection makes use of ranks, the ranks are used to keep the better individual near/on the Pareto front. These points have a higher chance to develop in the next generation to exceed the previous generation Pareto front points. It does a tournament based on the ranks. If both are the same rank, then they use the crowding distance of the two points. The crowding distance of a point is the measure of the difference in the objective values of the two nearest neighbors. The point with a higher average crowding point will be favored over the latter. Higher crowding distance means more diverse points. Combining these two function will make sure that with elitism the population will increase in fitness, but with it will not converse to a small diversity of points because of crowding distance.

Chapter 3

Experiment

3.1 Experiment description

We are going to separate the two experiments and start with the Bitcoin experiment. The goal of our first experiment is to see if there are schemata in bitcoin data that can be exploited to solve the Bitcoin faster than current methods. For this case we will compare the amount of leading zeros in little endian of the blocks. We will experiment if brute force nonces vs nonces chosen by GA+schema performs better. To do this we can compare which method can find more nonces with higher amount of leading zeros. Which we can visualize by comparing them in a single graph. The objectives of the graph are the amount of leading zeroes of the nonce vs the amount of nonces supporting that amount. The brute force is the method that miners now use. For the one with GA, we will first randomly choose nonces. Then run it on the first 10 Bitcoin blocks(#0 to #9) of the blockchain. On each of these blocks we will run 5.000.000 iterations. Each of these 5.000.000 will check a random nonce, of which we in the end only store those containing at least 6 leading zero's. This number can vary from 0 to $2^{32} - 1$(32 bit). Realistically you also have to change other parameters in the block header to find a solution, but while we are only experimenting on the nonces we use the same block header as in the current blockchain. If there are redundant nonces we remove them. To create the set of schemata we use all the nonces we collected. To find these schemata fast we use the NSGA-II algorithm. Our NSGA-II algorithm uses 2 objectives, the defined length of the schema and the amount of nonces supporting this schema. Because we do not want to have schemata that are too specific or not specific enough, we have to remove the beginning and the end of the Pareto front (so the point with only *'s or numbers). We only use the mutation and selection function for our GA with a mutation rate of 1/string length. We run each given set 5000 generations with 40 individuals in the population. While it does not necessary need this many, we wanted to assure that we obtained the best schemata. Lastly we can use the schema to hash a block. We use

an unused block #10 for that. It will run 5.000.000 iterations again and each of the schema will get an equal proportion of the amount of iterations. Each of the schemata will be converted to their possible strings. If a schema cannot comply to the amount of iterations it has received, we give it all possible strings and remove the other proportion. These hashes we are going to compare with the brute force algorithm. We give the brute force the same amount of iterations as the schema did, so 5.000.000− the removed iterations.

For our second experiment we are using the NK-landscapes. Here our goal is to visualize the difference between K = 0,1 and 2 and N-1, where we are the most interested in the change from polynomial time solvable to NP-complete which occurs at the transition from K = 1 to K = 2. We do this by plotting them all together in a single graph. To obtain this data we first have to run all our test cases on the NK-landscape program [MDSE15]. We test for all the different N's and K's which will give us 2^N different genotypes. Out of those combinations we only take the best 20% genotypes of each landscape. The best genotypes in our experiment are the ones with the highest fitness values. Just like the Bitcoin experiment, we then use the NSGA-II algorithm on them. But unlike the other experiment we are not interested in what schema we get, but we only look at the length and support of it. So we also take the whole Pareto front for it and we plot them in a single graph to compare them. The plot names in the legend the R stands for Random and A for Adjacent. The first number is N, the second is K and eventual third number is a different seed of the parameters.

3.2 Results

3.2.1 Bitcoin

We can see in the graph of figure 3.1 the difference between the incremental(brute force) method and the schema method. In total we tested each method 3608318 times on their nonces. The result of the experiment shows that between incremental method and the schema method there are minor differences. So it is likely that by using schemata there is not a difference for mining Bitcoins. It can even be said in that case that it performs worse, because of the extra time it needs to build the schemata. Also because schema are independent of others, there are likely many duplicate nonces hashed.

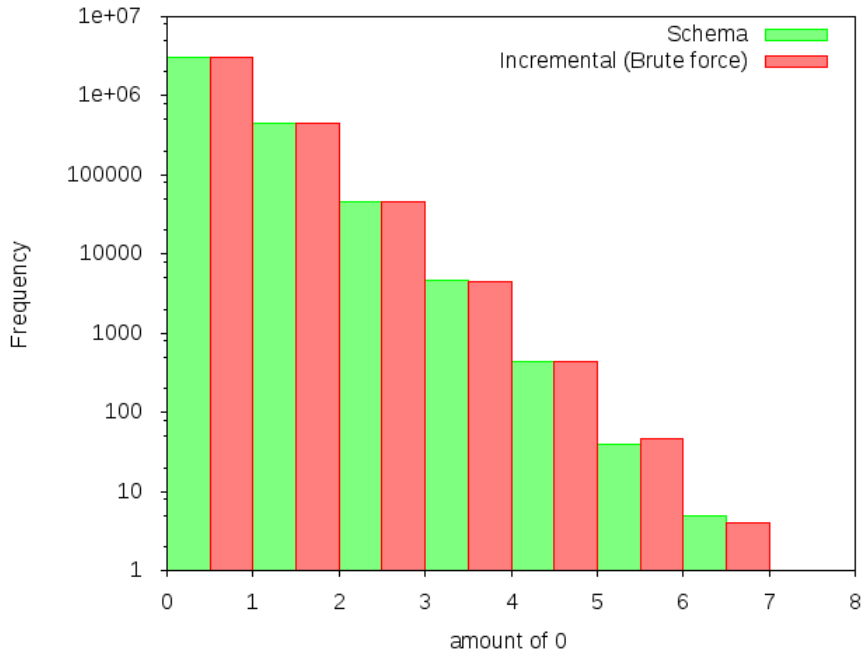


Figure 3.1: Bitcoin nonce amount leading zeros

3.2.2 NK-landscape

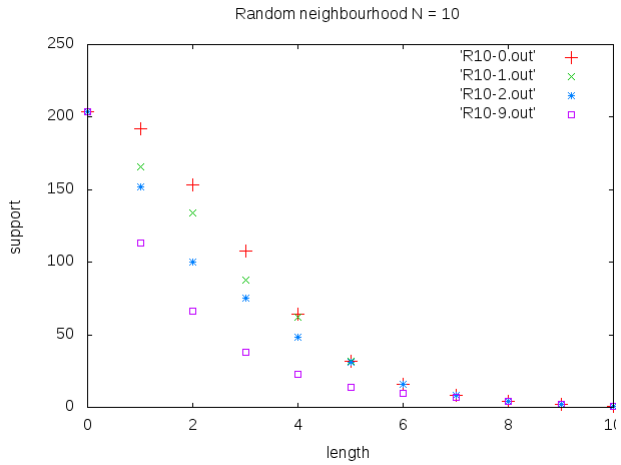


Figure 3.2: Random neighborhood N = 10

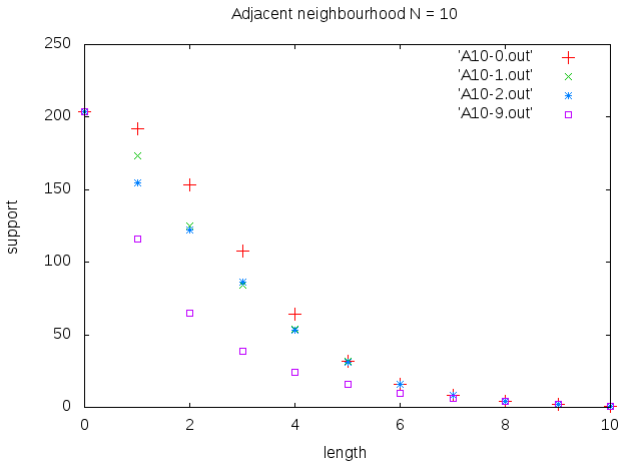


Figure 3.3: Adjacent neighborhood N = 10

In figure 3.2 we have an example of a random neighborhood landscape for N = 10. The four lines represent K = 0,1,2 and 9 (N-1). Each of these lines are the Pareto front of for their parameters. This figure shows that the lower K value have more support than high K value. When there is less interaction between the genes there is more likely a gene or gene combination which has a huge influence on the fitness. When comparing K = 0 to K = 1, the support for K = 0 is significantly higher than K = 1 for length 1,2 and 3. At length 4, K = 0 makes a rapid decrease in schemata and becomes almost equal to K = 1. This rapid decrease is not that weird, because the max. possible amount of support for length 4 is 64 ($2^{(MAX\_LENGTH)-(length)}$). This is also why everyone point at length 0 is the same, but after that will split up in their own curve. The max support

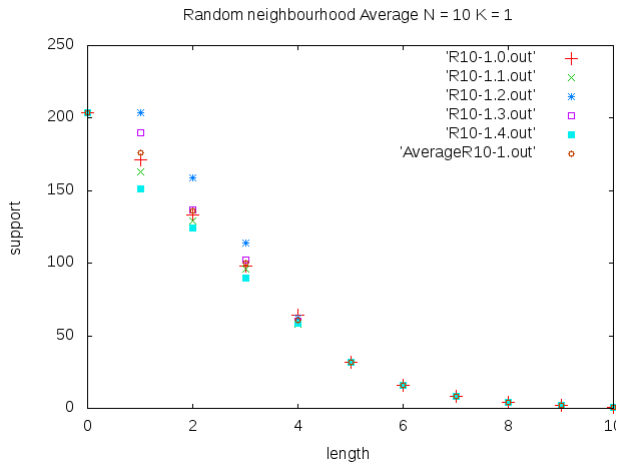


Figure 3.4: Random neighborhood N = 10 K = 1

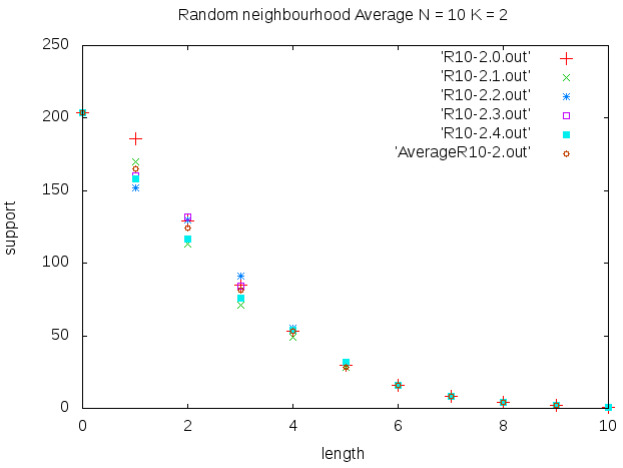


Figure 3.5: Random neighborhood N = 10 K = 2

is on average reached for K = 0 and 1 at length 5, for K = 2 at length 6 and for K = 9 it is at length 8. So at the higher length region the support of schemata for higher K can still differentiate from the max. The amount of schemata slowly becomes less which also is a sign that it slowly becomes random. This randomness is also very good visible at K = 9, you can see that each point is close at the minimum values. For example for K = 1 the amount of support is around 115, where the minimum for length 1 is 102 (50% of the 204 input points). Although at figure 3.2 it seems that lower K will never be on top of a higher K, but when you look at figure 3.4 and 3.5 you can see that the length can greatly vary between a wide range. These ranges can also overlap with each other. For each of these two figures we have plotted 5 different of their respective N and K values but with different seeds. The orange dots are the average of the 5 NK-landscape point for their objectives. At figure 3.4 you can see for length 1, one landscape still has a schema who is supported by all the input points. It is hypothetically possible that for a landscape where the interaction between the genes does not influence the fitness value of the gene by much and one gene has a high fitness contribution compared to the rest. We would get a landscape for K = 1 or higher, where the graph just shows max possible support for length 1. This could even be extended for every gene in the landscape. While this is unlikely, it does show why it is possible that higher K's have more schemata than lower K's. When comparing the average of K = 0,1,2 at figure 3.6 the values seems to become more reliable and the statement made earlier still holds. When comparing to adjacent neighborhood figure 3.3 there does not look like there is much differences, although there is some difference for K = 0. But that could be caused by the amount of different landscapes we used. At larger test sets which we obtain by increasing N, the error margin still is in effect. In figure 3.7 we have an example of a bigger NK-landscape with N = 16. We have a total of 13107 different genotypes for each K. for length 1 and 2 there is a better supported pattern for K = 2 than K = 1. Which makes it appear that having more data points does not make it more accurate.

So overall there is a certain difference between the amount of schemata for each K, but the amount are too fluctuating to separate the different K's that are close to each other. Although when comparing K = 1 and 2

to K = 9 it is pretty safe to say you can distinguish the difference between these transitions, because the curve of the amount of schemata of K = 9 is way lower than the other. In only rare cases this does not hold.

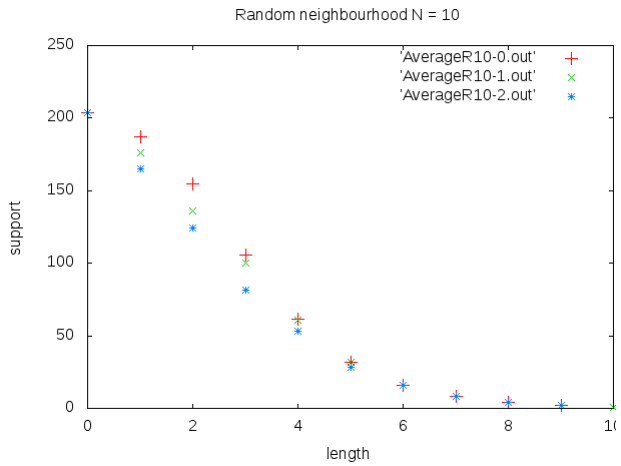


Figure 3.6: Average random neighborhood N = 10

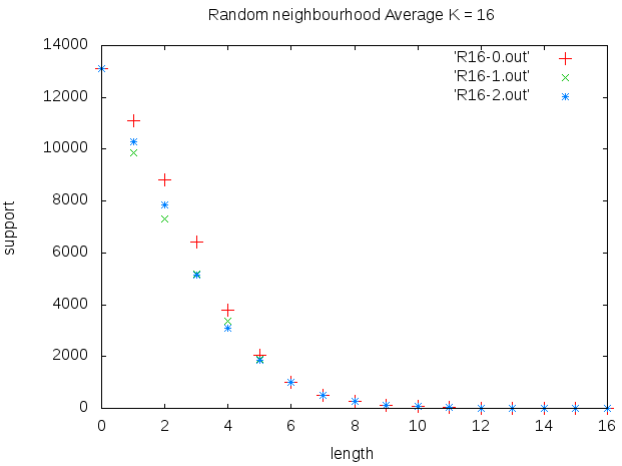


Figure 3.7: Random neighborhood N = 16

Chapter 4

Conclusions and future work

In this thesis a program was created that computes the trade-off between the support and the length of the given input. With the help of the NSGA-II algorithm we were are able to find patterns for our schema. In our first test case, that is finding patterns in high performing nonces for Bitcoins, we came to the conclusion that by using schemata we do not gain any advantage over brute force mining. It even had some disadvantage, because of duplicate answers. This result implied that it is not possible to crack the SHA-256 algorithm by using the schema method.

For our second test, which was on finding patterns in high fitness genotypes of NK landscapes, we managed to plot the difference between K = 0,1,2 and N-1. We could see in our graphs that there was definitely noticeable differences in the transition from polynomial solvable problems(N = 1) and NP-complete problems(N = 2). It seems that the lower the K is, the more cohesion between genes there are. However there are no clear borders between the amount of support for polynomial solvable to NP-complete problems, because lower K sometimes perform worse than higher K. So there is not always a clear difference in the amount of schemata for the transition. But when averaging the NK-landscapes from multiple examples there does seem to be a clearer distinction between support and length for each K. The biggest visible difference is between NP-complete problems and fully random problems(N - 1). Where the curve for the amount of schemata is way lower for fully random. By using a larger size test set (K=16) we were also not able to make distinguishes between the different transitions, as we for example saw for K = 1 and 2 in figure 3.7. Random neighborhood and adjacent neighborhood did not seem to have much differences, because the graphs were pretty identical(figure 3.2 and 3.3).

With the two test cases we have been able to test our program. The program is able to make the Pareto front for the multi-objectives of our input data.

For future work we want test how well it works on real test sets. We want would like to know if we can

actually point at certain genes. The most interesting is when $K \leq 2$, because it then is NP-complete. For the Bitcoin we could also play with other parameters, but likely would not result in any positive change. The best bet is looking at the intern mechanism of SHA-256 or more specifically for the Bitcoin version.

Bibliography

[CGN13] Nicolas T. Courtois, Marek Grajek, and Rahul Naik, *The unreasonable fundamental incertitudes behind bitcoin mining*, CoRR **abs/1310.7935** (2013), http://arxiv.org/abs/1310.7935.

[DAPM00] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and Tanaka Meyarivan, *A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii*, Lecture notes in computer science **1917** (2000), 849–858.

[Heu13] Jonathan Heusser, *Sat solving - an alternative to brute force bitcoin mining*, `http://jheusser.github.io/2013/02/03/satcoin.html`.

[Hol75] John H Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.*, U Michigan Press, 1975.

[Kau93] Stuart A. Kauffman, *The origins of order: Self-organization and selection in evolution*, Oxford university press, 1993.

[KW89] Stuart A Kauffman and Edward D Weinberger, *The nk model of rugged fitness landscapes and its application to maturation of the immune response*, Journal of theoretical biology **141** (1989), no. 2, 211–245.

[MDSE15] A. Maulana, A. H. Deutz, E. Schultes, and Emmerich, *M. t. m.: Community detection in nk landscapes - an empirical study of complexity transitions in interactive networks*, EVOLVE - A Bridge between Probability, Set Oriented Numerics and Evolutionary Computation, CD-ROM Proceedings of International Conference, June 18-24, 2015, Iasi, Romania, pp. (CD-ROM) (2015).

[Nak08] Satoshi Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*, Consulted **1** (2008), no. 2012, 28.

[Orr05] H Allen Orr, *The genetic theory of adaptation: a brief history*, Nature Reviews Genetics **6** (2005), no. 2, 119–127.

[RBK11] Grzegorz Rozenberg, Thomas Bäck, and Joost N Kok, *Handbook of natural computing*, Springer Publishing Company, Incorporated, 2011.

[Sam14] Niels Samwel, *Mining bitcoins with natural computing algorithms*, `http://www.liacs.nl/assets/Bachelorscripties/Inf-Studiejaar-2013-2014/2013-2014NielsSamwel.pdf`, 2014.

[Wei96] Edward D Weinberger, *Np completeness of kauffman's nk model, a tuneable rugged fitness landscape*, Tech. report, 1996.

[WTZ00] Alden H Wright, Richard K Thompson, and Jian Zhang, *The computational complexity of nk fitness functions*, Evolutionary Computation, IEEE Transactions on **4** (2000), no. 4, 373–379.

[YJ15] Zhe Yuan and Guoyin Jiang, *Evolution of service innovation in e–commerce based on the nk model*, International Journal of Simulation and Process Modelling **10** (2015), no. 1, 80–88.