Ragy Haddad
CS 7641
Assignment 2

Randomized Optimization

Content/Map:

Data Sources and Data Explanation:

Data Source:

The dataset used is for classifying cyber security attacks by analyzing packet capture data from tcpdump, similar to data you would collect using Wireshark or any packet capturing software for penetration testing and debugging networks.

http://www.kdd.org/kdd-cup/view/kdd-cup-1999/Data

Feature Explanation And Pre-Processing for Dataset I:

The dataset contains packet data and a label of the attack type the packet data corresponds to. So a row in the dataset looks like the following table:

Total number of features: **39** + **general label.**

Total number of rows: **300,000**

Noise: **The dataset includes some noise where the same features occasionally correspond to different labels.**

F1	F2	Fn	Label

Handling unbalanced data:

Attack Class Instances : 250436

Normal Class Instances: 60593

Based on the current ratio if we were to guess that every connection is an attack we would land 80% accuracy which doesn't mean our model is **80% percent accurate**. So using this distribution with shuffling the set . If we score higher than 80% and make sure we are getting good precision and recall it means our model is significantly improving. We could also upsample the Normal class and see how that affects our model. This would help us reinforce the signal of the normal class.

Implementation Method:

To implement the different set of algorithms I created a neural network in Python using the following tutorial https://enlight.nyc/projects/neural-network/ . This gives me freedom to tweak the Neural Network weights as I wish.

Neural Network Weights Optimization

Note the fitness function addressed in the following cases is our loss function

Random Hill Climbing:

The random hill climbing algorithm was done by initiating the neural network with random weights, after initiating a random weight we try to minimize the loss function until we converge, once we do that we reset the weights back to random and start with another set of random weights we also optimize those weights till conversion.

Neural Network Architecture:

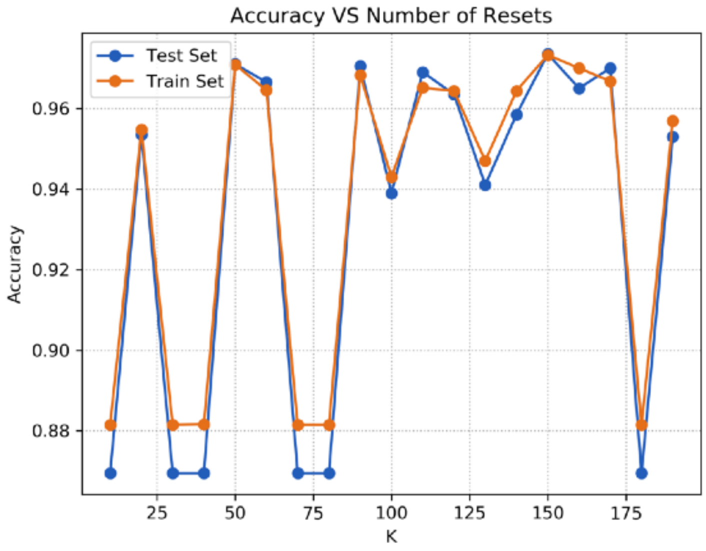
Hidden layers: 2 (Same as Network done in Assignment 1)

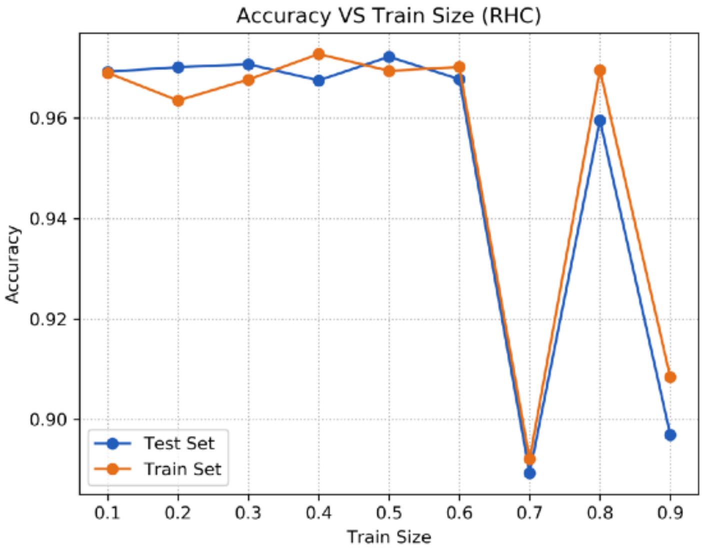
Number of Nodes per layer: 40 (Number of features)

We then focus on tweaking to parameters for our random search.

A) Number of resets

B) The training size of our sample.





In the **above figures** we are initializing random weights and converging, we then reset (Number of resets). From the figure above you can see that as we increase the number of resets we become more likely to find a better minimization of the loss function, which leads us to having better accuracy. However it is still probable to fall into local minima during our random search even if we are doing a large number of resets. It is also noticeable that there is no strong relationship between number of resets and accuracy. It just becomes more ***probable*** that we find a better space to climb with more resets. It is also probable that we could find the global best solution in the first time but, the probability of that to happen is certainly lower. What is reassuring about these results is that some local points are converged to several times that is due to randomness and converging at the sample place again.

The next part of the analysis was to explore the performance of random hill climbing with regard to the train size. Having a large train size gives a bigger space during our search and can provide for more local optima. This can allow us to become trapped in a local minima more easy and often or makes it more probable that we do, on the other hand having a smaller search space makes it less probable that we would not optimize well. The **figure on the right** explores these points. The smaller search space from 0.1 to 0.3 is helping us have a better chance in finding a good optimization, but as we increase the size it becomes more sporadic.
The right graph shows the convergence of each restart with varying test size, each point on the graph is a convergence point in the function. The function then restarts 100 times for each test size.

Random Hill climbing results:

Model Type	**Best RHC Parameters**	**Hyperparameters**	**Time to train**	**Train Size**	**Accuracy**
Neural Network	Resets = **300** *Trained each random starting weights till convergence*	2 layers (20,20)	20.712 seconds	0.8	0.969 *This accuracy can vary slightly due to sample shuffling*

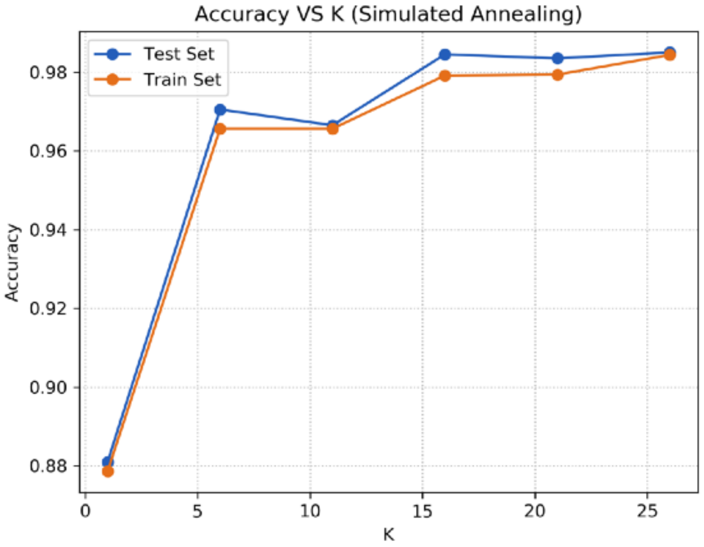
Class Name	**Precision**	**Recall**	**F1**	**Support**
Attack	0.99	1.0	0.99	1750
Normal	0.98	0.90	0.94	250

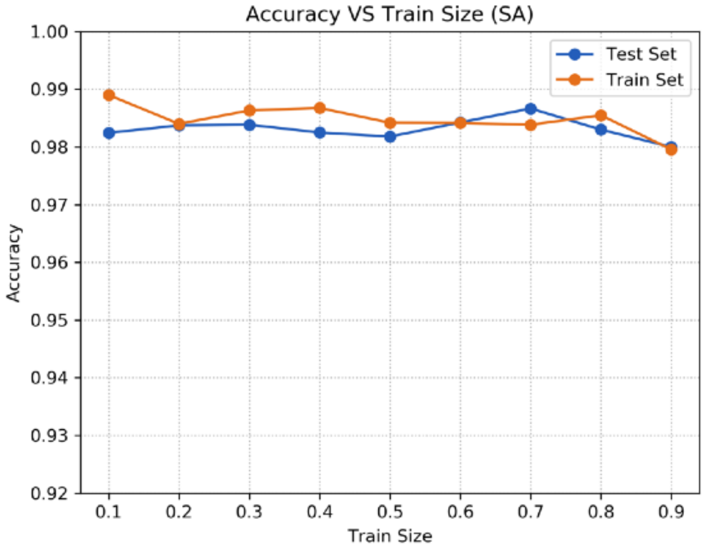
To conclude our RHC analysis, the bigger the search space and the harder the objective function the less likely that RHC will perform well.

<u>Simulated Annealing:</u>

Simulated is the process by which we try to check *K* number if neighboring solution and see if they optimize our function. In our case our neighboring solutions were created by altering only one weight at a time for our neural network then check the acceptance of the new solution, and based on the acceptance probability we would change the weights.

The Two parameters we will explore is A) K B) Train Size





Notice, we do not keep change the entire weights each time, we only change one weight from our starting location which helps with not getting stuck in local points as opposed to Random Hill Climbing.
From the above figure you can see that as we increase our K (Number of neighboring solutions to try) we get a better accuracy. **This was done using temperature 1.0 and alpha 0.9 (decay)**
From the figure above you can see that simulated annealing is not very sensitive to the size of the search space. It is able to escape local points in our objective function and find better optimal solutions. **Compared to random hill climbing which is more sensitive to the size of the search space. In addition Simulated Annealing proved to be faster than Random Hill Climbing.**

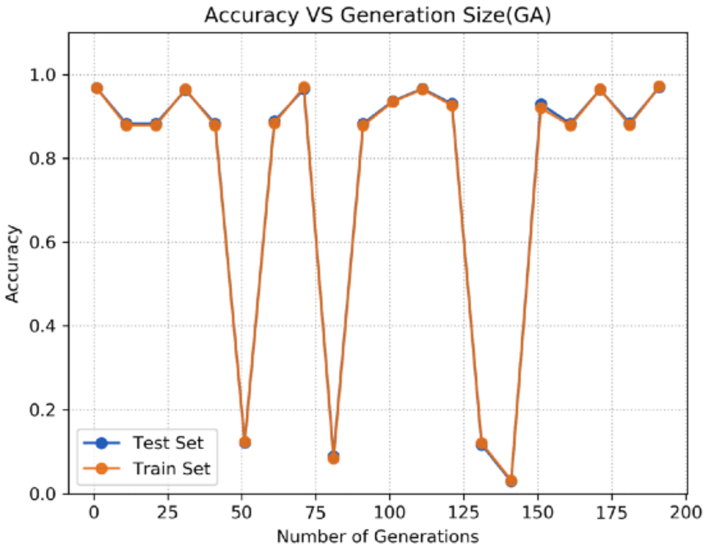
Simulated Annealing Results:

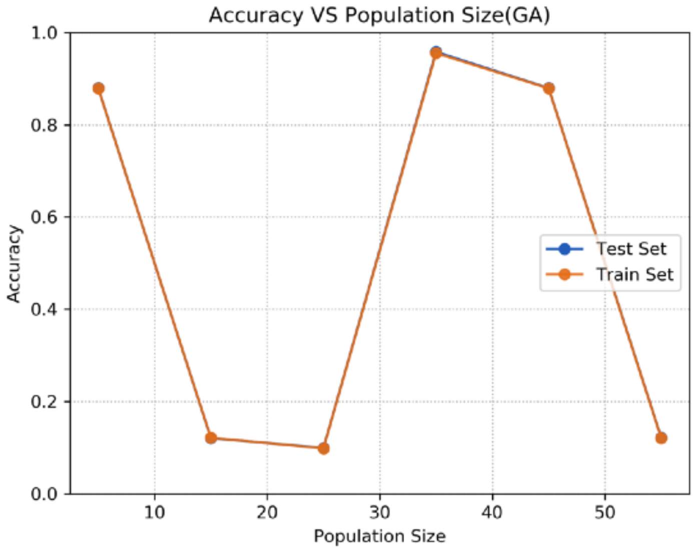
Model Type	Best RHC Parameters	Hyperparameters	Time to train	Train Size	Accuracy
Neural Network	K=3.0 T=1.0 Alpha=0.9	2 layers (20,20)	12.017s	0.8	0.9845 *This accuracy can vary slightly*

Class Name	Precision	Recall	F1	Support
Attack	0.99	1.0	0.99	1750
Normal	0.98	0.90	0.94	250

Genetic Algorithms:

Generic algorithms follow the evolution process where a population is created and the most fit individuals (Optimize the best) are chosen to mate creating a new population. The individuals selected to mate are passed on to the next generation where the process is repeated N times (Number of generations) this allows for fast escape of local minima. We also keep a balanced set of population size as we create our generations and select our parents
First we initialize a population and measure the fitness, this allows us to select the parents.
Parameters to consider in our GA A) Population Size B) Number of generations created
Set parameters:
a) **Cross Over probability: 80%.** b) **Mutation probability: 2%**





The **Figure above(left)** shows the effect of increasing the number of generations affects accuracy. It might not seem very obvious initially that the number of generations affect the accuracy, that is because we could have **started with a population (size=10) that already had lower frequency and therefore in that case we would need a higher number of generations to optimize well.** You can also see how Genetic Algorithms are able to **quickly jump to better solutions very quick.** Whats nice about this compared to a random hill search or a simulated annealing is that you can highly parallelize this process till you reach an optimization that you prefer.

Now we explore the effect on population size **(Figure top right**) on the optimization of genetic algorithm. Keep in mind that as we increase the population size, **we have a larger pool of individuals to pick from therefore we can easier quickly get stuck in local solutions OR quickly find global solutions, so we can fluctuate fast.** The parameter for number of generations was kept at **200**

GA Results:

Model Type	**Best RHC Parameters**	**Hyperparameters**	**Time to train**	**Train Size**	**Accuracy**
Neural Network	Cross Over: 80% Mutation: 2%	2 layers (20,20)	3.01s	0.8	0.9705 *This accuracy can vary slightly*

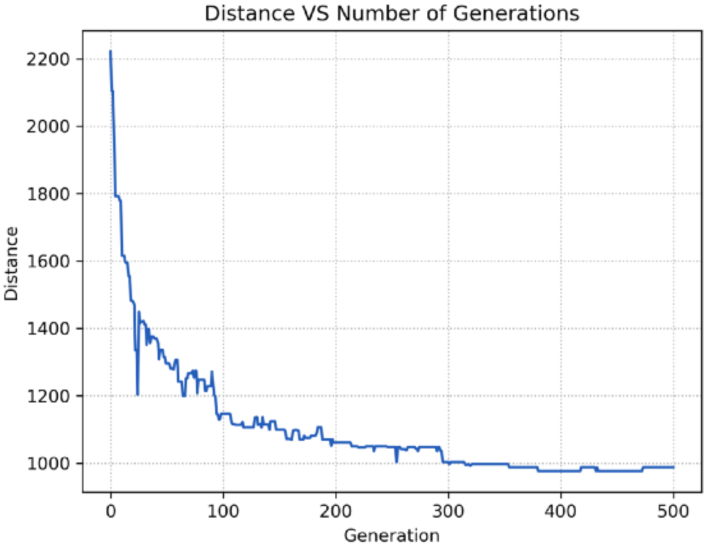
Class Name	**Precision**	**Recall**	**F1**	**Support**
Attack	0.94	1.0	0.97	1763
Normal	0.98	0.52	0.68	237

Optimization Problem I:

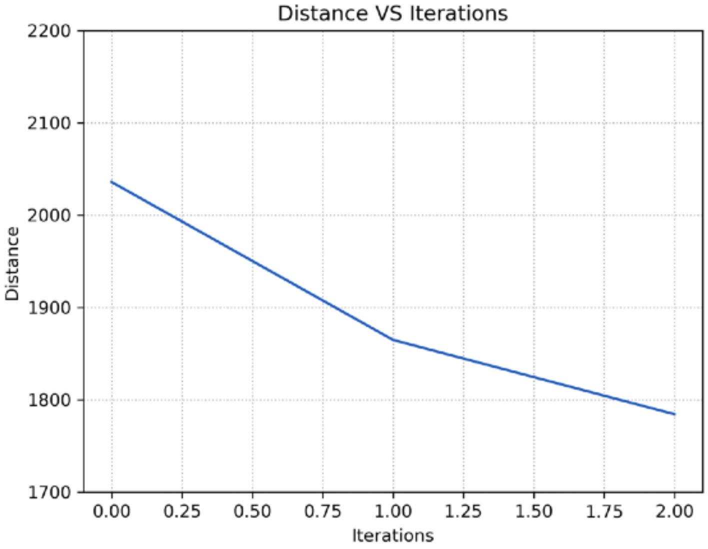
Traveling Salesman:

Traveling salesman is an NP-Hard problem studied in computer science and operations research.The problem is interesting for shipping and delivering agencies such as UPS or FEDEX etc. The naive approach to this problem would be to calculate total distance for each route and add keep doing that till you find the best solution. However with randomized optimization you are able to quickly jump to better solutions.

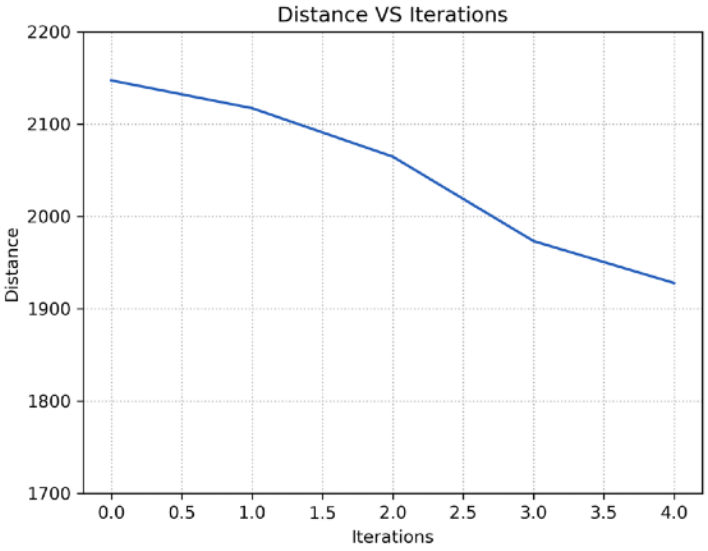
Genetic Algorithm
Generations = **500**
Population = **100**
Mutation = **0.01**



RHC
Resets = 500



Simulated Annealing K = 300, Alpha = 0.9, T = 5.0



Summary results in order of performance:

Algorithm	*Starting Distance*	*Final Distance*
Genetic Algorithm	2147	917
RHC	2147	1784
SA	2147	1927

We apply each optimization algorithm on the **same randomly generated dataset**, the results show that we get the **best optimization** from using the Genetic Algorithm.

Note: I am only plotting the improvements as we go through algorithm. Meaning that each time The fitness is improved I record the results.

The genetic results were able to give us a better result for the problem while Simulated Annealing seemed to converge to local maxima often. On the other hand Random Hill Search performed significantly better than Simulated Annealing in that type of problem. The **Acceptance Probability** used for our case was 50% multiplied by the current temperature. So as we decrease the temperature our chances of accepting a new set is lowered, but initially we are more flexible to change.

The **Neighboring State** used for the traveling salesman was a random swap between two locations in our data, we then see if this improves our fitness, if it does not improve our fitness it goes into the acceptance function and based on a probability we choose to alter our state or not.

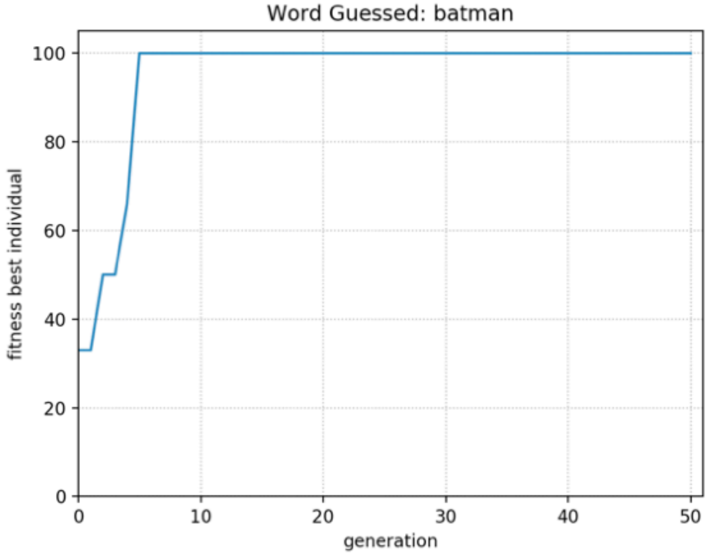
Optimization II:

Cracking a password:

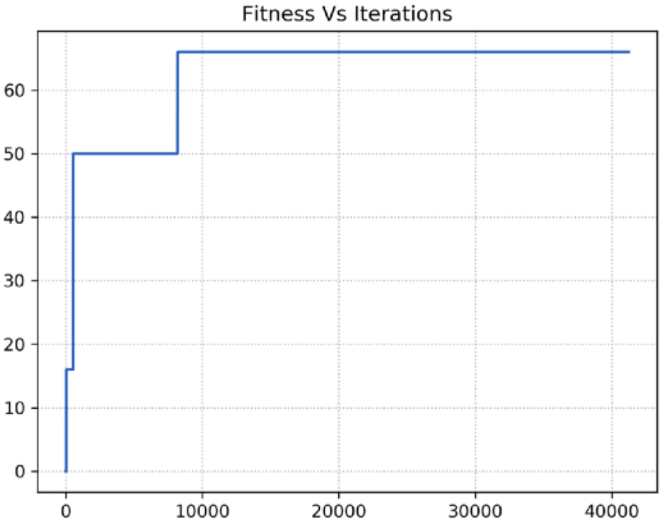
For the second problem I chose something a little bit fun in my opinion which works guess or cracking a password. In real life we don't have a fitness function to guess password so in a real world scenario we don't have guidance in cracking a password, therefore we would have to try every combination of variable length to guess it. But, if we did have a function that would let us know some sort of fitness value we would be able to apply randomized optimization and be able to guess a password relatively quickly using these algorithms.

Password = **'batman'**

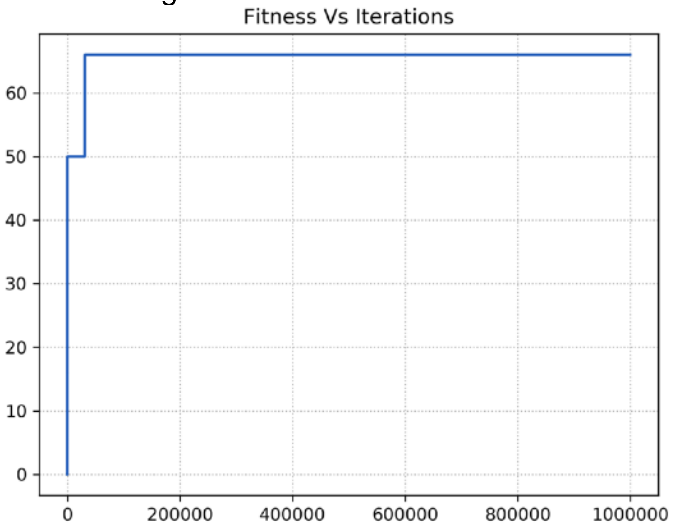
Genetic algorithm: Guessed word: **batman**



SA Guessed word: **balmen**



Genetic algorithm: Guessed word: **bamuan**



Summary Results:

Algorithm	*Final Fitness*	*Guessed Word*
Genetic Algorithm	100	Batman
RHC	66	Bamuan
SA	66	Balmen

Since this is about guessing of a string Genetic algorithm was quickly able to create offsprings with high fitness, this makes sense because if we are able to cut a string at different positions and through evolution we are able to piece together substrings that form the exact correct word. For Simulated Annealing is makes sense that we are less likely to get stuck in local maxima, however we are still liable to fall in local minima based on where we start. The random hill climbing was also similar to a brute force in someway because we try to reset and test the fitness at each time, The simulated annealing is still less likely to get stuck in local maxima than random hill climbing because we do not completely reset each time. However if we try a smaller password like 'cats' for example. Simulated annealing seems to perform a little *faster* and slightly more accurate than random hill search.

To conclude each optimization method has certain advantages with regards to **computation time, parallelization, search space and other factors.** Simulated annealing is less likely to fall in local positions than Random Hill Climbing. And Genetic Algorithms are more likely to jump to the solution right away.

For the problems addressed genetic algorithms have proved to have more power over the other algorithms except for **optimizing the neural network weights.** In my case the simulated annealing was quickly able to reach a highly accurate set of weights, but in all occasions Genetic Algorithm was always competing with simulated annealing. On the other hand in some cases a random hill search can be better such as the case with password cracking.