Game Design for Better Security of Combination Locks

Jean Pierre Astudillo Guerra, Karim Ahmed, Ryan Maher, Eddie Ubri, Jeremy Blum
The Pennsylvania State University
{jpa5180, kma5904, rxm5658, evu5018, jjb24}@psu.edu

Abstract

Dial locks are commonly used to secure a person's items. Commercially available dial locks often use four or five wheels of letters, allowing a user to select a word as a combination. In order to evaluate the security of these locks, we create a game, with an instance created by the lock designer, and played by a lock owner and a thief. In the game, the lock owner chooses a word as a combination, and the thief creates a brute force strategy to try all possible combinations that yield words until the combination is found. To accomplish the task, the thief will solve a version of the Probabilistic Travelling Salesman Problem (PTSP) by creating an a priori tour through all the words a lock can create. The goal for the game designer, then, is to create a lock configuration that maximizes the expected length of the best possible PTSP tour. This paper describes a Genetic Algorithm (GA) approach to design a near-optimal game, i.e. a lock configuration that makes it as difficult for the thief to crack. An analysis of the output of the GA shows that the locks that the system creates are significantly more secure than both commercial locks, in the context of this game..

Introduction

Figure 1 displays a common combination lock, in which an owner selects a n letter combination, chosen from sets of letters provided for each position. The most common configurations have four or five wheels with 10 letters on each wheel. Often a space is provided on the last wheel, so that a user can choose the space at the end of the combination for an $n - 1$ letter word. To analyze the security of these locks, we formulate a game, whose instance is created by a lock designer and played by a thief and a lock owner. In this game, the designer would like to choose a configuration for the lock that makes it difficult for the thief to discover the combination chosen by a lock owner. A lock owner chooses a word at random that can be made by the lock. The thief creates a plan to try all the possible words as combinations, with as few expected operations as possible.

Figure 1. A 4-letter code combination lock

More specifically, the formulation of this competition as a game uses the following rules and assumptions. First, we assume that the lock owner will choose as a combination, with equal probability, one of the dictionary words that can be made from the lock. The lock owner will set the initial state of the lock so that it initially it is not a dictionary word. The thief, starting from the initial state, will select an order to visit all possible words that minimizes the expected number of operations needed to check every dictionary word. We define an operation as either a turn of a wheel or an attempt to open the lock. The problem that the thief is solving is a version of the NP-hard Probabilistic Travelling Salesman Problem (PTSP) (Jaillet 1988).

Under the threat environment represented by this game, the lock designer must set two parameters: the letters on each wheel and the ordering of the letters within a wheel. Generally speaking, to maximize the length of the PTSP solution, the designer would like to create a design with a large number of possible words and a large distance between each word. Given the intractability of the PTSP problem, the lock design problem is also intractable.

To manage this intractability, the lock design algorithm estimates the optimal PTSP tour length through a using an algorithm that provides a lower bound for the length of this optimal tour. A GA algorithm then attempts to find the lock

design which maximizes the length of this lower bound for the PTSP tour length.

The lock configuration from the GA is then compared to commercially available dial locks. The results indicate that the lock configuration found by our algorithm greatly increased the amount of work required for the thief, with an increase of 77.4% operations on average and a 56.7% increase in the total number of words our lock can create. While these locks likely had additional design criteria, this significant improvement suggests that viewing the security of commercially available locks in the context of this game could improve their security.

Related Work

The PTSP is a well-studied stochastic routing problem in combinatorial optimization. In the PTSP, a demand to visit each node occurs (with probability p) or does not occur (with probability 1 − p) during a given day. In the Travelling Salesman Problem (TSP), the objective is to find the shortest tour through all the cities such that no city is visited twice, and the salesman returns at the end of the tour back to the starting city. However, in the PTSP the objective is to minimize the expected length of the a priori tour where each customer requires a visit only with a given probability (Marinakis and Marinaki 2010). The a priori tour can be seen as a template for the visiting sequence of all customers. In a given instance, the salesman travels along the a priori tour until all customers that should be visited are reached. The remaining ones that do not need to be visited will simply be skipped. The TSP can be treated as a special case of the PTSP. The main difference between PTSP and TSP is that in PTSP the probability of each node being visited is between 0.0 and 1.0 while in TSP the probability of each node being visited is 1.0 (Liu 2007).

The PTSP belongs to the class of NP-hard problems (Bertsimas, Jaillet, and Odoni 1990). This means that there is no known polynomial time algorithm for its solution. Therefore, there have been algorithms created with powerful heuristics to find good suboptimal solutions in reasonable amounts of time.

Let's consider this routing problem with a set of n nodes (Jaillet 1988). On any given instance of the problem, only a subset consisting of k out of the n nodes ($0 \leq k \leq n$) must be visited. However, the exact k nodes that must be visited are not known a priori. The length of an actual tour can be defined as the length of this a priori path from the start node to the last of the k nodes in a given instance. Since the instance is chosen from a probability distribution, and the tour is set before this instance is known, the goal is to minimize the expected length of this tour.

Using a similar approach to the PTSP, the lock designer wants to create as hard an instance of the PTSP problem as possible. Thus, the goal is to create a lock configuration that maximizes the expected length of this a priori tour through all the potential words.

Since the PTSP is NP-hard, we will not try to solve it exactly. Rather, we will use a lower bound on the PTSP problem as an estimate of the fitness for the optimal PTSP tour. Lower bounds for the TSP problem have been established for a variety of reasons in the past, including speeding up branch-and-bound algorithms (Christofides 1972). These lower bounds include ones derived from the Minimum Spanning Tree, and repeated reduction of the distance matrix through a contraction algorithm.

As discussed earlier in this paper, the task of finding the optimal lock configuration is an intractable problem. In this work, we use a Genetic Algorithm (GA) to find a near optimal solution to this intractable problem. Genetic Algorithms are a family of computational models inspired by evolution. These algorithms encode a potential solution to a specific problem on a simple chromosome-like data structure and apply recombination operators to these structures in such a way as to preserve critical information (Whitley 1994). Genetic algorithms simulate natural selection through the incorporation of a survival-of-the-fittest approach. A population is seeded with individuals that represent initial guesses at a good solution. Then, a crossover process combines genetic material of existing individuals to create a new generation of solutions (Dwivedi et. al. 2012). As part of the combination process, there is typically the chance of a mutation event which simulates mutations in real chromosomes. These mutations serve to widen the search of the solution space.

A Genetic Algorithm for Lock Configuration Optimization

The performance of the Genetic Algorithm for lock design depends mostly on the encoding scheme and the choice of genetic operators, including the initial population, the selection, the crossover, and the mutation operators. Before introducing the details of these operators, we first describe the lower bound for the PTSP problem that will be used to determine a solution's fitness.

Lower Bound for PTSP Solution

The thief's goal is to find an a priori tour through all the words that minimizes the expected number of operations, where an operation is defined either as a turn of the dial one position or an attempt to open the lock. The idea behind the lower bound for the PTSP problem is that the a priori tour must change one or more dials to move to a new word and then attempt to open the lock.

To calculate this lower bound, we start by averaging the minimum distance between a word and the two closest

Wheel 1	Wheel 2	Wheel 3	Wheel 4
BCFX	AOXZ	KRXZ	EKMT

Figure 2. Simple lock configuration.

words. Let d_i be the i^{th} distance in this list when ordered from least to greatest. This ordering represents the order in which the thief could visit the words, with the idea that the thief would start by trying words that are close together before moving on to words that require more operations to reach.

We use the average because for each word we must move to the word, and then we must also move away from the word. Note that we make an adjustment to the last entry in this list of possible words, d_n. Rather than taking the average of the distances to two closest words, we use just the distance to the closest word. The rationale is that the thief will not need to move off of the last word in the list.

After sorting the list of distances, we must add one to each distance to simulate the thief attempting to open a lock. Thus, the fitness of a solution S, $F(S)$, is estimated as:

$$F(S) = \frac{1}{n}\sum_{i=1}^{n}\sum_{i=1}^{n}(d_i + 1) \quad (1)$$

To better illustrate this fitness estimation, consider the following example. Assume that the list of words is: BAKE, CAKE, FAKE, FORK, FORM, and FORT. Now, assume the lock has a configuration as shown in Figure 2, with an starting code of BZKE. The optimal PTSP will visit the words in the order given by the list above. First, the thief will change the combination of BZKE to BAKE and try to open the lock, requiring two operations. Next the thief will change the combination from BAKE to CAKE requiring two more operations, and a running total of 4 operations. Working through the remaining words, the running number of operations for each word of 2, 4, 6, 10, 12, and 14. Summing these operations and dividing by 6, yields a fitness of 7.92 expected operations.

Tables 1 and 2 show the calculation for the lower bound of the fitness of this solution. We start by taking the average of the two closest words for each possible word in the list, as shown in Table 1.

Word	Nearest Words		Ave. Distance to Nearest Words
BAKE	CAKE	FAKE	1.5
CAKE	BAKE	FAKE	1
FAKE	CAKE	BAKE	1.5
FORK	FORM	FORT	1.5
FORM	FORK	FORT	1
FORT	FORM	FORK	1.5

Table 1. Distances to nearest words for each word in the word list.

Sorted d_i's	Distances Running Total
2	2
2	4
2.5	6.5
2.5	9
2.5	11.5
2.5	14
Sum of Distances:	47
Adjustment:	0.5
$\boldsymbol{F(S)}$**:**	7.75

Table 2. Calculation of the lower bound for the fitness.

Next, we sort these distances and add one to each value to calculate the list of sorted distances. Note that after we sum the distances running total, we make an adjustment for the last entry in the list, d_n, since we do not need to move from the last word tried. Therefore, for this specific example, $F(S) = 7.75$.

Solution Representation

A lock in the GA is encoded as lists of symbols, one per wheel, as shown in Figure 3. The symbols can be either letters or a blank space. The order of the letters corresponds to the order of the letter in the wheels. All of the mutation and crossover operations, described later, will ensure that the symbols on a wheel are unique.

Wheel 1	Wheel 2	Wheel 3	Wheel 4
PTNSBCADRM	UEOCSRAIMP	TRAMNCDSEP	A SCDEOTPI

Figure 3. Lock representation of a four-wheel lock used in the GA code.

Initializing the Population

Setting the initial population for the GA is crucial to the speed and the convergence of the algorithm (Abdoun, Abouchabaka, and Tajani 2012). In our approach, the initial group of lock configurations is set by three initialization operators. The operators are chosen uniformly at random, and each creates a single configuration. This process is repeated until the population reaches a size of 40 solutions.

Initialization Based on Dictionary Words. This initialization operator chooses a random word from the dictionary that is equal to or less than the length of the number of wheels a lock has. It then adds each letter of that word to each wheel of a lock depending on the position of the letter in that word. This heuristic is repeated until every wheel of the lock is filled to their max letter size, in our case 10 letters per wheel.

Initialization Based on Letter Frequency. This initialization operator simulates a raffle process where the letters are chosen with a probability equal to the frequency that the letter appears in the dictionary. When choosing a letter for the i^{th} wheel, the frequency for a letter is the proportion of

words where this letter is used as the i^{th} letter in a word whose length is less than or equal to the number of dials on the lock.

Initialization Based on Random Letter Selection. This initialization operator is a simple random letter selector. The operator simply chooses letters or the space at random until the lock dials are filled. The operator makes sure that the letters are not repeated in a wheel.

Selection

A selection algorithm selects solutions to combine to create a new solution in the next generation. While there are many different types of selection, we will use the most common type, roulette wheel selection (Abdoun, Abouchabaka, and Tajani 2012). In roulette wheel selection, the individuals in the population are given a probability of being selected to generate chil-dren for the next population, that is directly proportion-al to their fitness. Therefore, in this approach, individuals who have high values of the fitness function are more likely to be chosen among the individuals to generate the children. After selecting two of these solutions, called parents, two children are then created by applying a crossover algorithm to the parents.

Crossover

After a pair of parents are selected from the pool, the crossover operation creates two new solutions based on these parents. Our approach uses a modified ordered crossover method (OX) as the crossover operator, an approach that has proven to be one of the best approaches for the TSP (Abdoun and Abouchabaka 2011).

This modified order crossover method is used when the problem is order based, like the problem in our research. Given two parent chromosomes, two crossover points (called crossover sites) are selected at random, partitioning the solutions into a left, middle, and right portion. The ordered two points crossover sites behave as follows: child 1 inherits its left and right section from parent 1 and its middle section is determined by the middle section in parent 2. The same thing happens to child 2 but with parent 2 being the left and right sections instead, and the middle section determined by parent 1.

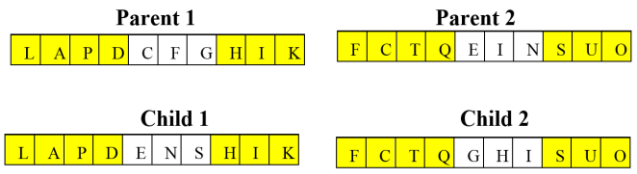

Figure 4. OX Crossover operator using only one wheel.

Figure 4 shows an example of the crossover algorithm OX with only one wheel. First, the crossover points are selected, which are at the fourth and seventh letter, in the example. The letters are then copied from a corresponding parent, at and before the first crossover point. and at and after the second crossover point.

The modification to the OX algorithm then follows. Rather than simply copy the middle section from the other parent, the operator starts copying letters from the other parent after the first crossover point, skipping any letters that have already been selected. We wrap around to the beginning of the wheel when looking for this next letter, if necessary. For example, in Figure 4, we cannot start by copying the letter C to child 2 because a C already appears in the wheel. The same is true of the letter F. Instead, we skip these letters and start copying from parent 1 at the G. Similarly, for child 1, normally the letter I would appear as the sixth letter. However, since it already appears in the wheel, we skip it, adding the N and the S from parent 2.

The modified OX operator is applied to all of the wheels in a similar fashion. Different crossover points are chosen, at random, for each wheel.

Mutation

After creating child solutions, GA's typically apply mutation operators. Our approach applies five mutation operations, each one running a random number of times between 0 and 50. The mutation produced by the operator is discarded if it does not yield an improvement in fitness. The first two mutations are designed to increase the number of words that can be made with the letters on the lock. The last three are designed to place the letters in such a way that the number of operations needed to visit all words increases.

Mutation based on a Random Word. The first mutation chooses a random letter from a random word in the dictionary and ensures that letter is present in the lock. The operator first picks a wheel and a letter to add at random. Then, with equal probability, it randomly picks a letter already configured in that wheel and replaces it with the letter chosen before.

Mutation Based on Underutilized Letters. This mutation operator first picks a letter and the wheel to insert that letter at random. Then, it selects the letter that participates in the fewest number of words in that wheel. The fitness after this change is then evaluated. If this new solution creates more words, the operator terminates. Otherwise, the operator will try to find a different replacement letter for a limited number of times.

Mutation based on Random Swap. This mutation operator is a simple mutation that is based on randomness. The operator first chooses a wheel at random. Then, it picks two letters in that wheel at random and swaps the locations of the letters.

Mutation to Interleave Common and Uncommon Letters. This mutation operator attempts to place commonly used letters next to uncommon letters in the wheels of a lock. This should increase the distance between words. The operator first chooses a wheel at random. Then, the operator divides the letters into two equal subsets, a common subset and an uncommon subset, based on the frequency that they appear in words. The operator then chooses a letter in that wheel at random, with position i, and checks if it is common or uncommon. Depending on the characteristic of letter i, the operator checks that the letter $i + 1$ is the opposite. If it is not, the operator chooses the first letter, of the opposite type, that it finds in that wheel and swaps both positions of the letters.

Mutation Swapping Replacement Letters. This mutation operator seeks to separate pairs of letters that occur in words that differ only in the pair of letters. For example, there are a large number of pairs of four-letter words that are identical with the exception of containing either an "a" or an "i" as the second letter, for example, "tack" and "tick", or "bake" and "bike". This operator would try to place the "a" and the "i" at some distance away in the second wheel.

To do this, the operator first picks a wheel and a letter in that wheel at random. Then, the operator goes through all the words that the lock can create and finds the letter that appears with this chosen one in the most number of similar words. The operator then moves this letter to a random location between 3 and 5 turns from the original letter.

Insertion

The GA uses elitism when inserting new solutions in the population. Rather than replacing all parents with new solutions, elitism preserves the best solutions in the population (Chakraborty and Chaudhuri 2003). The best 40 solutions are always retained regardless of whether they were present in the population prior to the generation of new solutions.

Experimental Results

To solve the lock designer's problem, an experiment was performed with three goals in mind: to find a lock that will create the greatest number of words, to find a configuration of this lock that maximizes the lower bound fitness, and to compare commercially available locks to this optimal lock configuration. The experiments found locks that produce usable combinations for more than 55% of the possible combinations. The results contained some surprises that indicate that heuristic approaches to lock design are not likely to yield good results. Moreover, the best lock found by the GA system greatly outperformed commercial locks with about a 60% increase in both the number of words that are possible as well as the lower bound on the expected number of operations needed for a thief to break the lock.

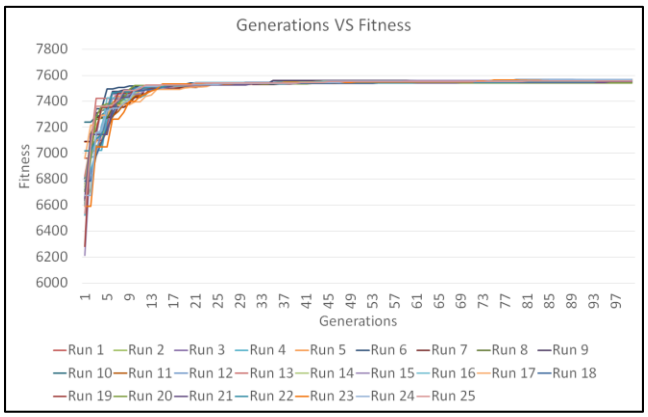

Figure 5. Fitness of best solution every run by generation.

	Min	Max	Mean	Median	Standard Deviation
Fitness	7542.555	7567.246	7558.562	7558.845	6.086481909
Words	5539	5540	5539.48	5539	0.509901951

Table 3. Fitness and number of words in every run.

Overview of Results

The GA was designed and implemented in Java. The word list used in the experiment came from a dataset containing the 333,333 most commonly-used single words on the English language web, as derived from the Google Web Trillion Word Corpus (Norvig 2008). This list includes 12,977 3-letter strings and 31,140 4-letter strings. The list is larger than typical dictionaries because it includes non-word strings that appear in web pages like "aaaa" and "yolo." It seems reasonable that these common non-word strings would make likely combinations chosen by lock owners.

To evaluate the performance of the GA, the algorithm ran 25 times each with a population size of 40 that evolved for 100 generations. Figure 5 plots the fitness of the best solution in each generation for these 25 runs. The GA performs consistently, with rapid improvement in early generations and plateaus to roughly the same fitness in later generations.

Table 3 displays statistics for the central tendencies and dispersion of the best solution in each of the 25 runs. As seen in the table, the number of words, which can be made by the locks, and the fitness of the best solution were consistent across these optimization runs. This consistency suggests that the design of the GA does well in avoiding being stuck in local optima far from the global optima.

Best Lock Configuration

The best lock configuration found in the 25 runs is displayed in Figure 6. There are some surprising results in this lock that suggest that simpler optimization approaches to lock design will not work as well as the GA.

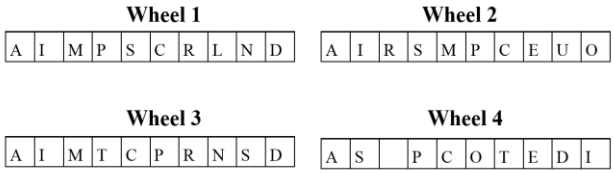

Figure 6. Best lock configuration design found by GA.

First, the lock with the best fitness made 5,539 words, rather than the maximum of 5,540 found in other runs. Recall that since the PTSP uses a running total of distances, the effect of adding one more word is significant since the last word includes the sum of all previous distances. This result suggests that the letter selection and placement should be optimized together, as in the GA system. The best lock configuration may not be found by a simpler, two-phase optimization process, in which a system first selects letters and then selects ordering of the letters.

Second, we had expected that the final wheel would contain consonants with the space and the letter "e." However, in the best lock, other vowels also appeared, and an examination of the dictionary confirmed that these vowels appeared in a large number of words. The 8 most common consonants in English are r, t, n, s, l, c, d, and p (Keating 2021). Table 4 shows the change in the number of words after substituting one of the consonants that was missing with one of the vowels. Note that the number of words after replacing one of these vowels with a consonant always decreased.

The third surprise was that we expected that vowels in a wheel would be better to be separated, rather than appearing in adjacent spots. We had assumed that separating these letters would increase the minimum distance between words. To confirm that the configuration is reasonable, we compared the fitness in the configuration found by the GA with ones where the vowels were separated. The results, shown in Table 5, confirm that the configuration produced by the GA outperforms the other possibilities that were tested.

Vowel	Consonant	Words Before Replacement	Words After Replacement
A	R	5539	5149
A	N	5539	5114
A	L	5539	5122
I	R	5539	5381
I	N	5539	5346
I	L	5539	5354
O	R	5539	5453
O	N	5539	5418
O	L	5539	5426

Table 4. Effect of replacing a vowel with a common consonant in wheel 4 of the best lock.

Wheel Num.	GA Config-uration	Alter-nate Config-uration	Fitness Before Change	Fitness After Change
1	AIMPS CRLND	AMPSC IRLND	7567.246	7545.232
2	AIRSM PCEUO	ARISE MUPOC	7567.246	7505.651
3	AIMTC PRNSD	AMTCP IRNSD	7567.246	7542.555
4	AS PC OTEDI	AS PO CETID	7567.246	7542.918

Table 5. Effect of rearranging letters in the GA lock.

Comparison with Commercial Locks

Commercial lock design can also be evaluated within the context of the game design which the GA system is attempting to optimize. The analysis reveals that under the threat environment modeled by the game, the GA system's lock design significantly outperforms commercial locks.

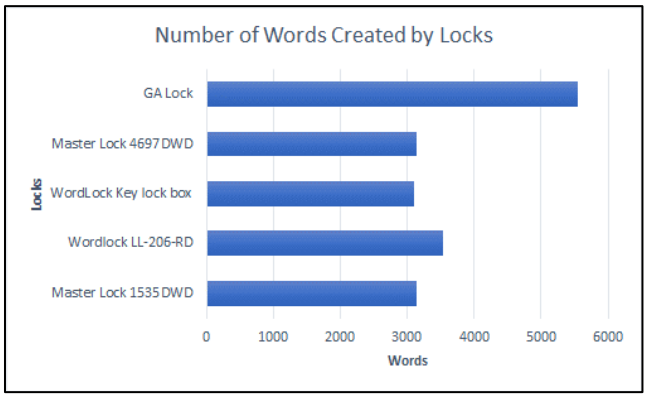

Figure 7. Number of words possible in each lock

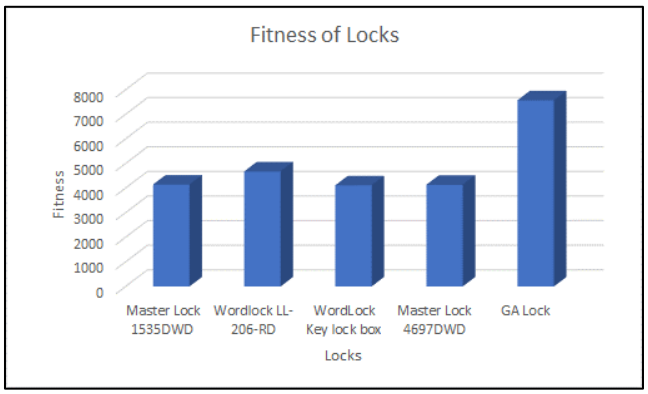

Figure 8. Fitness of commercial lock configurations versus the fitness of the lock configuration designed by the GA.

As shown in Figure 7, the lock designed by the GA makes 56.7% more words than the best commercial lock. Where the commercial locks are able to make words with about 1/3 of the possible combinations, the GA designed lock makes words in more than 1/2 of the possible combinations.

The larger number of possible words is a key reason why the GA designed lock also improves the difficulty of a brute force attack on the lock, as measured by the fitness. Figure 8 shows that the GA lock increases the expected length of the PTSP tour required to break the lock by 77.4% over the commercial locks, on average. Thus, under the threat environment considered by this paper, the GA lock is significantly more secure than the commercial locks in the market.

Limitations

There are limitations in the analysis of the context of the game used to evaluate lock security. These limitations include assumptions about the likelihood that words will be chosen as combinations, the threat environment that locks face, and other important design criteria that are not incorporated in the game structure.

First, the word list chosen for this study may not reflect commonly chosen passwords and within a word list, not all words will be equally likely to be chosen. If these non-uniform probabilities were incorporated, the PTSP tour would likely try to visit more common combinations first.

In addition, a brute force attempt to break the combination may not be the most likely attack on these locks. Other mechanisms including cutting the lock or picking the lock may be more likely real-world scenarios.

Finally, there are certainly other design considerations that are important for these locks. For example, designers may want to be able to make certain words with the lock. In the commercially available locks, for example, some are able to make the words shown in Figure 1, including "runs," "fast," "bike" or "loop." Other than "runs" which is a possible word in the GA lock configuration, these words appear to have been chosen not because their letters can be used for many other words, but rather for marketing reasons.

Conclusions and Future Work

In this paper, we analyzed the security of locks by formulating a game played between a lock designer and a thief. In the game, the thief will use an a priori tour for the PTSP problem to create a brute force strategy to find the combination. We presented a genetic algorithm (GA) system that attempts to maximize the time required for this attack.

The GA system produced significant improvements over commercial lock designs, in the context of this game. The number of words that could be created by the GA-designed lock is 56.7% more than commercial locks, and the difficulty of a brute force attack increased by 77.4%.

Future work, planned by the authors, includes expanding the study to larger lock sizes and addressing some of the limitations of the current study. In addition to four-letter locks, five-letter locks are also commonly available. The addition of one more wheel increases the complexity of the problem by 27 choose 10, increasing the number of possible configurations by a factor of more than 8 million. The authors would also like to explore the effect of assigning non-uniform probabilities to the likelihood that a combination is chosen. Currently, most of the transitions between combinations in the best PTSP tour would be accomplished with just 1 or 2 operations. Using non-uniform probabilities could potentially require that the thief visits combinations in an ordering that increases this number of operations.

References

Abdoun, O., and Abouchabaka, J. 2011. A Comparative Study of Adaptive Crossover Operators for Genetic Algorithms to Resolve the Traveling Salesman Problem. *International Journal of Computer Applications* 31(11): 49-57.

Abdoun, O.; Abouchabaka, J.; and Tajani, C. 2012. Analyzing the Performance of Mutation Operators to Solve the Travelling Salesman Problem. *International Journal of Computer Applications* 2(1): 61-67.

Bertsimas, D. J.; Jaillet, P.; and Odoni, A. R. 1990. A Priori Optimization. *Operations Research* 38(6): 1019-1033.

Chakraborty, B., and Chaudhuri, P. 2003. On The Use of Genetic Algorithm with Elitism in Robust and Nonparametric Multivariate Analysis. *AUSTRIAN JOURNAL OF STATISTICS* 32(1-2): 13-27.

Christofides, N. 1972. Technical Note—Bounds for the Travelling-Salesman Problem. *Operations Research* 20(5): 1044-1056. dio.org/10.1287/opre.20.5.1044.

Dwivedi, V.; Chauhan, T.; Saxena, S.; and Agrawal, P. 2012. Travelling Salesman Problem using Genetic Algorithm. *National Conference on Development of Reliable Information Systems, Techniques and Related Issues* (p. 25).

Jaillet, P. 1988. A Priori Solution of a Traveling Salesman Problem in Which a Random Subset of the Customers are Visited. *Operations Research* 36(6): 929-936.

Keating, B. 2021. The frequency of the letters of the alphabet in English. https://www3.nd.edu/~busiforc/handouts/cryptography/letterfrequencies.html/. Accessed 9/6/2021.

Liu, Y.-H. 2007. A Hybrid Scatter Search for the Probabilistic Traveling Salesman Problem. *Computers & Operations Research* 34: 2949-2963. doi.org/10.1016/j.cor.2005.11.008

Marinakis, Y., and Marinaki, M. 2010. A Hybrid Multi-Swarm Particle Swarm Optimization algorithm for the Probabilistic Traveling Salesman Problem. *Computers & Operations Research* 37(3): 432-442. doi.org/10.1016/j.cor.2009.03.004

Norvig, P. 2008. Natural Language Corpus Data: Beautiful Data. http://norvig.com/ngrams/. Accessed 9/7/2021.

Whitley, D. 1994. A Genetic Algorithm Tutorial. *Statistics and Computing* 4: 65-85.