Traps and Dangers when Modelling Problems for Genetic Algorithms

Stefan Wagner, Michael Affenzeller, Daniel Schragl
Institute of Systems Theory and Simulation
Johannes Kepler University
Altenbergerstrasse 69
A-4040 Linz, Austria
email: {sw,ma}@cast.uni-linz.ac.at, daniels@gmx.at

Abstract

This paper describes the relevant steps when modelling a given problem in order to be attacked by a Genetic Algorithm (GA). As an example for the whole modelling process the cryptanalysis of a generic 3-rotor machine is used. Thereby the authors want to especially highlight the various traps and dangers that might lead to a complete failure of a GA-based approach. Finally a theoretical analysis is given that shows why problems like the used example cannot be solved by Genetic Algorithms sufficiently.

1 Introduction

It is known from the area of complexity theory that problems can be divided into different classes to enable a differentiation concerning the effort (runtime or memory) required to achieve a globally best solution. The class of the so called P-problems contains all problems for which the effort in relation to the problem dimension can be dominated by a polynomial function. Consequently the best solution for such problems can be found in most cases due to the computing power available today. It is even possible in the worst case just to try all possible solution candidates iteratively.

The class of NP-problems in contrast represents such instances whose effort increases more than polynomial (i.e. exponential or even over-exponential) depending on the problem size. Therefore, problems of that type cannot be solved algorithmically even in rather low dimensions because of the enormous number of potential solution candidates that makes it impossible to find the global optimum in acceptable time.

Unfortunately the NP-class contains many problems that are of vital importance especially in the economic field: Among them are in particular routing problems like the Traveling Salesman Problem (TSP) or the Capacitated Vehicle Routing Problem (CVRP), scheduling problems like the preparation of timetables or the optimization of production planning systems and also packing problems like the Knapsack or the Bin Packing Problem. All these problems cannot be solved exactly, if their size is high enough for real-world applications. However, as solutions are required anyway it is a common practice to use so called heuristic optimization algorithms instead. Though these techniques are not able to guarantee a global optimal solution anymore, they have proven to be very reliably according to finding sufficient good solutions.

The development of such heuristic optimization techniques is very often inspired by natural archetypes. For example Genetic Algorithms (GAs) which imitate the natural evolution of a species belong to the most generic and very successful methods to solve combinatorial optimization problems. An overview about GAs and their implementation in various fields is e.g. given in [Goldberg, 1989] or [Michalewicz, 1996].

The idea of Genetic Algorithms (GAs) was first introduced by Holland in 1975 [Holland, 1975]. Based upon Holland's ideas the concept of the Standard Genetic Algorithm (SGA) became accepted (e.g. described by Tomassini [Tomassini, 1995]), which is still very much influenced by the biological archetype. The individuals (solutions) are encoded as bit-strings (i.e. binary encoding) which can be compared with the natural structure of DNA-strings. The solution quality of each individual is evaluated by a fitness function and stored as a so called fitness value. The natural evolution process is abstracted to the three genetic operators selection, crossover and mutation which are realized in the following way: At first individuals are selected for reproduction. In this selection step the probability of an individual to be selected is directly proportional to its fitness value (i.e. roulette wheel selection). After this the second operator (crossover) is used to create a new child out of two selected parents by breaking up the parents' bit-strings at a random position and mutually interchanging one substring (i.e. single point crossover). Finally mutation is used as an undirected background operator whereby a randomly chosen bit in the string of the child is flipped. Fig. 1 gives a schematic diagram of the described procedure.

Due to the enormous increase of computation power since 1975 the full potential of GAs was more and more tapped. Consequently the popularity of the GA-concept increased steadily and many groups around the world started to solve various problems with GAs. However, it soon became clear that for most practical

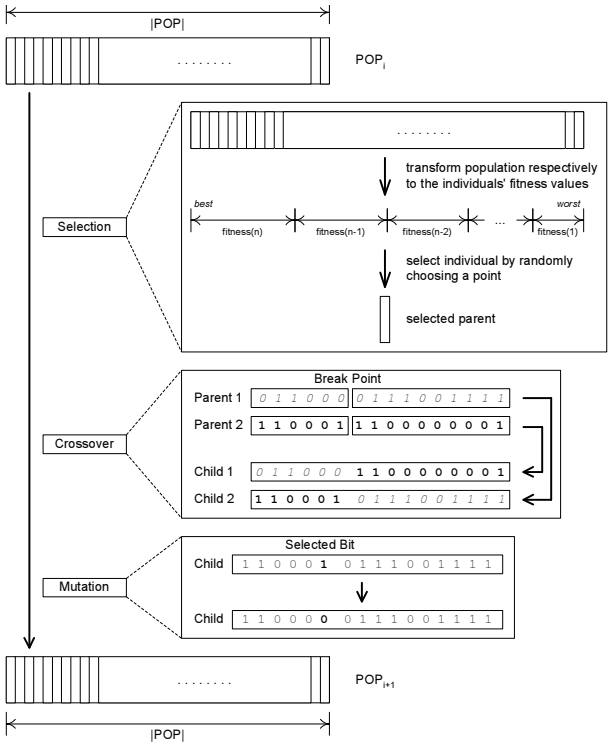

Figure 1: Reproduction sequence of the Standard Genetic Algorithm (SGA) with binary encoding

tasks the binary encoding originally used by Holland was not at all sufficient. Accordingly many different encodings and necessarily also new crossover and mutation operators were introduced which showed a qualitatively very diverse behavior (e.g. an overview of different encodings and operators developed for various applications can be found in [Davis, 1991]).

Until now choosing the encoding and the genetic operators is still one of the most challenging parts when modelling problems for GAs. Moreover, for a GA to be able to solve any problem with acceptable quality the design of an appropriate fitness function is also vitally important and in addition it always has to be kept in mind that not all problems are equally adequate to be attacked by a GA.

In this contribution we want to take a closer look at the whole modelling process by successively going through all relevant steps that are needed to solve a problem by a GA. Thereby the most common traps and dangers should be especially pointed out, as these may lead to a severe detraction of the algorithm's performance. Besides, it should also be pinpointed which properties of a problem make it completely impossible for the GA to solve the problem at all. Therefore, we have chosen the cryptanalysis of a generic 3-rotor machine as an example problem, as this challenge seems to be a combinatorial optimization problem and consequently very well suited for a GA at first sight. However, the following investigations show, why this assumption is not true and what the reasons are that the GA completely fails to solve the problem.

The rest of the paper is structured as follows: In section 2 we start with a detailed definition of the chosen example problem. Section 3 is dedicated to the GA-specific modelling aspects like the encoding, the used genetic operators, the fitness function and the whole algorithm itself. Then, in section 4, experimental results are presented that represent the basis for the theoretical analysis of the algorithm's performance in section 5. Finally, section 6 recapitulates the modelling steps and summarizes the important traps and dangers when using a GA to solve optimization problems.

2 Problem Specification

Before starting to think about the GA-specific modelling aspects it is very important to specify the problem itself and all relevant requirements very clearly. Only when having a precise picture of the problem in mind the modelling and consequently also the resulting algorithm may succeed. Therefore, we will now take a closer look at the problem we want to solve: a ciphertext only attack on a generic 3-rotor machine.

The basic setup of a generic 3-rotor machine can be described in the following way: On both sides each rotor has connections for each letter of the used alphabet. Inside the rotor these connections are catenated so that e.g. the first connection is mapped to the seventh, etc. So each rotor can be seen as a specific permutation of letters. By arranging some of these rotors successively a complex mapping of letters is achieved. Additionally the leftmost rotor is turned by one step after the encoding resp. decoding of a single letter and after a rotor has completed one total rotation the next right neighbor is also turned by one step. This procedure leads to very long periods of letter mappings as the initial mapping is reached again only after also the last rotor has completed one rotation. A schematic diagram of a 3-rotor machine is shown in Fig. 2.

For decoding a given ciphertext the text just has to pass the rotors in the opposite way after the machine has been set up with the same initial rotor positions. Therefore, when performing a ciphertext only attack it is necessary to discover the initial rotor wirings (i.e. the initial permutations) just be looking at the plausibility of the generated 'plaintexts' achieved with a potential initial wiring. Consequently it is necessary to know the language of the original plaintext and also the corresponding letter resp. letter group probabilities in order to be able to perform a statistical analysis to evaluate the various solution candidates.

3 Modelling Process

After having clearly specified the problem that should be solved by the GA we can now take a closer look at the GA-specific aspects. First of all a proper encoding has to be defined that makes clear how an individual represents a solution to the problem. Thereby it also should be kept in mind that genetic operators must be developed to be able to cross two individuals respectively to mutate the generated children. Furthermore, a fitness function is to be determined to measure the quality of each individual in order to steer the genetic

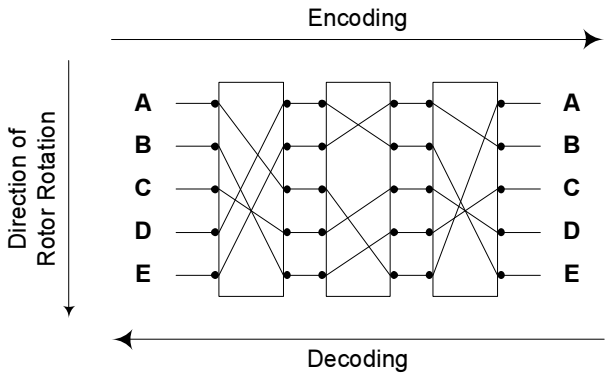

Figure 2: Schematic representation of a 3-rotor machine for a 5 letter alphabet

search process into promising directions. Finally the Genetic Algorithm itself has to be formulated as there are many different GA variants each differently adequate depending on the problem. In the following subsections we will take a closer look at each of these four essential steps.

3.1 Encoding

Basically we try to find the rotor wirings of a generic 3-rotor machine. As already described in section 2 a rotor can be seen as a permutation that is used to map incoming characters to other ones. Therefore, following the KISS-principle (i.e. "Keep It Short and Simple") it is quite obvious to model a solution candidate as a set of 3 permutations where every permutation represents exactly one possible instance of a certain rotor of the underlying rotor machine.

As it is the case in very many other GA applications the binary encoding initially used by Holland would not be reasonable in our case as the development of crossover and mutation operators that produce feasible binary encoded permutations again is rather tricky. Besides a permutation-based encoding is also preferable as this encoding type has been very successful concerning the Traveling Salesman Problem (TSP). Consequently very many different crossover and mutation operators with very differential properties have already been developed.

3.2 Genetic Operators

As mentioned above there are many different genetic operators available for the permutation encoding (an overview can be found e.g. in [Larranaga *et al.*, 1999]). So before choosing one of the existing operators or developing our own new one more thoughts should be spent according to the relevant genetic information.

In every GA application it is of fundamental importance that a generated child preferably consists of the genetic information of its parents only. Depending on how appropriate the used crossover operator for the problem-specific encoding is, so-called unwanted mutations may occur, i.e. that the crossing itself or maybe necessary repairs afterwards lead to elements in the child's genetic information that were not existing in the gene material of neither of the parents. As this effect of unwanted mutation is counterproductive to directed search it should be prevented as much as possible.

When using a permutation-based encoding there are two different possibilities about which aspects of the permutation represents the essential genetic information. E.g. in a permutation describing a tour of a TSP instance in path representation the positions of the numbers (cities) are absolutely not relevant for the quality of the solution. In that case the ordering of the numbers is essential because it represents the different edges of the tour. Therefore, special crossover operators have been developed for the TSP that particularly try to obtain the sequence information and consequently reduce the occurrence of unwanted mutations drastically (e.g. ERX, EERX).

Contrary to the TSP (path representation) when looking at the problem of the cryptanalysis of rotor machines the positions of the numbers in the permutations represent the important information, as these define the mapping of the characters during the encryption respectively decryption process. Consequently a crossover operator should be used that focuses on recombining and maintaining this position information. E.g. in [Oliver *et al.*, 1987] the so-called cycle crossover (CX) is described that was designed especially with respect to this issue. Furthermore, the CX is also a very simple and fast crossover operator which makes it very attractive.

Analogically to the considerations pertaining to the selection of a proper crossover operator it also has to be kept in mind when looking for a mutation operator that the positions in the permutation are relevant. Therefore, the mutation operator should systematically change only a few positions in the permutation to prevent a too drastic manipulation of the solution candidate. As a consequence all mutation operators that base on translocation or inversion of a part of the permutation are not qualified. Also in [Oliver *et al.*, 1987] a mutation operator called swap mutation (SM) is described that just exchanges two randomly selected numbers of the permutation (in the literature this mutation operator is also often named exchange mutation, point mutation, reciprocal exchange mutation or order based mutation). As this mutation operator directly aims at the altering of positions and doesn't change the child too much it seems to be quite suitable for our needs.

3.3 Fitness Function

When applying a ciphertext only attack on a cryptographic system no additional information about the structure or the content of the original text is available to evaluate the various solution candidates. Consequently a quality value for a possible key can only be computed by decrypting the ciphertext with that key and by measuring how good the letter distribution of the obtained text matches with the average letter distribution of the plaintext's language.

This technique is very common when trying ciphertext only attacks and it was used e.g. by Spillman as he attacked some simple substitution ciphers with

GAs [Spillman *et al.*, 1993]. Literature suggests many different formulae to measure the degree of congruence between the two letter distributions. One of these calculates a kind of error value that represents the deviation between letter probabilities in the deciphered text and the average letter probabilities of the considered language. This kind of quality value for ciphertext only attacks is used e.g. in [Clark, 1998] or [Spillman *et al.*, 1993].

Further possibilities for the design of fitness functions arise when not only unigram but also bigram or trigram letter probabilities are used for the evaluation of the deciphered text. By also looking at longer letter groups the sensitivity of the evaluation function is increased a lot, i.e. only if the letter and letter groups distribution of the deciphered text is practically identical with the average values of the specific language, the fitness function has a high value. Seen from a geometrical point of view this means that the steepness of the funnel of the fitness landscape around the correct key is strongly increased.

For your purpose we decided to use the error value fitness function in a form that considers unigram and also bigram letter probabilities. The fitness value is calculated as follows, where $ER(K)$ represents the fitness value of the possible key K, A is the set of all valid characters, $O_u(i)$ resp. $O_b(i,j)$ represents the observed probability of the unigram i or the bigram ij in the deciphered text and $P_u(i)$ resp. $P_b(i,j)$ is the average probability of the letter i or the bigram ij in the considered language:

$$ER(K) = \sum_{i \in A} \Big(|O_u(i) - P_u(i)| + \sum_{j \in A} |O_b(i,j) - P_b(i,j)| \Big)$$

3.4 Algorithm

As soon as the encoding, the genetic operators, and the fitness function are determined, the algorithm itself has to be specified. In literature there exist very many different GA versions and even more hybrid variants. For many applications it might be very promising to choose or develop a specialized algorithm so that e.g. also the positive effects of problem specific local search techniques can be utilized.

However, as the focus of this contribution lies on modelling problems for GAs in general a generic standard GA version seems to be well suited. Consequently the commonly used Standard Genetic Algorithm (SGA) is used, a GA with roulette-wheel selection, generational replacement and 1-elitism. Alg. 1 shows the used algorithm in pseudo-code.

Algorithm 1 Standard Genetic Algorithm (SGA)

```
Initialize total number of iterations
nrOfIterations ∈ ℕ
Initialize size of population |POP|
Initialize mutation probability mutProb
Randomly produce an initial population POP_0
of size |POP|
Calculate fitness of each individual of POP_0
for i = 1 to nrOfIterations do
  Initialize next population POP_{i+1}
  while |POP_{i+1}| ≤ |POP| do
    Select two parents par_1 and par_2 from POP_i
    (Roulette Wheel Selection)
    Generate a new child c from par_1 and par_2
    by crossover (CX)
    Mutate c with probability mutProb (SM)
    Calculate fitness of c
    Insert c into POP_{i+1}
  end while
  Overwrite worst individual in POP_{i+1} with
  best individual from POP_i
end for
```

4 Experimental Results

As naturally no meaningful text with finite length shows the same letter distribution as the average distribution of its specific language, the choice of suitable plaintexts for the experiments turned out to be rather difficult. To avoid such inaccuracies which would definitively affect the performance of the GA in a negative way we decided to generate the plain texts randomly using a given unigram letter probability table. Afterwards we determined the real unigram and bigram letter probabilities of the particular texts and used this distributions as reference values. For the generation we used the average unigram letter distribution of the English language as given in [Bagnall, 1996].

By this random production of plain texts it was also possible to a priori define the text length exactly. Consequently it became possible to analyze the effects of different text lengths on the quality of the attack. So we generated plain texts with 1000, 2000 and even 5000 letters each and encoded them via the generic rotor machine model (see section 2). The therefor necessary initial rotor wirings were taken from the historical archetype, the German Naval Enigma. The permutations of the rotors VI, VII and VIII can be found in [Bagnall, 1996].

The adjustment of the parameters of the used GA was based on experience gained during the experiments. The number of generations was set to 10'000, the population size to 100 and the mutation rate to 5% as these values led to very promising results and are also quite often used for GA experiments of various kinds.

The results of the performed experiments are presented in Table 1. Five test runs were done for each ciphertext length and consequently the given values are the best and average values of these five runs. For each run not only the achieved fitness values are given but also a measure very similar to the Hamming Distance of bit vectors. The Hamming value indicates at

Table 1: Results of the ciphertext only GA attack on a generic 3-rotor machine

Ciphertext Length	Fitness value		Hamming value	
	Best	Average	Best	Average
1000	1.08	1.18	73	75.33
2000	0.56	0.78	76	77.00
5000	0.22	0.38	76	76.80

how many positions the permutations (i.e. the key) found by the algorithm differ from the original key. Therefore a fitness value of 0 indicates that the letter distributions of the plaintext and the deciphered text are identical. However, a Hamming value greater than 0 states that the found key is still not identical with the original one and consequently the deciphered text is not correct.

As it can be seen in the results table the phenomenon of very fit solutions that are in fact totally useless due to their high Hamming value is occurring in every run. The ciphertext length doesn't have any effect on that behavior. Therefore it seems that a GA is completely unable to solve the given problem.

In the following section we will now take a closer look at the observed performance and also think of a theoretical explanation. Especially one interesting question remains: Why did the approach fail? Was there any problem concerning the modelling or is the problem not suitable for Genetic Algorithms or perhaps even for any optimization technique at all?

5 Theoretical Analysis

Looking back to the modelling process it seems that we were rather careful: We formulated a precise and well structured problem definition, we chose an intuitive and well established encoding, we thought of the relevant genetic information (in our case the number positions in the permutations), we also took a crossover operator that was especially designed to maintain that information and to combine parts of it in an appropriate way and finally we also decided to use a mutation operator that produces small position changes only.

In fact the GA should be able to combine the parents' genetic material in an advantageous way, i.e. in our context bring together longer and longer sequences of correct connections in the different rotors. However, this is not the case. Obviously the GA is optimizing the solutions in a certain way as the fitness value of the actual best solution is constantly decreasing. However, the key represented by the currently best solution is not coming closer to the original key.

This observed inability can be explained by the fact that the fitness function is not enough correlated with the real correctness of the found solution. If two solutions of different correctness (i.e. different Hamming value) are compared, it is not sure that the fitness values of these two solutions also represent the same situation. That is, the fitness value of the better solution can actually be worse than the fitness of the worse solution.

However, as the correct key is certainly not available, the GA can only use the fitness values to steer the search process into the 'right' direction. Consequently, when the fitness values don't adequately represent the real solution qualities the GA is misled and ends up with a solution that has a good fitness value but is not appropriate at all.

Another problem which could be observed during the experiments was the steepness of the fitness function's funnel around the correct solution. If the individuals of the current generation are located outside of the funnel, the GA is unable to perform any directed search as the fitnesses have all about the same value. Then after some time the population gets attracted by a funnel and quite quickly gets sucked into it. However, as the solution space is strongly multimodal for most practically relevant problems (and also for the cryptanalysis of rotor machines) there are a lot of funnels. Consequently the probability is very high that the whole population converges to a solution that has no practical value, especially in the case of cryptanalysis as there is just one relevant solution (i.e. the correct key). This characteristic of the fitness landscape can be compared with the Shekel's Foxholes test function (as described in [Schoeneburg *et al.*, 1994]) which is considered to be almost unsolvable for GAs and other heuristic optimization techniques. Fig. 3 shows a 3D plot of the function.

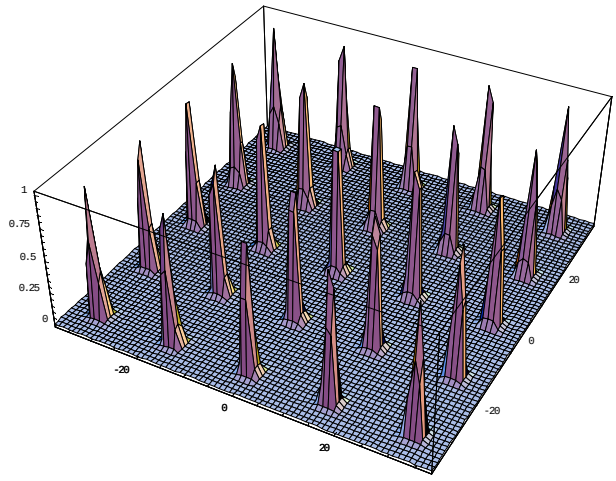

Figure 3: Test function Shekel's Foxholes

6 Conclusion

The main objective of this paper was to show with the concrete example of the cryptanalysis of a generic 3-rotor machine which modelling steps are required to be able to attack a given problem by a genetic algorithm. Additionally the different critical aspects of the modelling process were highlighted that may lead to a failure of the approach. These essential steps and the related traps and dangers can be summarized in the following enumeration:

1. **Problem Specification**
   Before any GA-specific modelling can be done a

precise problem specification is needed. The user must have a clear picture about the problem itself, the given information and the optimization objective. Otherwise there is naturally a high danger to go into totally wrong and not target-oriented directions.

2. **Encoding**
   The next and very important part is to choose a proper encoding for the Genetic Algorithm. This step is very much connected with the next one, as the encoding strongly determines the useable crossover and mutation operators. To keep the model as simple as possible the encoding should be intuitive and it should be absolutely clear what the relevant genetic information is. However, this is often very hard to do in real-world applications.

3. **Genetic Operators**
   After having specified the encoding, genetic operators for crossover and mutation can be developed. Thereby it is most necessary that the user knows what the relevant genetic information is. The crossover operator should be designed to maintain and recombine this information in an advantageous way. It is very important that the genetic material of the generated children is as identical to the parents as possible. Each deviance leads to unwanted mutations and to a drifting of the GA towards random search.

4. **Fitness Function**
   Finally an appropriate fitness function has to be used. As we have seen in the considered example it is absolutely fatal, if the fitness value is not directly correlated with the solution's suitability. I.e. if a solution has a better fitness value than another one, this must indicate that the fitter solution also represents a more suitable solution according to the specific problem. Otherwise the GA will be able to detect very fit solutions but these will not be useful at all.

References

[Bagnall, 1996] A. J. Bagnall. The applications of genetic algorithms in cryptanalysis, 1996.

[Clark, 1998] A. J. Clark. *Optimisation Heuristics for Cryptology.* PhD thesis, Information Security Research Center, Faculty of Information Technology, Queensland University of Technology, 1998.

[Davis, 1991] L. Davis. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, 1991.

[Goldberg, 1989] D. E. Goldberg. *Genetic Algorithms in Seach, Optimization and Machine Learning.* Addison Wesley Longman, 1989.

[Holland, 1975] J. H. Holland. *Adaption in Natural and Artificial Systems.* University of Michigan Press, 1975.

[Larranaga *et al.*, 1999] P. Larranaga, C. M. H. Kuijpers, R. H. Murga, I. Inza, and D. Dizdarevic. Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review*, 13:129–170, 1999.

[Michalewicz, 1996] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer Verlag, Berlin Heidelberg New York, 3rd edition, 1996.

[Oliver *et al.*, 1987] I. M. Oliver, D. J. Smith, and J. R. C. Holland. A study of permutation crossover operators on the TSP. In J. J. Grefenstette, editor, *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference*, pages 224–230. Lawrence Erlbaum, Hillsdale, New Jersey, 1987.

[Schoeneburg *et al.*, 1994] E. Schoeneburg, F. Heinzmann, and S. Feddersen. *Genetische Algorithmen und Evolutionsstrategien - Eine Einfuehrung in Theorie und Praxis der simulierten Evolution.* Addison-Wesley, 1994.

[Spillman *et al.*, 1993] R. Spillman, M. Janssen, B. Nelson, and M. Kepner. Use of a genetic algorithm in the cryptanalysis of simple substitution ciphers. *Cryptologia*, 17(1):31–44, 1993.

[Tomassini, 1995] M. Tomassini. A survey of genetic algorithms. *Annual Reviews of Computational Physics*, 3:87–118, 1995.