A New Heuristic Based on the Cuckoo Search for Cryptanalysis of Substitution Ciphers

Ashish Jain[1(✉)] and Narendra S. Chaudhari[1,2]

[1] Discipline of Computer Science and Engineering,
Indian Institute of Technology Indore, Indore, India
phd11120101@iiti.ac.in
[2] Visvesvaraya National Institute of Technology Nagpur, Nagpur, India
nsc0183@gmail.com

Abstract. The efficient utilization of one of the latest search heuristic, namely, cuckoo search for automated cryptanalysis (or attack) of substitution ciphers is addressed. A previously proposed genetic algorithm based attack of the simple substitution cipher is enhanced. A comparison is obtained between various attack algorithms that are based on cuckoo search, genetic algorithm, enhanced genetic algorithm, tabu search and scatter search where cuckoo search shows the best performance. It is worth pointing out that the proposed cuckoo search algorithm provides a valid and efficient option for solving similar permutation problems.

Keywords: Metaheuristics · Cuckoo search · Genetic algorithm · Tabu search · Scatter search · Cryptanalysis · Substitution ciphers

1 Introduction

The goal of the cryptanalyst is to systematically recover the original text (plaintext) and/or key by mounting an attack on the cipher. The attack may involve several ciphertexts and/or some plaintexts, intelligent mathematical computer algorithms and usually some luck. A ciphertext-only attack is one where the objective of the adversary (or cryptanalyst) is to recover the plaintext and/or deduce decryption key by observing only the "known ciphertext". This class of attack is most challenging and in this paper, we study such a cryptanalytic attack on a classical cipher (a most general form –mono-alphabetic (or simple) substitution cipher, hereinafter, substitution cipher). Although classical ciphers, namely, substitution and transposition ciphers were used hundreds of years ago, a particular interest in the study of these ciphers has been retained due to the fact that most of the modern cryptosystems still utilize the functions of these ciphers as their basic building blocks. For instance, the Advanced Encryption Standard (AES) used all over the world by finance community is based on the principle of the substitution-permutation network. Due to such important facts, the classical ciphers are normally considered as first choice when investigating new attack strategies such as the one discussed in this paper. In literature, several heuristics have been applied successfully for cryptanalysis of classical ciphers. For example,

S. Arik et al. (Eds.): ICONIP 2015, Part II, LNCS 9490, pp. 206–215, 2015.
DOI: 10.1007/978-3-319-26535-3_24

[1–8], where the main goal of researchers was to develop efficient cryptanalytic algorithms by utilizing the search (or optimization) heuristics.

1.1 Simple Substitution Cipher

In this type of cipher a key can be represented as a permutation of the plaintext alphabet (for example, if the plaintext alphabet consists of 26 English alphabets and the space character, then a permutation of these 27 alphabet characters usually forms a key). During the encryption process each letter of the plaintext message is replaced by the corresponding key element (i.e. ciphertext alphabet) that forms a ciphertext of length equal to the length of the plaintext message. The original plaintext message from the ciphertext is then recovered by intended recipient using decryption process. For details about this cipher we refer the interested reader to [4].

1.2 Our Contributions

It is surprising that in the last one decade no heuristic attack more efficient than the tabu search attack has been reported for breaking the substitution cipher. In this paper, we overcome the limitation by developing a new attack algorithm by utilizing one of the latest search heuristic, namely, cuckoo search. Moreover, for optimizing the cryptanalysis process we fine-tune some of the parameters of the cuckoo search. Additionally, two related and efficient attacks of the substitution cipher proposed by Clark [4] and are based on the genetic algorithm and tabu search are implemented. Furthermore, the genetic algorithm attack proposed by Clark [4] is enhanced by incorporating a new adaptive mutation operator. Although the scatter search based attack of the substitution cipher proposed by Garici and Drias [5] is not efficient (with respect to time complexity), it is being considered to obtain a fair comparison between previously proposed evolutionary heuristic based attacks and those presented in this paper.

2 Cryptanalysis of Substitution Ciphers

The main weakness of substitution ciphers is that their character frequency statistics (or n-grams) are not changed by the encryption algorithm. In other words, for every grouping of characters in the plaintext there is a distinct and corresponding grouping of characters in the ciphertext [4]. Therefore, most attacks of substitution ciphers attempt to match the n-grams of the encrypted text with those of known language (for example, English). In this paper, we use this fact as the basis to attack the substitution cipher and at the same time automate the search for the required n-gram frequencies by developing efficient algorithms based on the optimization heuristics.

2.1 Fitness Function or Cost

During the optimization process, each candidate key (or individual solution) is used to decrypt the known ciphertext and at the same time the n-gram statistics of the decrypted text is compared to the language statistics. In general, Eq. (1) can be used for comparison of these statistics.

$$Cost_k = \alpha \left(\sum_{i \in \mathscr{A}} |\mathcal{K}_i^u - \mathcal{D}_i^u| \right) + \beta \left(\sum_{i \in \mathscr{A}} |\mathcal{K}_i^b - \mathcal{D}_i^b| \right) + \gamma \left(\sum_{i \in \mathscr{A}} |\mathcal{K}_i^t - \mathcal{D}_i^t| \right), \quad (1)$$

where $\mathscr{A}$ denotes the language alphabet (e.g. in English language: A, B, ..., Z, _), and $\mathcal{K}$ and $\mathcal{D}$ denote statistics of the known language and decrypted text, respectively. The indices u, b and t denote the unigram, bigram and trigram statistics, respectively. Different weights in the range (0.0–1.0) with step size 0.1 can be assigned to α, β and γ. However, the following constraint must be satisfied to keep the number of combinations of α, β and γ workable.

$$\alpha + \beta + \gamma = 1.0. \quad (2)$$

Generally, in order to do more accurate evaluation of candidate keys, the n-grams should be higher in number. Also as per literature, the inclusion of trigram statistics in the fitness function are generally the most effective basis for cryptanalysis of classical ciphers [4]. However, for cryptanalysis of the substitution cipher the fitness function used in literature (for example, [1,4]) is purely based only on the bigrams. The reason is that the complexity associated with computation of trigram statistics is high (order of N^3, where N is the key size), while the benefit over the bigrams is minimal. Moreover, even though small amount of ciphertexts are known, the attack on the substitution cipher is more effective using fitness function which is based on the bigrams only than one which utilizes trigrams alone [4]. Due to these facts, the following fitness function which is purely based on the bigrams is used in this paper to attack the substitution cipher.

$$Cost_k = \sum_{i \in \mathscr{A}} |\mathcal{K}_i^b - \mathcal{D}_i^b|. \quad (3)$$

2.2 Cryptanalysis Using Genetic Algorithms

In this section, we enhance the genetic algorithm attack that was proposed by Clark (particularly, mutation operator). Note, the crossover (or mating) operator employed in Algorithm 1 is identical to that proposed by Clark. Due to lack of space, we do not present the crossover operator, but we refer readers to [4].

Mutation-I. This mutation operator is utilized in most of the previously reported attacks on the classical cipher. It is based on the simple way of perturbing a cipher key where two randomly selected elements of a key are swapped. In addition to this mutation operator, we use a similar mutation operator but an adaptive one which can be described as follows.

Algorithm 1. GA and Enhanced GA Attack (GA/EGA Attack)

1: **Initialization:** Randomly generate the initial population (call this population, $Current_{pop}$) containing m individuals, where individuals are keys of the substitution cipher. Evaluate the fitness of each individuals using Eq. (3).
2: **repeat**
3: Select $m/2$ pairs of individuals from the $Current_{pop}$. These pairs are referred to as parents of the new generation. Here we use a tournament selection method that choose a best individual (for each pairs of individuals) among five individuals, where the selection of these five individuals should be random.
4: Each pair of parents are then mated that produces two offspring. These m offspring form the new population abbreviated as New_{pop}.
5: **In case of GA-Attack: Apply Mutation-I, while in case of EGA-Attack: Apply Mutation-I and Mutation-II to each of the individual of New_{pop}.**
6: Evaluate the cost of each of the individuals exist in the New_{pop} using Eq. (3).
7: Merge $Current_{pop}$ with New_{pop} and in parallel remove duplicates. The merging process give a list of sorted (least cost to most cost) solutions, where the size of list will be between m and $2m$. The m best individuals from the merged list then become the new $Current_{pop}$.
8: **until** (Maximum-Iterations)
9: Output the best solution from the $Current_{pop}$.

Mutation-II. Interchange three randomly chosen elements of the key. However, this mutation operator is adapted when evolution starts to languish, i.e. no improvement in the solution is observed after some number of iterations. We limit this number to seven based on the experiments. The main benefit of the adaptive mutation operator is that it improves the rate of convergence and the success probability (in terms of full key recovery) of the genetic algorithm attack.

2.3 Cryptanalysis Using Cuckoo Search Combined with Lévy Flights

In 2009, Yang and Deb [9,10] have proposed a new nature-inspired population based metaheuristic known as cuckoo search via Lévy flights. "This metaheuristic is formed by inspiration from the obligate brood parasitic behavior of few cuckoo species in combination with Lévy flight behavior of some birds and fruit flies [9]". For simplicity, the standard cuckoo search method is described in Algorithm 2.

To generate a new nest (or a solution vector, see Algorithm 2, step 3) $\mathbf{x}_j(t+1)$ from an existing nest $\mathbf{x}_j(t)$, for, a cuckoo j; Lévy flight is performed as [9,11]:

$$\mathbf{x}_j(t+1) = \mathbf{x}j(t) + \mu. \tag{4}$$

The above equation is essentially a stochastic equation for a random walk. In general, a random walk is a Markov chain whose next state/location depends only on the current location (the first term) and the transition probability (the second term) [9]. In the second term, $\mu > 0$ is a step-size scaling factor which should be associated to the scales of the problem of interest [12]. The term l is the step-size

which is drawn from a probability distribution with a power law tail (also known as Lévy stable distribution) [13]. It is worth mentioning that the cuckoo search via random walk using Lévy flights is more efficient than other popular swarm intelligence techniques (e.g. PSO) in exploring the search space as its step-size distribution is pseudo-random [12]. Nonetheless, it is not a trivial task to produce pseudo-random steps that accurately obey the Lévy stable distribution. However, from the implementation aspects, Mantegna's [13] algorithm is one of the most efficient and yet straightforward method that generates a stochastic variable [12]. Note that the stochastic variable has probability density arbitrarily close to Lévy stable distribution characterized by an arbitrarily chosen control parameter ($0.3 \leq \lambda \leq 1.99$) [13]. Finally, the step-size l using Mantegna's algorithm can be calculated as [13]:

$$l = \frac{u}{|v|^{1/\lambda}}, \tag{5}$$

where u and v are two Gaussian stochastic variables with a zero mean and standard deviations of $\sigma_u(\lambda)$ and $\sigma_v(\lambda)$, respectively, where $\sigma_u(\lambda)$ and $\sigma_v(\lambda)$ can be given by Eq. (6). Here, $\Gamma(z)=\int_0^\infty t^{z-1}e^{-z}dt$ [13]. For detailed description on the standard cuckoo search interested reader can refer [9,11,12].

$$\sigma_u(\lambda) = \left[\frac{\Gamma(1+\lambda)\,\sin(\pi\lambda/2)}{\Gamma((1+\lambda)/2)\lambda 2^{(\lambda-1)/2}}\right]^{1/\lambda} = 0.696575 \;\text{ and }\; \sigma_v(\lambda) = 1(\text{if: } \lambda = 1.5), \tag{6}$$

Cuckoo Search Attack. The cuckoo search attack algorithm for cryptanalysis of the substitution cipher is intuitive (see Algorithm 3). Therefore, here we discuss only the mapping of main components of the cuckoo search (i.e. nest, egg and Lévy flights) to the problem under consideration. In most applications of the cuckoo search, often it is assumed that each nest has one egg. However, in case of cryptanalysis of the substitution cipher, we consider each nest has N distinct eggs/elements (i.e. N distinct characters of a key: n_1, n_2,..., n_n, where N=27 (i.e. A —Z and the space character). For simplicity, consider each egg has a unique number ($\in$ [1, N]) associated with it. That is there must be a unique

Algorithm 2. Standard Cuckoo Search [9]

1: Generate the initial population of m host nests $\mathbf{x}_i, i = 1, 2, ..., m$.
2: **repeat**
3: Get a cuckoo randomly (say, $\mathbf{x}_j$), and generate a new solution by Lévy flights
4: Compute its cost, let it be f_j
5: Randomly choose a nest among m host nests, say $\mathbf{x}_k$
6: **if** (f_j¡f_k) (Let the problem has minimization objective) **then**
7: Replace $\mathbf{x}_k$ by the new solution $\mathbf{x}_j$
8: **end if**
9: Abandon a fraction (p_a) of worst nests/solutions and construct new ones
10: Keep the best solutions; rank the solutions and find the current best
11: **until** (Termination condition is satisfied)
12: Post-process results

identity of each elements in the nest/solution. In other words, a key cannot have two similar characters. In this regard, the difficulty is how to preserve distinctness property of the key elements, since we can clearly observe that the Eq. (4) will destroy the distinctness property of the elements of the current solution during updation. Note that the importance of Eq. (4) is that it builds the new solution from an existing solution via Lévy flights which is an efficient approach, since the step-size is heavy tailed and any large step is possible. That is the existing solution has better chance to get changed to a better quality solution in lesser computations. Hence, we do not change this equation, rather we utilize its efficiency for improving existing solutions using the current best solution and new solutions generated by Eq. (4) (see Algorithm 3, step 7–11).

Algorithm 3. Cuckoo Search Attack (CS Attack)

1: **Initialization:** Randomly generate the initial population of m host nests. Call this population, POP_{nests}, where host nests are keys of the substitution cipher.
2: **repeat**
3: Evaluate the cost of each nest of the POP_{nests} using Eq. (3). Select the nest form the POP_{nests} that has the lowest cost. Call this nest $BEST_{nest}$.
4: Choose a nest randomly (say, POP^i_{nests}), and generate a new nest/solution (abbreviated as NEW_{nest}) by the Lévy flights as: NEW_{nest}=POP^i_{nests}+μ l, where l is computed using Eq. (5) with μ=0.01 and λ=1.5.
5: Evaluate the cost of NEW_{nest} using Eq. (3). If the cost of NEW_{nest} is better (lower) than $BEST_{nest}$ then it become the new $BEST_{nest}$. If it is then go to step 4, otherwise, update the i^{th} existing solution (i.e. POP^i_{nests}) by using NEW_{nest} and $BEST_{nest}$ as follows.
6: Generate a random number in the range [1,N], call this number n.
7: **repeat**
8: Find the next egg/element in the NEW_{nest} that has the identity less than n. Call the position of this element P_1. (Note: in case of first iteration, read first instead of next).
9: Examine the identity of the element which is located at position P_1 in the $BEST_{nest}$. Call this identity n_1.
10: Let P_2 is the position in POP^i_{nests} where the element of identity n_1 is located. Swap those elements of the POP^i_{nests} that are located at positions P_1 and P_2. As obvious this operation will store the element of identity n_1 at position P_1, since P_1 is the best position of the same identity element in the $BEST_{nest}$. Here we emphasized that this swapping operation is performed in order to preserve the element that have already placed in POP^i_{nests} to its best position.
11: **until** (NEW_{nest} list has been traversed completely)
12: If $BEST_{nest}$ and updated POP^i_{nests} are identical then swap the eggs that are located at two different positions in POP^i_{nests}; positions are determined randomly.
13: Abandon a fraction (p_a) of worst nests. Create new nests as replacement of abandon nests. The new nests are created again from the best nest by swapping its two randomly chosen elements. In the experiment, we fixed p_a=0.01.
14: **until** (Maximum-Iterations)
15: Output the best solution from the POP_{nests}

2.4 Cryptanalysis Using Tabu Search

The main focus of this optimization algorithm [4,14] is to provide a heuristic for searching a good solution to the problem under consideration without becoming trapped in a local optima. This algorithm maintains a tabu list as a short term memory list where at each iteration the current key is added to the tabu list, and the key remains 'tabu' for a fixed number of iterations. An intuitive level heuristic for cryptanalysis of the substitution cipher is shown in Algorithm 4. Here, we mention that the choice of N_{poss} parameter must be less than $N(N\text{-}1)$, since this is the maximum number of distinct keys which can be created from the best key of the tabu by swapping two elements [4], where N is the key size (e.g. 27). For a detailed description of this heuristic we refer readers to [3,4].

Algorithm 4. Tabu Search Attack (TS Attack) [4]

1: **Initialization:** Randomly initialize the tabu list containing m keys of the substitution cipher. Set N_{poss}, the size of the possibilities list.
2: **repeat**
3: Select the key form the tabu list that has the lowest cost. Call this key $TABU_{best}$.
4: **for** i = 1,2,..., N_{poss} **do**
5: Choose n_1, $n_2 \in$ [1, N], such that $n_1 \neq n_2$.
6: Construct a new key K_{new} by swapping the elements n_1 and n_2 of $TABU_{best}$.
7: Ensure that K_{new} is not already in the possibilities list or the tabu list. If it is return to Step 5.
8: Add K_{new} to the possibilities list and determine its cost.
9: **end for**
10: From the possibilities list find the key of lowest cost. Call this key $POSS_{best}$.
11: From the tabu list find the key of highest cost. Call this key $TABU_{worst}$.
12: **while** the cost of $POSS_{best}$ is less than the cost of $TABU_{worst}$ **do**
13: Replace $Tabu_{worst}$ with $POSS_{best}$ and find the new $POSS_{best}$.
14: Find the new $TABU_{worst}$.
15: **end while**
16: **until** (Maximum-Iterations)
17: Output the best solution from the tabu list.

3 Parameters, Experimental Setup and Results

The inputs to all the above presented algorithms are: known ciphertext, its length and bigram statistics of the language (which are assumed to be known). The output of each algorithm is either full or partial substitution cipher key. Note that in order to recover the message that is readable, it is not essential to recover every element of the key, i.e. considerable partial key recovery is also significant to understand the message. The parameters such as m (i.e. population/tabu-list size), Maximum-Iterations (i.e. maximum number of iterations) and N_{poss} (i.e. size of possibilities list in case of tabu search) were fine-tuned by a combination of several experiments. Note that the fine-tuning was performed separately for each of the algorithms in order to optimize the cryptanalysis process.

In some scenarios guidelines are helpful, for example in Algorithm 3, μ=0.01 and λ=1.5 are taken that has been reported in [10] as general choice. Furthermore, in order to obtain a clear comparison between above algorithms, the guidelines reported in [4] followed, i.e. three criteria were used: amount of known ciphertext available for the attack, number of keys examined prior to determination of correct solution and the time needed to find the correct solution.

Now, we discuss the experimental setup and their details. We tested each of the algorithms on 100 different messages. For each message, each of the cryptanalytic algorithms was run 3 times (it comes to a total of 300 times) and the best of the 3 was recorded. Afterwards, 100 best recorded results corresponding to each algorithm were then averaged. The above described process of recording and averaging is repeated for known ciphertext of size 100 to 800 with step size of 100. The average results of each of the algorithms are then plotted in Fig. 1. From Fig. 1, we can clearly observe that each of the attack algorithms perform well and comparably. Nevertheless, the best way to identify the proper efficiency of the approximation algorithms is; to compare them on the basis of two factors: state space searched (i.e. number of keys examined before evolving the best result) and complexity of the attack (i.e. performance time). For this purpose, we tested each of the attack algorithms on the 100 different known ciphertext of length 1000 characters and then recorded the amount of keys examined and the time taken by the attack. The average results are shown in Table 1. Note that the scatter search method has not been considered in Table 1 (and not re-implemented in this paper), since Garici and Drias (investigators of this algorithm) have concluded that the scatter search method takes 75 % more time than the genetic algorithms (please see Sect. 4.2 in [5]), while the quality of the results is only 15 % better. From Table 1, we can clearly see that the mean performance time of enhanced genetic algorithm is comparatively lesser than the genetic algorithm proposed by Clark, while the average number of keys examined is approximately equal. From Table 1, we can note that the tabu search is more efficient in both respects than genetic algorithms. However, it is also clear from the table that the cuckoo search outperforms tabu search in both respects. Most importantly, the overall performance of cuckoo search is much better among all the attack algorithms (including mean and standard deviation of the key elements correctly found).

Table 1. A comparison based on: mean performance time (T_{mean}), average number of keys examined before the best solution found (M_{avg}) and number of key elements correctly found–mean ($\bar{X}$) & standard deviation (S).

Method →	CS Attack	TS Attack	EGA Attack	GA Attack
T_{mean} (in sec.)	**0.137**	0.241	0.397	0.416
M_{avg}	**1823**	3306	3707	3695
$\bar{X}$	**25.4**	24.05	24.3	24.1
S	**1.04**	1.52	1.23	2.04

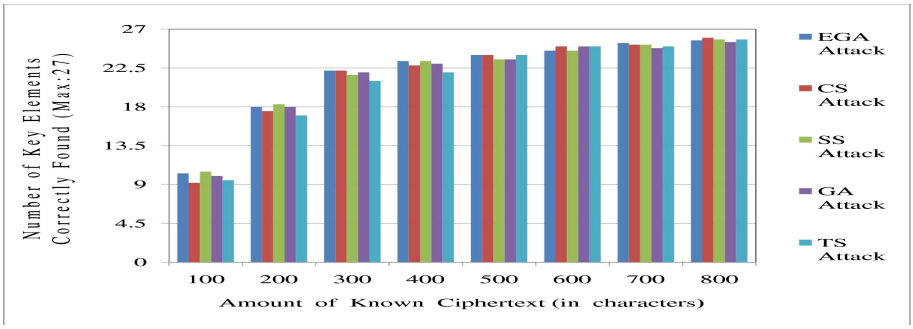

Fig. 1. A comparison based on the amount of known ciphertext. (In case of scatter search (SS) attack, the results were taken from Fig. 3 of [5] which may be not exact.)

4 Conclusion

This paper presents various attacks on the substitution cipher where genetic algorithms, tabu search and cuckoo search have been utilized. We examined that the adaptive mutation operator along with appropriate selection procedure improves the performance of previously proposed genetic algorithm attack. It is worth pointing out that the cuckoo search attack has shown the best performance among all attacks. Most importantly, we needed to fine-tune lesser number of parameters for cuckoo search than genetic algorithms and tabu search. This study indicates that the developed attack algorithm (which utilizes the cuckoo search) is able to produce results that are clearly better than previous attack algorithms of the substitution cipher, and therefore it can be used as a valid and efficient alternative for solving this kind of permutation problems.

References

1. Forsyth, W.S., Safavi-Naini, R.: Automated cryptanalysis of substitution ciphers. Cryptologia **17**(4), 407–418 (1993)
2. Spillman, R., Janssen, M., Nelson, B., Kepner, M.: Use of a genetic algorithm in the cryptanalysis of simple substitution ciphers. Cryptologia **17**(1), 31–44 (1993)
3. Clark, A.: Modern optimisation algorithms for cryptanalysis. In: Proceedings of the 1994 Second Australian and New Zealand Conference on Intelligent Information Systems, pp. 258–262. IEEE (1994)
4. Clark, A.J.: Optimisation heuristics for cryptology. Ph.D. thesis (1998)
5. Garici, M.A., Drias, H.: Cryptanalysis of substitution ciphers using scatter search. In: Mira, J., Álvarez, J.R. (eds.) IWINAC 2005. LNCS, vol. 3562, pp. 31–40. Springer, Heidelberg (2005)
6. Song, J., Yang, F., Wang, M., Zhang, H.: Cryptanalysis of transposition cipher using simulated annealing genetic algorithm. In: Kang, L., Cai, Z., Yan, X., Liu, Y. (eds.) ISICA 2008. LNCS, vol. 5370, pp. 795–802. Springer, Heidelberg (2008)
7. Cowan, M.J.: Breaking short playfair ciphers with the simulated annealing algorithm. Cryptologia **32**(1), 71–83 (2008)

8. Boryczka, U., Dworak, K.: Genetic transformation techniques in cryptanalysis. In: Nguyen, N.T., Attachoo, B., Trawiński, B., Somboonviwat, K. (eds.) ACIIDS 2014, Part II. LNCS, vol. 8398, pp. 147–156. Springer, Heidelberg (2014)
9. Yang, X.S., Deb, S.: Cuckoo search via lévy flights. In: World Congress on Nature & Biologically Inspired Computing, 2009, NaBIC 2009, pp. 210–214. IEEE (2009)
10. Yang, X.S., Deb, S.: Engineering optimisation by cuckoo search. Int. J. Math. Model. Numer. Optimisation **1**(4), 330–343 (2010)
11. Yang, X.S., Cui, Z., Xiao, R., Gandomi, A.H., Karamanoglu, M.: Swarm Intelligence and Bio-Inspired Computation: Theory and Applications. Elsevier, Waltham (2013)
12. Yang, X.S.: Nature-Inspired Optimization Algorithms. Elsevier, Amsterdam (2014)
13. Mantegna, R.N.: Fast, accurate algorithm for numerical simulation of levy stable stochastic processes. Phys. Rev. E **49**(5), 4677–83 (1994)
14. Glover, F.: Tabu search: a tutorial. Interfaces **20**(4), 74–94 (1990)