Cryptanalysis of Substitution Ciphers Using Scatter Search

Habiba DRIAS[1] and Mohamed Amine GARICI[2]

[1] Institut National d'Informatique, Oued Smar, Algiers, Algeria

[2] Electronics and Computer Science Faculty
Research Laboratory in Artificial Intelligence, LRIA-USTHB
BP 32 El-Alia Bab-Ezzouar, 16111, Algiers, Algeria
Tel : (+213) 21247892 - Fax: (+213) 21247892
Drias@wissal.dz

Abstract. This paper presents an approach for the automated cryptanalysis of substitution ciphers based on a recent evolutionary metaheuristic called Scatter Search. It is a population-based metaheuristic founded on a formulation proposed two decades ago by Fred Glover. It uses linear combinations on a population subsets to create new solutions while other evolutionary approaches like genetic algorithms resort to randomization.
First, we implement the procedures of the scatter search for the cryptanalysis of substitution and transposition ciphers. This implementation can be used as a framework for solving permutation problems with scatter search. Then, we test the algorithm and show the importance of the improvement method and the contribution of subset types. Finally, we compare its performances with those of a genetic algorithm.

Key-Words. automated cryptanalysis, substitution and transposition ciphers, scatter search, evolutionary approach, heuristic search, genetic algorithm, optimization problem.

1. Introduction

Cryptology is the science and study of systems for secret communications. It consists of two complementary fields: cryptography and cryptanalysis. Cryptography is the science of making communications unintelligible to everyone except the intended receiver. At its opposite, there is the cryptanalysis whose aim is breaking ciphers, i.e. to recover the plaintext from the ciphertext without knowing the decryption key.

Simple ciphers were first used hundreds years ago. A particular interest is carried to this kind of systems because most of the modern cryptosystems use operations of the simple ciphers as their building blocks. Many ciphers have a finite key space and, hence, are vulnerable to an exhaustive key search attack. Yet, these systems remain secure from such an attack because the key space size is such that the time and resources for a search are not available. Thus, automated reasoning tools can be used to perform attack against this systems. Many researches showed that a range of modern-day cryptological problems can be attacked successfully using metaheuristic search [3].

Many automated attacks have been proposed in the literature for cryptanalysing classical ciphers. Previously, Spillman and al. [12] have published an attack on the simple substitution cipher using a genetic algorithm, Forsyth and Safavi-Naini [6] presented an attack using simulated annealing. Tabu search was also used in [4]; and recently, Mathews and al. [11] used ants (ACO) to attack this ciphers.

The evolutionary population based approach, the scatter search has been introduced recently as a metaheuristic for solving complex optimization problems [9,10]. It is based on a formulation for integer programming developed in 1977 by Fred Glover [7] and uses linear combination of a population subset to create new solutions. It could be viewed as a bridge between taboo search and genetic algorithms [8]. It has been recently applied with success to a number of combinatorial optimization problems, for instance, the linear ordering problem [2] and the satisfiability problem (SAT) [5].

In this paper, scatter search is used to attack classical ciphers. First, we implement the procedures of the scatter search for the cryptanalysis of this class of ciphers. Since cryptanalysis of simple ciphers is a permutation problem, then this implementation can be used as a framework for solving permutation problems with scatter search. After, we test the method and we show the importance of the improvement method and the contribution of subset types. Finally, we compare its performances with those of a genetic algorithm.

1.1 Simple ciphers

There are several variants of substitution ciphers, the ones used here are the most general form (mono-alphabetic substitution. A detailed description of these ciphers is given in[4]. In simple substitution ciphers, each symbol in the plaintext is replaced by another symbol in the ciphertext. A substitution cipher key can be represented as a permutation of the plaintext alphabet symbols. The main propriety of this kind of ciphers is that the *n*-grams statistics are unchanged by the encryption procedure.

2. A General Overview of the Scatter Search

Basically, the scatter search method starts with a population of good and scattered solutions. At each step, some of the best solutions are extracted from the collection to be combined and included in a set called the reference set. A new solution is then obtained as a result of applying a linear combination on the extracted solutions. The quality of the new solution is then enhanced by an improvement technique such as a local search. The final solution will be included in the reference set if it presents interesting characteristics with regards to the solution quality and dispersion.
Although it belongs to the population-based procedures family, scatter search differs mainly from genetic algorithms by its dynamic aspect that does not involve randomization at all. Scatter search allows the combination of more than two solutions, it gets thus at each step more information. By combining a large number of solutions, different sub-regions of the search space are implicated to build a solution. Besides, the reference set is modified each time a good solution is encountered and not at the combination process termination. Furthermore, since this process considers at least all pairs of solutions in the reference set, there is a practical need for keeping the cardinality of the set small (<=20). The scatter search can be summarized in a concise manner as follows:

Generate an initial population *P*
while not Stop-Condition **do**
- Initialize the reference set with the solutions selected to be combined
- Generate new solutions by applying the combination process
- Improve new solutions quality
- Insert new solutions in population with respect to quality and dispersion criteria

end while

The procedure stops as in many metaheuristics, when during a small number of iterations no improvement in solutions quality is recorded or when we reach a certain number of iterations limited by physical constraints.

The fact that the mechanisms within scatter search are not restricted to a single uniform design allows the exploration of strategic possibilities that may prove effective in a particular implementation. These observations and principles lead to the following template for implementing scatter search[9]:

- A *Diversification Generation Method* to generate a collection of diverse trial solutions, using an arbitrary trial solution (or seed solution) as an input.
- An *Improvement Method* to transform a trial solution into one or more enhanced trial solutions.
- A *Reference Set Update Method* to build and maintain a *reference set* consisting of the *b* "best" solutions found (where the value of *b* is typically small, e.g., no more than 20), organized to provide efficient accessing by other parts of the method. Solutions gain membership to the reference set according to their quality or their diversity.
- A *Subset Generation Method* to operate on the reference set, to produce a subset of its solutions as a basis for creating combined solutions.
- A *Solution Combination Method* to transform a given subset of solutions produced by the subset generation method into one or more combined solution vectors.

2.1 The Reference Set

The utility of the reference set *RefSet* consists in maintaining the *b* best solutions found in terms of quality or diversity, where *b* is an empirical parameter. *RefSet* is partitioned into $RefSet_1$ and $RefSet_2$, where $RefSet_1$ contains the b_1 best solutions and $RefSet_2$ contains the b_2 solutions chosen to augment the diversity. The distance between two solutions is defined to measure the solutions diversity. We compute the solution that is not currently in the reference set and that maximizes the distance to all this solutions currently in this set.

2.2 The Subset Generation Method

The solution combination procedure starts by constituting subsets from the reference set that have useful properties, while avoiding the duplication of subsets previously generated. The approach for doing this, consists in constructing four different collections of subsets, with the following characteristics:

- Subset-Type 1: all 2-element subsets.
- Subset-Type 2: all 3-element subsets derived from the 2-element subsets by augmenting each 2-element subset to include the best solution not in this subset.
- Subset-Type 3: all 4-element subsets derived from the 3-element subsets by augmenting each 3-element subset to include the best solution not in this subset.
- Subset-Type 4: the subsets consisting of the best *i* elements, for *i*=5 to *b*.

The experiments described in [1] showed that at least 80% of the solutions that were admitted to the reference set came from combinations of type-1 subsets, but this should not be interpreted as a justification for completely disregarding the use of combinations other than those from type-1 subsets.

2.3 The Solution Combination

Scatter search generates new solutions by combining solutions of *RefSet*. Specifically, the design of a combination method considers the solutions to combine and the objective function. A new solution replaces the worst one in $RefSet_1$ if its quality is better. In the negative, the distances between the new solution and the solutions in *RefSet* are computed. If diversification is improved, the new solution replaces the element of $RefSet_2$ that has the smallest distance. Otherwise, it is discarded.

3. The design of Scatter search for the cryptanalysis of substitution ciphers

In our case, a solution is a cipher key. A key is a permutation of the plaintext's alphabet. The alphabet's characters are ordered according to the decreasing order of their standard frequency; e.g., in English this order is (_,e,t,a,o,n,h,i,s,r,d,l,u,m,w,g,y,c,f,b,p,k,v,x,j,q,z). The reason for this ordering will become apparent when generation method and combination method are presented.

Before implementing scatter search's methods, we must define two basic notions of the scatter search: how to estimate solution's fitness and distance between two given solutions. This last measure is a typical characteristic of the scatter search.

3.1 The fitness function

To estimate the fitness of a given solution, this solution is used to decrypt the intercepted ciphertext, then we calculate the difference between *n*-gram statistics of the decrypted text with those of the language assumed known.

$$f_C(K) = \sum_{j=1}^{max\_ngram} \left(\alpha_j \cdot \sum_{i_1,...,i_j \in \mathsf{A}} \left| P(i_1,...,i_j) - C(i_1,...,i_j) \right| \right) \quad (1)$$

where:

- A: The plaintext's alphabet.
- α_j: constants which allow assigning of different weights to each *n*-gram, and $\sum \alpha_j = 1$.
- $P(i_1,...,i_n)$: standard frequency of the *n*-gram $(i_1,...,i_n)$.
- $C(i_1,...,i_n)$: frequency of the *n*-gram $(i_1,...,i_n)$ in the decrypted message.

All attacks on classical ciphers *max_ngram* = 3, i.e. the *n*-grams are restricted to unigrams, bigrams and trigrams. Equation (1) provides an estimation for the distance between frequencies of decrypted text's *n*-grams and frequencies of the plaintext's language, many keys can provide the optimum value, or the authentic key don't give the optimum value. For this, we'll use another heuristic called '*Word*', which is more time-consuming. It estimates the number of correct words in the decrypted text.

$$Word(P) = \frac{1}{L} \sum_{M_P^l \subset P} \left(l \times Recognized\_Word\left(M_P^l\right) \right) \tag{2}$$

$$Recognized\_Word\left(M_P^l\right) = \begin{cases} 1\ ; & \text{if } M_P^l \in \text{Dictionary} \\ 0\ ; & \text{else} \end{cases} \tag{3}$$

where:

- M_P^l : a word belonging to the text *P* whose length is *l*.
- *L*: length of the text *P*.

Formula (2) estimates the ratio of the sum of recognized word's lengths on the total text's length. The use of this function is restricted to evaluate solutions newly inserted in *RefSet* at each iteration, and therefore to stop the search if a suitable value is reached. Formula (1) will be used when an evaluation of the solution's quality is required in the scatter search's methods.

3.2 The distance measure

The way to evaluate the distance between two solutions is an important element of the scatter search, because the diversification aspect is essentially based on this measure. We defined the distance between two given solutions $p = (p_1, p_2, ..., p_n)$ and $q = (q_1, q_2, ..., q_n)$ as follow:

$$d(p,q) = \text{number of permutations of } p_i \text{ et } p_{i+1} \ (1 \le i \le n-1) \text{ to obtain } p = q \tag{4}$$

For example: the distance between $s_1 = (1,2,3,4)$ and $s_2 = (2,1,4,3)$ is 2.

$$s_2 = (\underline{2,1},4,3) \rightarrow (1,2,\underline{4,3}) \rightarrow s_1 = (1,2,3,4) \ \Rightarrow \ d(s_1, s_2) = 2$$

3.3 The improvement method

This method consists of a simple local search procedure exploring the solution's neighbourhood. In this context, a neighbouring solution is a solution obtained by permuting two neighbouring elements of the current solution. The research of the best improvement for all solutions is very expensive, therefore this implementation is restricted to explore the first improvement and stops when a local optimum is found or after a fixed number of iterations.

3.4 The diversification generation method

Our method generates the initial population by two different approaches, where each one generates a part of the population, all generated solutions are improved by the previous method before being inserted in the initial set *P*.

- The first generator uses an existing solution of good quality (seed solution) and browses its neighbourhood (solutions being to a small distance of this solution but with avoiding the immediate neighbours since those will be browsed by the improvement method). The seed solution can be reached by a previous resolution tentative, or according to a heuristic like this one: order characters according to the decreasing order of their apparition frequencies in the ciphertext, it permits to minimize the difference of unigrams frequencies.
- The second generator employs controlled randomised process drawing upon frequency memory to generate a set of diverse solutions.

3.5 The solution combination method

This method –like the improvement method– is a problem-specific mechanism, since it is directly related to the solution representation. The adopted method uses a vote mechanism, it browses each solution to combine in a left to right direction, and the new solution is constructed element by element: at each step the vote mechanism determines the following element to add. The number of voices granted by a solution to its element depends on the position of this element in this solution. For example, An element being in the 1st position of a solution, and after 3 iterations not appearing again in the constructed solution will receive 3 voices of its solution.

Procedure *Combination method* (S_1,…,S_p : solution) : Solution ;
Var
Vote : Array [1..*N*] of integer;
F : Array [1..*N*] of integer;
OldElement : Array [1..*p*] of lists of characters ;

Begin
For *i* =1 **to** *N* **do**
begin
For *j* =1 **to** *p* **do**
if $S_j[i] \notin \{ S_{New}[1],\ldots, S_{New}[i\text{-}1]\}$ **then**
begin
find $k \setminus S_j[i] = a_k$; // $S_j[i]$ is the k^{th} alphabet character
Vote[*k*]++ ;
if $f(S_j) > F[k]$ **then** $F[k] = f(S_j)$;
end ;
find $m \setminus Vote[m] = \underset{j=1}{\overset{N}{Max}}(Vote[j])$ and $F[m] = \underset{j=1}{\overset{N}{Max}}(F[j])$;
$S_{New}[i] = a_m$;
Vote[*m*] = 0 ; *F*[*m*] = 0 ; //don't consider this element at the next votes
For *j* =1 **to** *p* **do**
begin
Delete a_m from the list *OldElement*[*j*] ;
For (each element *c* in *OldElement*[*j*]) **do** *Vote*[*k*]++ \ $c = a_k$;
if $S_j[i] \notin \{ S_{New}[1],\ldots, S_{New}[i]\}$ **then** Add $S_j[i]$ in the list *OldElement*[*j*] ;
end ;
end ;
Return (S_{New}) ;
End ;

where:
- *Vote* : contains the vote scores.
- *F* : contains for each element, the maximal fitness value obtained by its solutions.
- *OldElement* : contains for each solution, a list of its not elected elements.
- (a_1,…,a_N) : the alphabet.
- $S_j[i]$: denotes the i^{th} element of the j^{th} solution.

3.6. The overall procedure

The scatter search procedure can be summarized as follow:

Procedure *SS-Cryptanalysis* (*Seed*: Solution) : Solution ;
Begin
  For *Iter* = 1 **to** *MaxIter* **do**
    begin
    P = *GenerationMethod* (*Seed*,Pop_Size);
    RefSet(*P*,b_1,b_2);
    For each Solution $S \in RefSet_1$ **do If** (*Word*(*S*)>MinValue) **then** *Return*(*S*) **and** Stop;
    NewElements = TRUE; *Stop* = FALSE;
    While (*NewElements* **and not** *Stôp*) **do**
      begin
      NewElements = FALSE;
      GenerationSubsets(*RefSet*);
      For each Subset (S_1,…, S_p) **do**
        begin
        S_{new} = *CombinationMethod*(S_1,…, S_p);
        *S** = *ImprovementMethod*(S_{new});
        If ($S^* \notin RefSet$ **and** $\exists$ S$\in RefSet_1$/ OV(*S**)>OV(*S*)) **then** // OV is the evaluation function
            Begin replace *S* by *S**; *NewElements* = TRUE; **end**
        Else If ($S^* \notin RefSet$ **and** $\exists$ S$\in RefSet_2$/ dmin(*S**)>dmin(*S*)) **then**
            Begin replace *S* by *S**; *NewElements* = TRUE; **end;**
        end;
      For each Solution $S \in RefSet_1$ **do If** (*Word*(*S*)>MinValue) **then** *Return*(*S*) **and** *Stop* = TRUE;
      end;
    if (*Iter* < *MaxIter*) **then** *Seed* = the best solution of $RefSet_1$;
    end;
  Return(the best solution of $RefSet_1$);
End;

4. Experiments

The cryptanalysis procedure has been implemented in Pascal on a personal computer. First, numerical tests were carried out to set the parameters of the scatter search algorithm. In a second steps, we performed experiments in order to evaluate method's performances and to compare them with those of a genetic algorithm. The used plaintexts become from various texts (articles, classics) chosen at random and of total size adjoining 10 Millions of characters. Standard frequencies have been calculated from these texts.

5.1 Setting the algorithm's parameters

The effectiveness of the *n*-grams types. The aim of this experiment is to evaluate the effectiveness of each one of the *n*-grams types. We evaluated the average number of key elements correct with varying values of the constants α_j in equation (1). We applied the following restriction to the *n*-gram's weights followed in order to keep the number of combinations of the constants α_1, α_2 and α_3 workable.

- $\alpha_1, \alpha_2, \alpha_3 \in \{0\ ;\ 0{,}1\ ;\ 0{,}2\ ;\ 0{,}3\ ;\ 0{,}4\ ;\ 0{,}5\ ;\ 0{,}6\ ;\ 0{,}7\ ;\ 0{,}8\ ;\ 0{,}9\ ;\ 1\}$
- $\alpha_1 + \alpha_2 + \alpha_3 = 1$

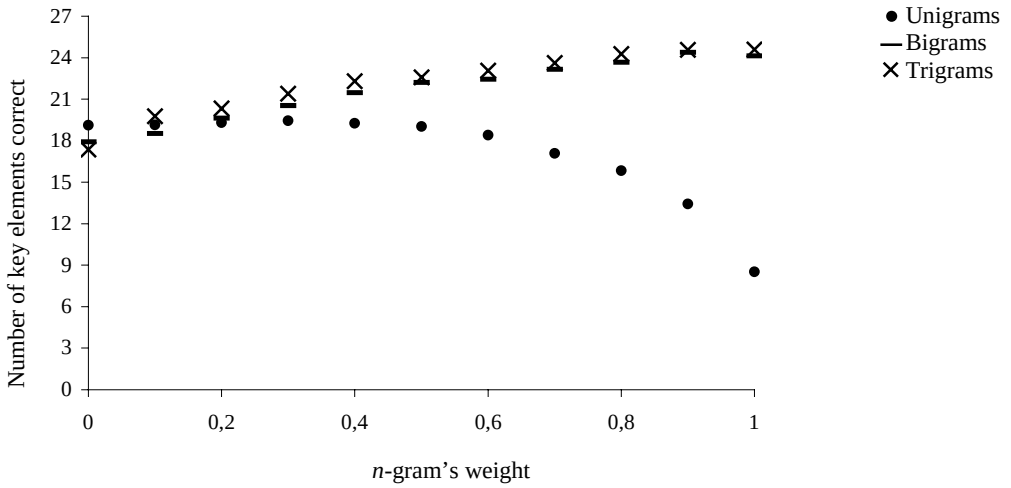

Figure 2: The search's results with varying weights of *n*-grams

Figure 1 shows that a fitness function using bigrams or trigrams have better results than the one using unigrams, with a small advantage for the trigrams. But the profit obtained of the grams doesn't compensate the necessary resources for their use. A similar result is found in [4].

Contribution of the Subset's types. In this experiment, we determine types of subsets that contribute best in the generation of reference solutions, and thereafter, to eliminate types of subsets that seem inert. We calculated number of necessary algorithm's iterations to find the best solution for each subset type and for every possible combination of subset types. The best combination is subset type 2 and subset type 3.

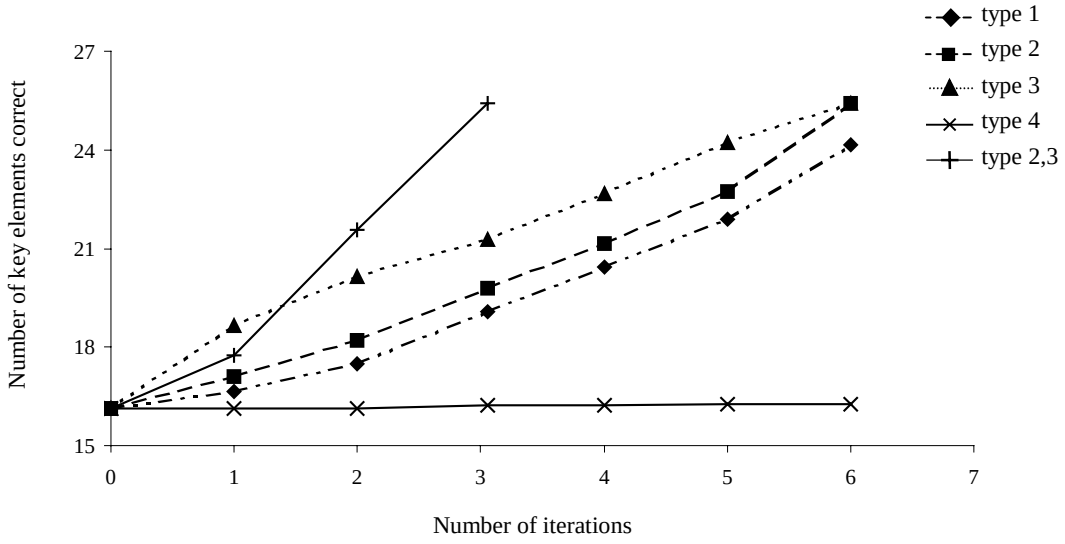

Figure 3: Number of necessary iterations to find the best solution for each subset type

Most reference solutions are generated by the combination of solutions of subsets of type 2 and type 3, contrary to most of the scatter search's implementations for other problems[1]. This difference can be explained by the combination method mechanism: high quality solutions are often close, and in most cases the combination of those solutions returns one of the input solutions. But when combination method have 3 or 4 input solutions, it works better and returns new solutions of high quality. It's clear that using all subset types will give the best quality, but using only type 2 and 3 will give almost the same quality and reduce significantly necessary time for the execution.

Remaining Parameters. After extensive experiments, the following values yield solution of high quality:

- Maximum number of iterations of the scatter search: *MaxIter* = 7.
- Reference set size: $b = 10$ (b_1=5, b_2=5).
- Initial set size: *Pop_Size* = 120.

5.2 Computational results

In this section, we present results concerning performances of the cryptanalysis procedure and a comparison with a genetic algorithm. For the genetic algorithm implementation, we used parameters values presented in [12]: population size = 1000 and mutation rate = 0,1. The crossover and mutation operators are similar to those presented in [4].
The following curve shows the average number of correct elements in the recovered keys according to the size of the text encoded.

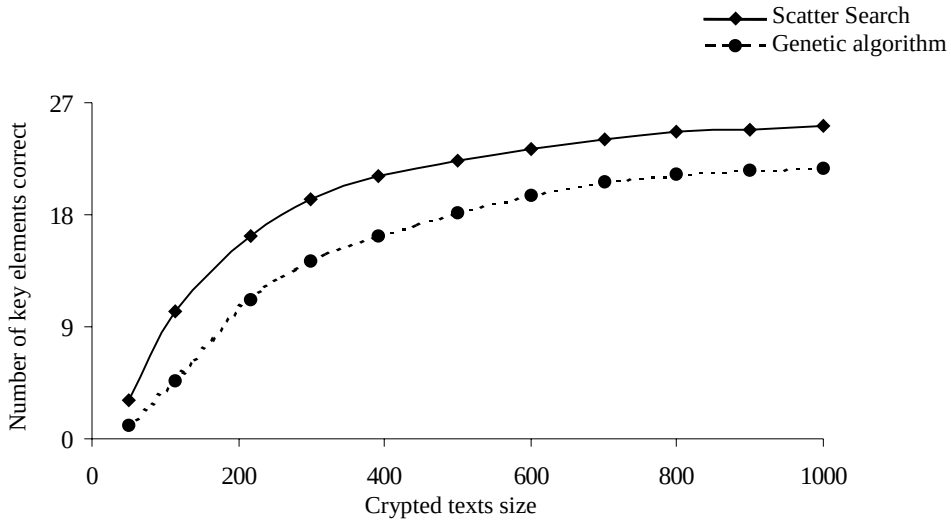

Figure 4: A comparison between scatter search and a genetic algorithm

Figure 3 shows clearly that scatter search returns solutions of better quality than the genetic algorithm (approximately 15%). The good performance of the scatter search procedure is -for the most part- the result of the improvement method which permits to explore better the neighbourhood of every considered solution. But in return, it needs more time to converge to the returned solution (approximately 75%).
To clearly show the importance of the improvement method; we remade the previous experimentation, but without using the improvement method for the research procedure, we get the following results:

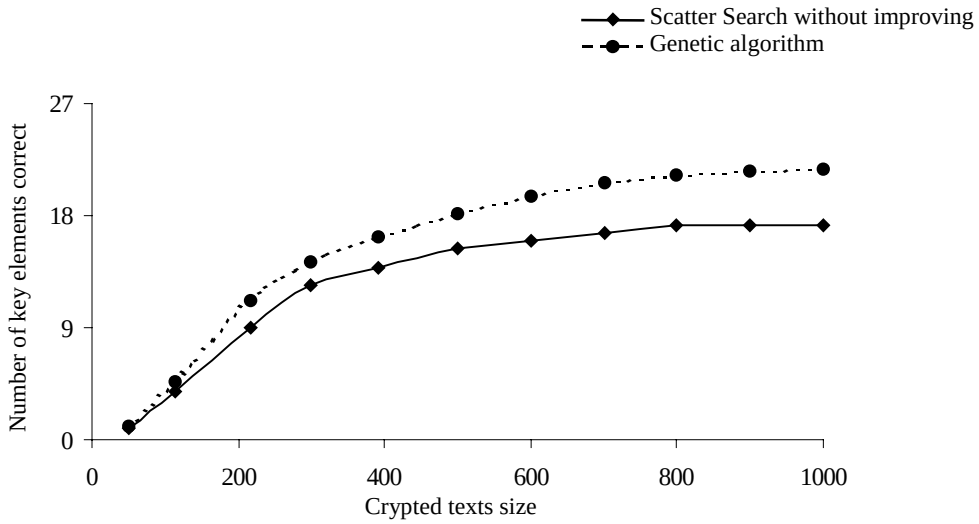

Figure 5: A comparison between scatter search (without improvement method) and a genetic algorithm

The curve above shows that solutions returned by the scatter search are of very lower quality than those returned previously.

The elimination of the improvement method from the scatter search decreased the rate of correct elements, because the scatter search mechanism is in this case equivalent to the genetic algorithm's one; but the fact that scatter search operates on a small population sees to it that the set of references converges prematurely and often toward middle quality solutions.

6. Conclusion

In this paper, scatter search is used to perform an automated attack against classical ciphers. First, we presented an implementation of the scatter search's procedures for the cryptanalysis of this class of ciphers. Since cryptanalysing simple ciphers is a permutation problem, then this implementation can be used as a framework for solving permutation problems with scatter search. Then, we performed tests, and the algorithm gave good results. We showed the contribution of each subset type and stressed the difference of contribution of certain subset types between this problem and other problems. We showed also that the robustness of the algorithm relies essentially on the improvement method.

It is clear that the heuristic methods have an important role to play in cryptanalysis. The next step is the cryptanalysis of more complex systems which use bit as an encoding unity. In this case, it's impossible to perform attack based on linguistic characteristics or frequencies analysis, but a 'known plaintext' attack or a 'chosen plaintext attack' are used. Another perspective is to use the meta-heuristic methods in conjunction with classical cryptanalysis techniques (like the differential analysis) to improve their efficacy.

Bibliography

[1] Campos V., Glover F., Laguna M., Martí R.: An Experimental Evaluation of a Scatter Search for the Linear Ordering Problem. *Journal of Global Optimizatio*n, vol. 21, 2001.

[2] Campos V., Glover F., Laguna M., Marti R.: An Experimental Evaluation of a Scatter Search for the Linear Ordering Problem. *Journal of Global Optimizatio*n, vol. 21, 1999.

[3] Clark J. A.: Metaheuristic Search as a Cryptological Tool. *PhD Thesis*, University of York, 2001.

[4] Clark A. J.: Optimisation Heuristics for Cryptology. *PhD Thesis*, Queensland University of Technology, 1998.

[5] Drias H., Azi N.: Scatter search for SAT and MAX-W-SAT problems. *Proceedings of the IEEE SSST*, Ohio, USA, 2001.

[6] Forsyth W. S., Safavi-Naini R.: Automated cryptanalysis of substitution ciphers. *Cryptologia*, n°17, vol. 4, 1991.

[7] Glover F.: Heuristics for Integer Programming Using Surrogate Constraints. *Decision Sciences*, vol 8, n° 1, 1977.

[8] Glover F., Kelly J.P., Laguna M.: Genetic Algorithms and Taboo Search: Hybrids for Optimization. *Computers and Operation Reseach*, vol 22, n° 1, 1995.

[9] Glover F.: A Template for Scatter Search and Path Relinking. *Notes in Computer Sciences*, Spinger-Verlag, 1998.

[10] Glover F., Laguna M., Marti R.: Fundamentals of scatter search and path relinking. *Control and Cybernetics*, vol. 39, n° 3, 2000.

[11] Russell M., Clark J. A., Stepney S.: Using Ants to Attack a Classical Cipher. *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO), 2003.

[12] Spillman R., Janssen M., Nelson B., Kepner M.: Use of a genetic algorithm in the cryptanalysis of simple substitution ciphers. *Cryptologia*, n°17, vol. 1, 1993.