ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/261422189

Heuristic Method to Find Magic Squares

Conference Paper · December 2012

DOI: 10.1109/ICCSE.2012.26

CITATIONS
3

READS
3,200

2 authors, including:

Enguerran Grandchamp
Université des Antilles

52 PUBLICATIONS **78** CITATIONS

SEE PROFILE

All content following this page was uploaded by Enguerran Grandchamp on 17 December 2018.

The user has requested enhancement of the downloaded file.

Heuristic Method to Find Magic Squares

Evelín Fonseca Cruz
Department of Mathematics and Computer Science
Havana University
Email: evelin@uh.cu

Enguerran Grandchamp
Department of Mathematics and Informatics
Antilles and Guyana University
Email: egrandch@univ-ag.fr

***Abstract*—Finding magic squares of order n is a search problem in a combinatorial space of $n^2!$ different squares. The construction of a magic square is simple for all n, because there are methods that create a deterministic solution for each n. These algorithms ensure the construction of a specific magic square for each n and other squares can be created from this by rotation and other operations. However, as n grows, the number of squares that cannot be obtained from these algorithms increases. The problem of finding different magic squares, not only those particulars provided by the deterministic solutions, is a challenge for any search method. This paper presents a new method to solve the magic square problem using a heuristic algorithm. The solution is separated in two phases. For the first phase, three heuristics are presented, which are used combined to construct a semi-magic square. In the second phase, a simple heuristic is used to find a magic square using the semi-magic square constructed before.**

I. Introduction

The magic square problem consists in distributing the numbers from 1 to n^2 in a squares of size $n \times n$ such that each row, column and diagonal adds up to the same value. During the late nineteenth century mathematicians applied the magic square in problems of probability and analysis, today they are studied in relation to factorial analysis, combinatorial mathematics, matrices, modular arithmetic and geometry [1]. With the development of computers, it has found many practical applications in artificial intelligence, graph theory, game theory, experimental design, industrial arts and electronic circuits [1], [2]. These squares have also been applied in recent decades in image processing [3], [4], [5], as well as in novel methods of encryption and transmission of digital signals [6], [7], [8], [9], [10], [11].

The construction of a magic square is simple for all n, because there are methods that create a deterministic solution for each order. These algorithms ensure the construction of specific magic squares for each n and other squares can be created from these by rotation and other operations. But the number of different magic squares increases with the order, and determining the number of different solutions for an order n is an open problem, hence there are many squares that remain unknown [1]. In addition, statistical analyses revealed that the difficulty of the search increases exponentially with the order [1].

Basically, find a magic square is a search problem in a combinatorial space of $n^2!$ different squares. This problem tests any heuristic solution, because the search domain grows exponentially with n, while the percentage of magic squares in the possible permutations decreases [1]. Motivated by the high complexity of its resolution, this problem has been used by search algorithms to demonstrate the convergence to global optima, appearing in various works which mixes local search algorithms with other methods [1], [12], [13], [14]. These works combine Artificial Intelligence tools with magic square specific heuristics to guide the search in the solution space. However, these tools hardly succeed in constructing magic squares of order greater than 10 without the use of these specific heuristics.

This paper discusses specific heuristics that can be used by local search methods to find different magic squares starting from a random square. The paper is organised as follows. The second section introduces a simple local search algorithm and a separation of the problem in two phases is presented. The section 3 summarises three heuristics for the first phase of the solution, then a second phase heuristic is drawn in the forth section followed by the conclusion in the section 5.

II. Local Search Algorithm

Magic Square Definition: A magic square of order n is a matrix $M = (a_{ij})_{n\times n}$, where $i, j = 1, 2, \ldots, n$, $a_{ij} \in \{1, 2, \ldots, n^2\}$ with $a_{ij} \neq a_{kl}$, for all $i \neq k$ or $j \neq l$ and it holds that:

$$\sum_{j=1}^{n} a_{ij} = \sum_{i=1}^{n} a_{ij} = \sum_{i=1}^{n} a_{ii} = \sum_{i=1}^{n} a_{i,n-i+1} = \frac{n(n^2+1)}{2} = S \quad (1)$$

Semi-magic Square Definition: A semi-magic square of order n is a matrix $M = (a_{ij})_{n\times n}$, where $i, j = 1, 2, \ldots, n$, $a_{ij} \in \{1, 2, \ldots, n^2\}$ with $a_{ij} \neq a_{kl}$, for all $i \neq k$ or $j \neq l$ and it holds that:

$$\sum_{j=1}^{n} a_{ij} = \sum_{i=1}^{n} a_{ij} = \frac{n(n^2+1)}{2} = S \quad (2)$$

The aim of this paper is to find different solutions to the magic square starting from an initial solution randomly created. The first objective is to find a semi-magic square, which is not taking into account the sums of the diagonals. The second objective is to find a magic square starting from a semi-magic square, iterating several squares that maintain the magic sum for all rows and columns.

In the first phase, the local search algorithm begins with a random solution and iterates to a neighbour by the permutation of a pair of cells. A random solution is a permutation of the numbers $\{1, 2, \ldots, n^2\}$ in the square of size $n \times n$. The method

Algorithm 1: Local Search

```
Input  : C, D, n
Output: The best square found
cBest = randomSolution();
count = 0;
while count < C do
    count ++;
    c = cBest;
    for 1 to D do
        c = getNeighbour(c);
        if c.betterThan(cBest) then
            cBest = c;
            count = 0;
            break;
```

proposed is divided into two phases, each with a different purpose. The **error function** $E_1(c)$ of a square c is:

$$E_1(c) = \sum_{i=1}^{n} |S - \sum_{j=1}^{n} a_{ij}| + \sum_{j=1}^{n} |S - \sum_{i=1}^{n} a_{ij}| \quad (3)$$

Then, a square c_1 is better than another c_2 if $E_1(c_1) < E_1(c_2)$.

In the second phase, two squares are considered neighbours if you can get from one to another through the exchange of a pair of rows or columns or by exchanging two pairs of cells, so as to maintain the property of a semi-magic square. To maintain the same property with the exchange of two pairs of cells, these must meet one of the following conditions:

1) The pairs of cells a_{ij} and a_{il} are exchanged with a_{kj} and a_{kl}, respectively if:

$$a_{ij} + a_{il} = a_{kj} + a_{kl} \quad (4)$$

2) The pairs of cells a_{ij} and a_{kj} are exchanged with a_{il} and a_{kl}, respectively if:

$$a_{ij} + a_{kj} = a_{il} + a_{kl} \quad (5)$$

The **error function** of a square c is:

$$E_2(c) = |S - \sum_{i=1}^{n} a_{ii}| + |S - \sum_{i=1}^{n} a_{i,n-i+1}| \quad (6)$$

Then, a square c_1 is better than another c_2 if $E_2(c_1) < E_2(c_2)$.

In both phases a local search algorithm is performed, iterating from neighbour to neighbour keeping c_{best} as the best solution founded so far. Each time a square c better than c_{best} is found, is set $c_{best} = c$ and a new cycle begins from this value. The search algorithm is summarised in algorithm 1, where C y D are predefined parameters. If the search arrives at a depth greater than D and there is not a better solution founded, the search is restarted from c_{best}. The stop condition of the algorithm is achieved when the square meets the objective of the phase or when it reaches a number of cycles C without find a better solution.

III. First Phase

Three heuristics are presented to solve the first phase: RRC, DRRC and CEB. The firsts two heuristics generalise the rectification process used by Xie and Kang [1]. The CEB heuristic was taken from [12], [13] and adapted for the first phase. Each heuristic return a pair of cell to exchange its values in order to improve the error of the square. If is not possible to improve the square, a random pair of cell will be exchanged.

A. RRC

The RRC, Rectification of Rows and Columns Heuristic, returns in each iteration a pair of cells to exchange its values. This heuristic is focused in diminish the error of a pair of rows or columns in each iteration, in this way the error of the semi-magic square also diminishes.

Taking X as a row or a column, the error of X is defined as:

$$E(X) = |S - \sum_{x \in X} x| \quad (7)$$

Let us define the row $f_i = \{a_{i1}, a_{i2}, \dots a_{in}\}$ and the column $c_i = \{a_{1i}, a_{2i}, \dots a_{ni}\}$. The heuristic RRC, exchanges the values of a pair of cells $\langle a_{ij}, a_{kj} \rangle$ in the same column j, allowing an improve in the error of the rows f_i and f_k without changes in the sum of others rows or columns of the square. Similarly, the heuristic can returns a couple of cells $\langle a_{ij}, a_{ik} \rangle$, ensuring that this movement only modify the sum of columns c_j and c_k. With the exchange of this pair of cells, the semi-magic error E_1 of the square decreases. The first phase of the algorithm is divided in two steps, the first step selects a pair of rows or a pair of columns to attempts to improve their error.

Defining each pair of rows or columns as: $\langle x_i, x_j \rangle \in \{\langle f_i, f_j \rangle, \langle c_i, c_j \rangle\}$ with $i, j \in \{1, 2, \dots, n\}$, the heuristic creates one sorted list L with all pairs of rows and columns $\langle x_i, x_j \rangle$ in ascending order, depending on the value of:

$$V(\langle x_i, x_j \rangle) = |(S - \sum_{x \in x_i} x) + (S - \sum_{x \in x_j} x)| \quad (8)$$

Pairs who have the same value of V are then sorted in descending order by the sum of their errors:

$$SE(\langle x_i, x_j \rangle) = E(x_i) + E(x_j) \quad (9)$$

After order L, the first pair $\langle x_i, x_j \rangle$ is selected. If $\langle x_i, x_j \rangle$ represents the rows $\langle f_i, f_j \rangle$, a pair of cells in a same column c_k will be chosen, to exchange the values $\langle a_{ik}, a_{jk} \rangle$. If the best pair is a pair of columns $\langle c_i, c_j \rangle$, a row f_k will be chose to exchange the pair $\langle a_{ki}, a_{kj} \rangle$. Let us then define $y_k \in \{f_k, c_k\}$ as the row or column to select depending on the value of $\langle x_i, x_j \rangle$:

$$\mathrm{y_k} = \begin{cases} f_k & \text{if } \langle x_i, x_j \rangle = \langle c_i, c_j \rangle \\ c_k & \text{if } \langle x_i, x_j \rangle = \langle f_i, f_j \rangle \end{cases} \quad (10)$$

The second step of the heuristic is to analyse the square error obtained by exchanging the cells for the selected $\langle x_i, x_j \rangle$ and y_k, for all y_k with $k = 1.2, \dots, n$:

- If $\langle x_i, x_j \rangle = \langle f_i, f_j \rangle$, $\langle a_{ik}, a_{jk} \rangle$ are exchanged.
- If $\langle x_i, x_j \rangle = \langle c_i, c_j \rangle$, $\langle a_{ki}, a_{kj} \rangle$ are exchanged.

The resulting square errors are analysed and the pair of cells that involves the best square is exchanged. If does not exist a couple of cells for $\langle x_i, x_j \rangle$ for any y_k that improves the square error, the next pair in the list L obtained in the first step will be selected and the search is repeated for all y_k.

1) Implementation and algorithmic cost: The algorithm to organise all the pairs of rows and columns has a computational time cost, in the first phase, of $\Theta(n^2 \log n)$ while the second phase has order $\Theta(n)$. Like the first step is to expensive, the ordering will be created just in the beginning of the search. Each time a pair of cell will be exchanged, the state of the list L will be updated in the pairs involved in the exchange. This means that if $\langle a_{ij}, a_{kl} \rangle$ are the pair of cells to exchange, only the pairs that contains the elements $\{f_i, c_j, f_k, c_l\}$ will be updated in the list L and the rest remains the same. Then, if we use a binary tree to create the list, the updating process will cost $\Theta(n \log n)$ and this will be the general cost of the heuristic.

B. DRRC

The heuristic DRRC, Double RRC, is used in combination with RRC and uses the same list L created in the first step. The difference between these two heuristics resides in the second step. DRRC exchanges two pairs of cells instead of one, ensuring that only the error of x_i and x_j changes and no other row or column in the square. Then, in the second step, for $\langle x_i, x_j \rangle$, we analyse all the pairs $\langle y_k, y_l \rangle$ with $k, l = 1, 2, \ldots, n$, which involves the exchange of the cells:

- a_{ik} with a_{jk} and a_{il} with a_{jl}, if $\langle x_i, x_j \rangle = \langle f_i, f_j \rangle$
- a_{ki} with a_{kj} and a_{li} with a_{lj}, if $\langle x_i, x_j \rangle = \langle c_i, c_j \rangle$

and once again the exchange that gets to the best square is selected. If a pair that improves the current square is not found, for $\langle x_i, x_j \rangle$, the next couple $\langle x_i, x_j \rangle$ is analysed. After choose the two pairs, only one is selected by the heuristic. This first exchange in DRRC not necessarily improve the error, the target will be completed later if the RRC uses the pair of cells that remains without been exchanged. Is for this reason that DRRC will be always used in combination with RRC.

The time cost of this second step is order $\Theta(n^2)$. The exchange in this kind of heuristics may involve more than two pairs, but the algorithm increases in complexity. For this reason, just these two options are considered.

C. Cell Error Based Heuristics: CEB

The second heuristic approach was taken from [12], [13] and adapted for the first phase. It uses the definition of a cell error a_{ij} as:

$$Efc(a_{ij}) = |(S - \sum_{k=1}^{n} a_{ik}) + (S - \sum_{k=1}^{n} a_{kj})| \tag{11}$$

The method selects two cells to swap their values:

- the first cell, a_{ij}, presents the worst error:

$$a_{ij} = \arg \max_{1 \le x,y \le n} Efc(a_{xy}) \tag{12}$$

Algorithm 2: Heuristics Combination

Input : Heuristics ordered set H, square c
Output: Action

```
action = null;
probability = nextRandom(0,1);
forall heuristic h ∈ H do
    if probability < Prob(h) then
        if h.generateAction(c) then
            return h.getAction();
return RandomAction();
```

- the second cell, a_{kl}, when exchanged with a_{ij} achieves the best error E_1 for the resulting square.

The algorithmic cost is $\Theta(n^2)$, determined by the second step. The first step could be done in time $\Theta(1)$ if the rows and columns remain ordered by its errors.

D. Heuristics Combination

The heuristics presented for the first phase can be used independently or combined. For both approaches, each heuristic h is used with a probability $Prob(h)$, specified in advance. These probabilities determine the moments when each heuristic is used, as is explained in the algorithm 2, where $h_1, h_2, ..., h_k$ are the heuristics to use in the solution, which are defined and used in that order and c is the actual square. The coexistence of different heuristics require that $Prob(h_k) \ge Prob(h_{k-1})$.

E. First Phase Results

The heuristics presented in the first phase can be used separated to find magic squares, but their combination offers better results. In order to obtain the best combination, different experiments were made, combining the heuristics, their probabilities and the depth in the search. For each case, 50 experiments were performed in squares of order $n = 20$. The probability are in $\{0.5, 0.6, 0.7, 0.8, 0.9, 1\}$ and the depth in $\{10, 20, 30, 40, 50\}$.

Some differences were observed for each combination regarding the percent of convergence to a global solution. The heuristic with the worst result in the convergence was CEB, which did not converges in any experiment. The best results were obtained from the combinations of the three heuristics, where the global convergence is always achieved. For the other combinations, results are presented in Fig. 1, where the first graphic represents the percentage of convergence only with RRC, the second shows the combination RRC/CEB and the third the RRC/DRRC.

From the convergence analysis, we conclude that the best approaches are the combinations of the three heuristics and the RRC/DRRC. The time of convergence is studied to decide the best combination. The Fig. 2 shows the average time of the experiments for RRC/CEB/DRRD, RRC/DRRD/CEB and RRC/DRRC, running on a PC Pentium Dual-Core CPU T4200 @ 2.00 GHz.

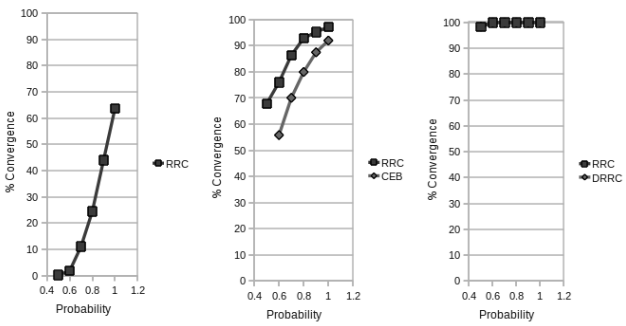

Fig. 1. Convergence Percent

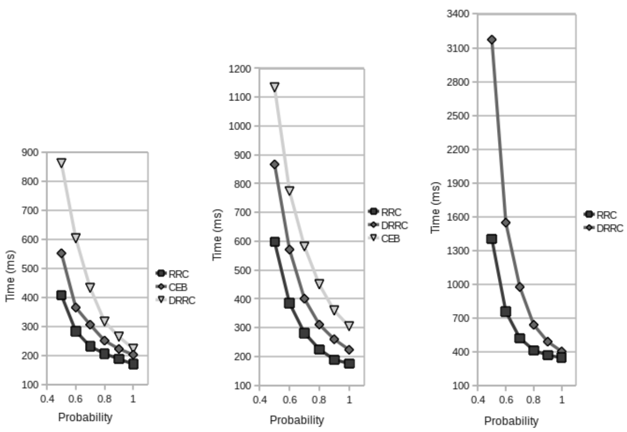

Fig. 2. Convergence Time

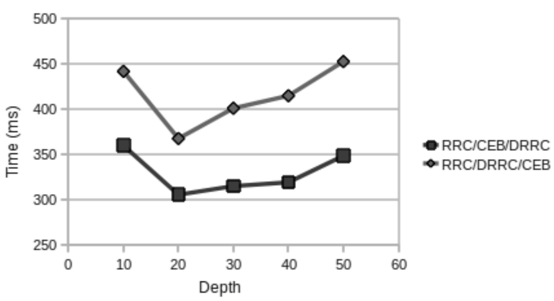

Fig. 3. Depth and time

The RRC is used as the first option in all combinations because it has the best time complexity, $\Theta(n \log n)$. On the other hand, the RRC shows a better behaviour than the CEB regarding the convergence percent. Besides, as the DRRC is a complement of RRC, it will be always used after.

The results shows that as higher the probability, better will be the convergence time, and the best time performances are achieved using the heuristics every time, with probability equals 1. The best combination is RRC/CEB/DRRC. The results for the depth parameter in the three heuristics combinations are shown in Fig. 3 where it can be seen that the best value is 20.

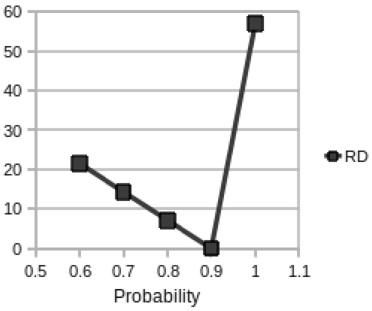

Fig. 4. Experiments that does not converge distributed by probabilities

IV. Second Phase Rectification Heuristic: RD

Following the same idea of generalising the rectification process, used by Xie and Kang [1], we propose a rectification version for the diagonals. The heuristic **RD**, Rectification of the Diagonals, exchanges values to improve or maintain the error of the diagonals without changing the sum of any row or column. RD consists of two steps, the first one attempts the permutation of a pair of rows or columns to improve the error of the square. If this is not possible, the second step is executed, which searches two pairs of cells that ensure a improvement in the square error while keeps the semi-magic square property. If neither of the two steps is possible, a pair of rows or columns is randomly exchanged.

The first step swaps all the pairs of rows or columns $\langle x_i, x_j \rangle \in \{\langle f_i, f_j \rangle, \langle c_i, c_j \rangle\}$, to calculate the resulting error of the square and then select the best result. If there is no pair that improves the error, we proceed to analyse the exchange of two pairs of cells, choosing at least two of them belonging to the diagonals:

- $\langle a_{ii}, a_{ji} \rangle$ and $\langle a_{ij}, a_{jj} \rangle$ if they satisfy:
$$a_{ii} + a_{ij} = a_{ji} + a_{jj} \tag{13}$$
- $\langle a_{i,n-i+1}, a_{j,n-i+1} \rangle$ and $\langle a_{i,n-j+1}, a_{j,n-j+1} \rangle$ if they satisfy:
$$a_{i,n-i+1} + a_{i,n-j+1} = a_{j,n-i+1} + a_{j,n-j+1} \tag{14}$$
- $\langle a_{ii}, a_{ij} \rangle$ and $\langle a_{ji}, a_{jj} \rangle$ if they satisfy:
$$a_{ii} + a_{ji} = a_{ij} + a_{jj} \tag{15}$$
- $\langle a_{n-i+1,i}, a_{n-i+1,j} \rangle$ and $\langle a_{n-j+1,i}, a_{n-j+1,j} \rangle$ if they satisfy:
$$a_{n-i+1,i} + a_{n-j+1,i} = a_{n-i+1,j} + a_{n-j+1,j} \tag{16}$$

A. *Second Phase Results*

The results of the experiments, using different values for the probability and depth of the search in the second phase, shows that the solution converges to a global solution in most of the experiments. Only the 0.9 percent didn't find a magic square. The Fig. 4 shows the distribution of these experiments between the values of probabilities.

The Fig. 5 shows the result of the experiments that achieve a global solution.

Table I shows the average time obtained with higher values of n using probability 0.9 and depth equals 20 for the second phase.

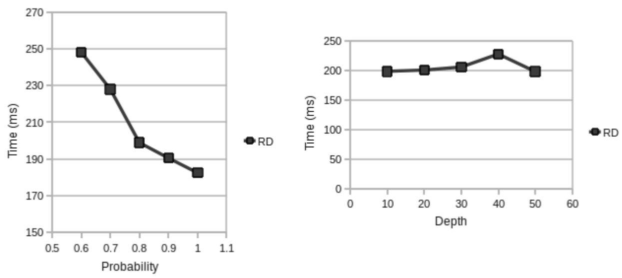

Fig. 5. (a) Convergence time versus probabilities values, (b) Convergence time versus depth

TABLE I
AVERAGE CONVERGENCE TIME IN SECONDS

n	**Phase 1**	**Phase 2**
40	0.632s	0.825s
60	1.150s	2.601s
80	2.232s	7.211s
100	4.799s	18.966s
200	33.538s	436.837s
300	148.436s	1873.630s
400	446.389s	3474.894s
500	997.416s	7847.922s

V. CONCLUSION

In this work various heuristics were presented to find different magic squares, starting with a random solution. With the division of the problem in two phases, the solution is simplified and the computational results shows that is possible to find magic square from a semi-magic square exchanging rows, columns or spatial pairs of cells. The solution presented for the first phase shows a better performance than the second, which could be improve using global search algorithms. Additionally, the heuristics presented for the first phase could be conveniently used with other local search algorithms like Tabu Search and Simulating Annealing.

REFERENCES

[1] T. Xie and L. Kang, "An evolutionary algorithm for magic squares," *Conference on Evolutionary Computation (CEC), China, IEEE*, pp. 906–913, 2003.

[2] M. Ramzan and M. K. Khan, "Distinguishing quantum channels via magic squares game," *Quantum Information Processing*, vol. 9, no. 6, pp. 667–679, Dec. 2010. [Online]. Available: http://dx.doi.org/10.1007/s11128-009-0155-4

[3] C.-C. Chang, D. Kieu, Z.-H. Wang, and M.-C. Li, "An image authentication scheme using magic square," *2nd IEEE International Conference on Computer Science and Information Technology, ICCSIT*, pp. 1–4, Aug. 2009.

[4] H. fen Huang, "Perceptual image watermarking algorithm based on magic squares scrambling in dwt," *Fifth International Joint Conference on INC, IMS and IDC. NCM '09*, pp. 1819–1822, Aug. 2009.

[5] J. Chen, T.-S. Chen, Y.-M. Hsu, and K. Wu, "A heuristic magic square algorithm for optimization of pixel pair modification," *Electrical Engineering and Control, LNEE 98*, pp. 765–772, 2011.

[6] M.-G. Wen, S.-C. Huang, and C.-C. Han, "An information hiding scheme using magic squares," *IEEE International Conference on Broadband, Wireless Computing, Communication and Applications (BWCCA)*, pp. 556–560, Nov. 2010.

[7] Y. Xiong and Z. Bu, "Magic squares transmission scheme for mimo ofdm systems under fast fading channel," *IEEE International Conference on Communications, Circuits and Systems. Proceedings*, vol. 1, pp. 175–179, May 2005.

[8] J. B. D. Fonseca, "From the magic square to the optimization of networks of agvs and from mip to an hybrid algorithm and from this one to the evolutionary computation," *Proceedings of the 5th WSEAS Int. Conf. on Artificial Intelligence*, pp. 117–122, 2006.

[9] P. Hill and A. Haisan, "Novel magic-square codes for hiding ds/ss signals from intercept receivers," *IEEE Third International Symposium on Spread Spectrum Techniques and Applications, ISSSTA*, vol. 2, pp. 484–488, Jul. 1994.

[10] Y. Zhang, P. Xu, and L. Xiang, "Research of image encryption algorithm based on chaotic magic square," *Advances in ECWAC*, vol. 2, pp. 103–109, 2012.

[11] G. Ganapathy and K. Mani, *Information Security Enhancement to Public-Key Cryptosystem Through Magic Squares*. Springer, 2010.

[12] P. Codognet and D. Diaz, "Yet another local search method for constraint solving," in *Proceedings of the International Symposium on Stochastic Algorithms: Foundations and Applications*, ser. SAGA '01. London, UK, UK: Springer-Verlag, 2001, pp. 73–90. [Online]. Available: http://dl.acm.org/citation.cfm?id=645844.668485

[13] P. Codognet, D. Diaz, and C. Truchet, "The adaptive search method for constraint solving and its application to musical csps," *First International Workshop on Heuristics, Beijing, China*, 2002.

[14] S. Kadioglu and M. Sellmann, "Dialectic search," *I.P. Gent*, pp. 486–500, 2009.

View publication stats