Open Mathematics **Open Access**

Research Article

Shonak Bansal*, Neena Gupta, and Arun Kumar Singh

Nature–inspired metaheuristic algorithms to find near–OGR sequences for WDM channel allocation and their performance comparison

DOI 10.1515/math-2017-0045
Received May 9, 2016; accepted February 21, 2017

Abstract: Nowadays, nature–inspired metaheuristic algorithms are most powerful optimizing algorithms for solving the NP–complete problems. This paper proposes three approaches to find near–optimal Golomb ruler sequences based on nature–inspired algorithms in a reasonable time. The optimal Golomb ruler (OGR) sequences found their application in channel–allocation method that allows suppression of the crosstalk due to four–wave mixing in optical wavelength division multiplexing systems. The simulation results conclude that the proposed nature–inspired metaheuristic optimization algorithms are superior to the existing conventional and nature–inspired algorithms to find near–OGRs in terms of ruler length, total optical channel bandwidth, computation time, and computational complexity. Based on the simulation results, the performance of proposed different nature–inspired metaheuristic algorithms are being compared by using statistical tests. The statistical test results conclude the superiority of the proposed nature–inspired optimization algorithms.

Keywords: Channel spacing; Conventional computing; Equally and unequally spaced channel allocation; Four–wave mixing; Metaheuristic; Nature–inspired algorithm; Near–optimal Golomb ruler; Optimization.

MSC: 90C26, 90C29, 91-08

1 Introduction

There exists a rich collection of nonlinear optical effects [1–9] in optical wavelength division multiplexing (WDM) systems, each of which manifests itself in a unique way. Out of these nonlinearities, the four–wave mixing (FWM) crosstalk signal is the major dominant noise effects in optical WDM systems employing equal channel spacing (ECS). FWM is a third–order nonlinear optical effect in which two or more wavelengths (or frequencies) combine and produce several mixing products. For uniformly spaced WDM channels, the generated FWM product terms fall onto other active channels in the band, causing inter–channel crosstalk. The performance can be substantially improved if FWM crosstalk generation at the channel frequencies is prevented. The efficiency of FWM depends on the channel spacing and fiber dispersion. If the frequency separation of any two channels of an optical WDM system is different from that of any other pair of channels, no FWM crosstalk signals will be generated at any of the channel frequencies [1–9].

***Corresponding Author: Shonak Bansal:** Department of Electronics and Communication Engineering, PEC University of Technology, Sector-12, Chandigarh, India, E-mail: shonakk@gmail.com
Neena Gupta, Arun Kumar Singh: Department of Electronics and Communication Engineering, PEC University of Technology, Sector-12, Chandigarh, India

To suppress the crosstalk due to FWM signals in optical WDM systems, several unequally spaced channel allocation (USCA) algorithms have been proposed in the literatures [1, 10–16]. However, the algorithms [1, 10–16] have the drawback of increased optical bandwidth requirement compared to the equally spaced channel allocation (ESCA). This paper proposes an unequally spaced optical bandwidth channel allocation algorithm by taking into consideration the concept of near–OGR sequences [7, 17–19] to suppress FWM crosstalk in optical WDM systems.

Studies have been shown that Golomb rulers represent a class of NP–complete [20] problems. For higher order marks, the exhaustive computer search [21, 22] for such problems is difficult. In the literatures [21–26], there are numerous algorithms to tackle the Golomb ruler problem. To date, no efficient algorithm is known for finding the shortest length ruler. The realization of nature–inspired metaheuristic optimization algorithms such as Memetic approach [26], Tabu search (TS) [26], Genetic algorithms (GAs) [27–30] and its hybridizations (HGA) [29], Biogeography based optimization (BBO) [30–32], Big bang–Big crunch (BB–BC) [33–36], Firefly algorithm (FA) [36–38], and hybrid evolutionary (HE) algorithms [38] in finding relatively good solutions to such NP–complete problems, provide a good starting point for algorithms of finding near–OGRs. Therefore, nature–inspired algorithms seem to be very effective solutions for such NP–complete problems. This paper proposes the application of three recent nature–inspired algorithms, namely Bat inspired algorithm (BA), Cuckoo search algorithm (CSA), Flower pollination algorithm (FPA) and their modified forms to find either optimal or near–optimal Golomb rulers in a reasonable time and their performance compared with the existing conventional and nature–inspired algorithms to find near–OGRs.

The structure of this paper is organized as follows: Section 2 presents the concept of Golomb rulers and its background. A brief account of nature–inspired based optimization algorithms is explained in Section 3. Section 4 and 5 provides the problem formulation and experimental results comparing with conventional and nature–inspired based optimization algorithms for finding unequal channel spacing (UCS), respectively. Conclusions and future work are outlined in Section 6.

2 Golomb Rulers and its Background

W. C. Babcock [7] firstly introduced the concept of *Golomb rulers*, and further was described by S. W. Golomb et. al. [17].

Golomb rulers are an ordered set of unevenly marks at non–negative integer locations such that no distinct pairs of numbers from the set have the same difference [43–46].

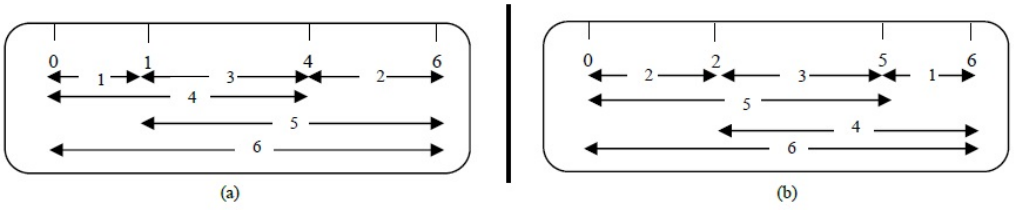

Figure 1: (a) A 4–marks optimal and perfect Golomb ruler with its associated distances and; (b) it's mirror image

Definition (Golomb ruler) 1: An n–marks Golomb ruler G is an ordered set of n distinct non–negative (positive) integer numbers

$$G = \{x_1, x_2, \ldots, x_{n-1}, x_n\} \qquad x_1 < x_2 < \ldots < x_{n-1} < x_n \tag{2.1}$$

such that all the positive differences

$$|x_i - x_j|, \qquad x_i, x_j \in G \qquad \forall\ i > j \text{ or } i \neq j \tag{2.2}$$

are distinct.

The non–negative integer numbers are referred to as *marks* or *order*. The number of marks on a ruler is referred to as the *ruler size*. The difference between the largest and smallest number is referred to as the *ruler length RL* [17, 30, 31], i.e.

$$RL = \max(G) - \min(G) = x_n - x_1 \tag{2.3}$$

where

$$\max(G) = \max\{x_1, x_2, \ldots, x_{n-1}, x_n\} = x_n \tag{2.4}$$

and

$$\min(G) = \min\{x_1, x_2, \ldots, x_{n-1}, x_n\} = x_1 \tag{2.5}$$

Definition (Golomb ruler) 2: A set of n distinct non–negative integer numbers $G = \{x_1, x_2, \ldots, x_{n-1}, x_n\}$ $x_1 < x_2 < \ldots < x_{n-1} < x_n$ is a Golomb ruler of n–marks if and only if [40]

$$\forall i, j, k, l \in \{1, 2, \ldots, n-1, n\}, x_i - x_j = x_k - x_l \Leftrightarrow i = k \wedge j = l. \tag{2.6}$$

Definition (Golomb ruler) 3: Let f: {1, 2,…,n–1, n}→ {0, 1,…,m–1, m} be injective with $f(1) = 0$, $G(n) = m$; f is a n–marks Golomb ruler if and only if [45, 46]

$$\forall i, j, k, l \in \{1, 2, \ldots, n-1, n\}, f(i) - f(j) = f(k) - f(l) \Leftrightarrow i = k \wedge j = l. \tag{2.7}$$

Generally, the first mark x_1 of Golomb ruler set G may be assumed on position 0. That is in such a canonical form, if

$$x_1 = 0 \tag{2.8}$$

then the n–marks Golomb ruler set G becomes

$$G = \{0, x_2, \ldots, x_{n-1}, x_n\} \tag{2.9}$$

The ruler length RL of such n–marks Golomb ruler set G is:

$$RL = x_n \tag{2.10}$$

Figure 1 illustrates an example of a Golomb ruler set $G = \{0, 1, 4, 6\}$ with mark $n = 4$ and ruler length $RL = 6$. The associated distance with each pair of marks is also shown in Figure 1.

2.1 Algebraic Properties of Golomb Rulers

The simple algebraic properties that find new Golomb ruler set G' are the translation property, the multiplication property, linearity property, and the mirror image or reflection property [39, 40, 47].

Translation Property: If the non–negative integer set $G = \{x_1, x_2, \ldots, x_{n-1}, x_n\}$ is an n–marks Golomb ruler, then the set

$$G' = \{a + x_1, a + x_2, \ldots, a + x_{n-1}, a + x_n\} \tag{2.11}$$

is also an n–marks Golomb ruler set.

Proof. If G is Golomb ruler and G' is not, there must exist i, j, k, l such that

$$(a + x_i) - (a + x_j) = (a + x_k) - (a + x_l) \tag{2.12}$$

$$\Rightarrow x_i - x_j = x_k - x_l \tag{2.13}$$

The equation 2.13 gives a contradiction since set G is a Golomb ruler. □

Multiplication Property: If the non–negative integer set $G = \{x_1, x_2, \ldots, x_{n-1}, x_n\}$ is an n–marks Golomb ruler, then the set

$$G' = \{ax_1, ax_2, \ldots, ax_{n-1}, ax_n\} \tag{2.14}$$

is also an n–marks Golomb ruler for every non–zero integer a (i.e. $a \neq 0$).

Proof. Likewise, in previous property, if set G is Golomb ruler and $G^{'}$ is not, there must exist i, j, k, l such that

$$ax_i - ax_j = ax_k - ax_l \tag{2.15}$$

$$\Rightarrow x_i - x_j = x_k - x_l \tag{2.16}$$

The equation 2.16 gives a contradiction fact, as original set G is a Golomb ruler. □

Linearity Property: If the non–negative integer set $G = \{x_1, x_2, \ldots, x_{n-1}, x_n\}$ is an n–marks Golomb ruler, then the set

$$G^{'} = a.G + b = \{ax_1 + b, ax_2 + b, \ldots, ax_{n-1} + b, ax_n + b\} \tag{2.17}$$

is also an n–marks Golomb ruler for all non–negative integers a, b, with non–zero value of integer a (i.e. $a \neq 0$).

Proof. As in the previous properties, if G is Golomb ruler set and $G^{'}$ is not, then there must exist i, j, k, l such that

$$(ax_i + b) - (ax_j + b) = (ax_k + b) - (ax_l + b) \tag{2.18}$$

$$\Rightarrow x_i - x_j = x_k - x_l \tag{2.19}$$

The equation 2.19 gives the contradiction fact, as G is a Golomb ruler set. □

Mirror Image or Reflection Property: Let a mirror is put on the one end of a Golomb ruler set $G = \{0, 1, 4, 6\}$ as shown in Figure 1. Then the integer numbers called *marks* will be estimated to some other points in the other half plane with the same mutual distances. A new Golomb ruler set $G^{'} = \{0, 2, 5, 6\}$ will be formed as shown in Figure 1. Thus the new n–marks Golomb ruler set by using both translation and multiplication properties can be written as [41]:

$$G^{'} = x_n - 1G = \{x_n - x_1, x_n - x_2, \ldots, x_n - x_{n-1}, x_n - x_n\} \tag{2.20}$$

This new n–marks Golomb ruler set is the *mirror image* of the original one. The new n–marks Golomb ruler set has a first mark at position 0 (i.e. $x_1 = 0$) and has the same ruler length as that of the original (i.e. $RL = x_n$). In the mirror image of n–marks Golomb ruler set $G^{'}$, the n-th mark of the original Golomb ruler set G will be at the leftmost position of the mirror image Golomb ruler. The new position of each mark is given by [25, 44]:

$$new\ position = n - old\ position \tag{2.21}$$

Each Golomb ruler set has a mirror image for $n > 2$ and all the obtained mirror images are different from the original Golomb ruler set.

Proof. As stated in the above mentioned properties, if set G is Golomb ruler and $G^{'}$ is not, then there must be i, j, k, l such that

$$(x_n - x_i) - (x_n - x_j) = (x_n - x_k) - (x_n - x_l) \tag{2.22}$$

$$\Rightarrow x_i - x_j = x_k - x_l \tag{2.23}$$

The equation 2.23 contradicting the fact that the set G is a Golomb ruler. □

2.2 Perfect Golomb Ruler

A perfect Golomb ruler measures all the positive integer distances from 1 to ruler length RL [20, 27].
Definition (Perfect Golomb ruler): An n–marks Golomb ruler set $G = \{x_1, x_2, \ldots, x_{n-1}, x_n\}$ is called perfect if for every integer, say d, $1 \le d \le (x_n - x_1)$, there is at least one solution to $d = x_i - x_j$, $x_i, x_j \in G \quad \forall \; i > j$.

Figure 1 shows the example of a 4–marks perfect Golomb ruler set which measures all the integer distances from 1 to 6. Together with the translations and reflections around the midpoint $\left(\frac{x_1+x_n}{2}\right)$, the only perfect Golomb rulers are {0}, {0,1}, {0,1,3}, and {0,1,4,6}. There exist no perfect Golomb rulers for $n > 4$ [40]. An n–marks perfect Golomb ruler must have ruler length RL [25, 27]

$$RL = \left(\frac{n^2 - n}{2}\right) = \sum_{i=1}^{n-1} i \tag{2.24}$$

units long because it must include the lengths RL, $(RL - 1)$, $(RL - 2)$,…,$(RL - (RL - 1))$. If the ruler length $RL > \left(\frac{n^2-n}{2}\right)$, then there is a distance between 0 and RL that is not measured, as the number of marks on the ruler cannot be changed.

2.3 Optimal Golomb Rulers (OGRs)

Since perfect Golomb rulers for $n > 4$ cannot exist, the higher mark rulers are stated in terms of whether or not they are optimal. So, an optimal Golomb ruler (OGR) is the shortest length ruler for a given number of marks [48, 49].

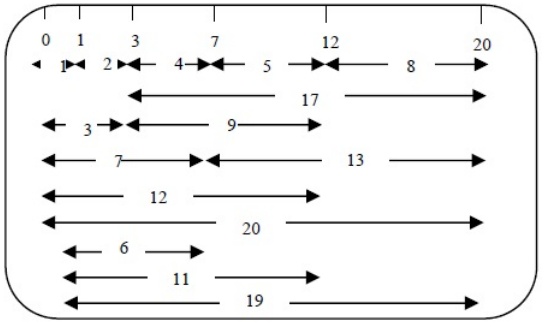

Figure 2: A 6–marks non–OGR with its associated distances

Definition (Optimal Golomb ruler): An n–marks Golomb ruler set $G = \{0, x_2, \ldots, x_{n-1}, x_n\}$ is referred to as optimal if it is of shortest possible length.

There can be multiple different OGRs for a specific number of marks. However, the unique optimal 4–marks Golomb ruler is shown in Figure 1. In a case where a Golomb ruler is not optimal, but its length is near to optimal, it will be considered as a near or non–optimal Golomb ruler. For example, the set $G = \{0,1,3,7\}$ is a 4–marks near–OGR since its differences are {1 = 1 − 0; 2 = 3 − 1; 3 = 3 − 0; 4 = 7 − 3; 6 = 7 − 1; 7 = 7 − 0}, all of which are distinct. It is clear from the differences that the integer number 5 is absent so it cannot be designated as a perfect Golomb ruler. An n–marks Golomb ruler is said to be an optimal Golomb ruler if and only if, [30]

1. there is no other n–marks Golomb ruler with the smaller ruler length, and
2. the ruler is in canonical form as the *smaller* of the equivalent rulers $\{0, x_2, \ldots, x_{n-1}, x_n\}$ and $\{0, \ldots, x_n - x_2, x_n\}$. This *smaller* means that the difference distance between the first two marks in the ruler set is less than the corresponding distance in the equivalent ruler.

For example the set $G = \{0,1,3,7,12,20\}$, shown in Figure 2 is a non–optimal 6–marks Golomb ruler having ruler length 20. From the differences it is clear that the numbers 10, 14, 15, 16, 18 are missing, so it is not a perfect Golomb ruler sequence. The distance associated between each pair of marks is also shown in Figure 2.

The OGRs play an important role in a variety of real–world applications including radio frequency allocation, sensor placement in X–ray crystallography, computer communication network, pulse phase modulation, circuit layout, geographical mapping, self–orthogonal codes, VLSI architecture, coding theory, linear arrays, fitness landscape analysis, radio astronomy, antenna design for radar missions, sonar applications and NASA missions in astrophysics, planetary and earth sciences [7, 17, 25, 27, 41, 42, 50–56].

Although the definition of Golomb rulers does not place any limitation on the ruler length, researchers are usually interested in rulers with smallest length, i.e. to find the n–marks OGR, the function $G(n)$ is given by:

$$G(n) = \min\{RL(G) : G \text{ is Golomb ruler set, } |G| = n\} \tag{2.25}$$

So far the exact values of OGRs $G(n)$ for up to $n = 27$ are known and there are some prospects for OGRs up to 158–marks [57, 58]. Firstly, the idea of Golomb rulers up to 10–marks were studied by W. C. Babcock [7], while analyzing the position of radio channels in the frequency spectrum. According to the literatures [40–42], all of rulers' up to 8–marks introduced by Babcock are optimal; the 9 and 10–marks are near–to–optimal. A. Dimitromanolakis [41] computationally estimated and proved a detailed discussion of OGRs for every integer n that

$$G(n) \leq n^2, \qquad \forall n \ \leq 65000 \tag{2.26}$$

As there are $\left(\frac{n^2-n}{2}\right)$ distinct non–negative differences between any two marks, so the ruler length is at least the same as given by equation 2.24. The computer searches give the largest known value of $G(n)$. The 27–marks OGR has an optimal ruler length of 553 i.e. from the equation 2.25, $G(27) = 553$ which satisfies the equation 2.26 as well. The search for OGRs and to prove its optimality took several months or even years. The optimal Golomb rulers for 5 to 7–marks were found by J. P. Robinson et. al. [51] and for 8 to 11–marks by W. Mixon. The OGRs with 12 and 13–marks were found by J. P. Robinson [21], those from 14 to 17–marks were found by J. B. Shearer [22] by exhaustive computer search. According to [41, 57, 58], the rulers for 17 and 18–marks were proved to be optimal by O. Silbert, while 19–marks OGR was found by W. T. Rankin [25] computer search approach. The search for OGRs for 20 to 27–marks was completed by distributed.net [43] OGR project. Distributed.net is now actively searching the OGRs for $n > 27$.

2.3.1 Related Literature on Constructing OGRs

The n–marks optimal Golomb ruler problem has been solved by using numerous different methods. In this subsection, a quick review of some known relevant approaches used to find either optimal or near–optimal Golomb rulers are presented.

2.3.1.1 Algebraic approaches

In the literature several algebraic approaches and their proof are available for constructing Golomb ruler sequences. The best known construction algebraic approaches to produce Golomb rulers are Extended quadratic congruence (EQC) and Search algorithm (SA), Erdös–Turan, Rusza–Lindström, Bose–Chowla, Singer, and Symmetric Golomb Costas arrays constructions [1, 15, 41, 46]. The construction approaches EQC and SA proposed in [1, 15] works always, but, unfortunately, are far from optimal and are limited to prime number of marks. A drawback of the construction approaches presented in [41, 46] is that they can find either optimal or near–OGRs only to a prime or the power of a prime number of marks.

2.3.1.2 Exact approaches

There are some classical approaches used to find and verify OGRs. One of these approaches, Scientific American algorithm was presented in [59, 60]. This approach had two parts: the ruler generation and the Golomb verification. The input to generation part was the number of marks n and an upper bound on the ruler length and constructed a Golomb ruler. The second part of the approach verifies whether or not the generated ruler sequence meets all criteria for a Golomb ruler. Other approaches to find OGRs are Token Passing algorithm and the Shift algorithm [25, 42]. To improve the execution time, approaches in [25, 42] considered a set of search space reduction methods. Generally, both non–systematic and systematic (exact) approaches had been applied in order to find the OGRs. With non–systematic approaches, optimal or near–optimal Golomb rulers can be computed up to 158–marks [57, 58]. The most efficient exact approach to find OGRs was presented by J. B. Shearer [22] to enumerate OGRs up to 16–marks. This approach was based on the combination of branch–and–bound and backtracking approaches, making an upper–bound set to the length of the best known solution. This approach positively influenced the performance by avoiding the divergence of the results. This approach is used in several parallel schemes such as the OGR project by Distributed.net. The drawback of this approach is that it took several months or even years of calculation to achieve results with high computational power [25, 42, 43].

Thus, finding Golomb ruler sequences and to prove its optimality with a large number of marks is, computationally speaking, very costly. For example, the search for 19–marks OGR took approximately 36,200 CPU hours on a Sun Sparc workstation using a very particular approach [42]. Moreover, the OGRs for 20 to 27–marks were obtained by using exhaustive parallel search computation on distributed.net, taking several months – even years of calculations for 24 or 27–marks [43]. In [27], it was proved that Golomb rulers represent a class of NP–complete problem. For the larger marks values the exhaustive search, without metaheuristics, of such NP–complete OGR problems is impossible. The time required to find an optimal value of n–marks Golomb ruler, i.e. $G(n)$ increases exponentially by a factor of more than 10 when the value of mark n is increased by 1 [41].

Since the exact approaches for finding Golomb rulers is impractical in terms of computational resources, different nature–inspired based metaheuristic optimization algorithms [26–39] have been proposed to find optimal and near–optimal Golomb rulers sequences at a reasonable time. Therefore, the nature–inspired based metaheuristic optimization algorithms appear to be the best solutions for such NP–complete OGR problem.

This paper presents the application of three new nature–inspired metaheuristic optimization algorithms to find near–OGR sequences as a WDM channel allocation algorithm. On applying the OGRs as channel allocation, it was possible to achieve the smallest distinct number to be used for the WDM channel allocation problem. As the difference between any two numbers is distinct, the new FWM frequency signals generated would not fall into the one already assigned to the carrier channels.

3 Nature–Inspired Metaheuristic Algorithms

Due to the highly nonlinearity and complexity of the problem of interest, design optimization in engineering fields tends to be very challenging. As conventional computing algorithms are local search algorithm, they are not the best tools for highly nonlinear global optimization, and thus often miss the global optimality. In addition, design solutions have to be robust, low cost, subject to uncertainty in parameters and tolerance for imprecision of available components and materials. Nature–inspired algorithms are now most widely used optimization algorithms. The guiding principle is to devise algorithms of computation that lead to an acceptable solution at low cost by seeking for an approximate solution to a precisely/imprecisely formulated problem [61–68].

This section is devoted to the brief overview of nature–inspired based metaheuristic optimization algorithms based on the theories of the echolocation characteristics of microbats called BA, brood parasitism of the cuckoo species called CSA and flow pollination process of flowering plants called FPA.

The power of nature–inspired optimization algorithms lies in how faster the algorithms explore the new possible solutions and how efficiently they exploit the solutions to make them better. Although all optimization algorithms in their simplified form work well in the exploitation (the fine search around a local optimal), there are some problems in the global exploration of the search space. If all of the solutions in the initial phase of the optimization algorithm are collected in a small part of search space, the algorithms may not find the optimal result and with a high probability, it may be trapped in that sub–domain. One can consider a large number for solutions to avoid this shortcoming, but it causes an increase in the function calculations as well as the computational costs and time. So for the optimization algorithms, there is a need by which exploration and exploitation can be enhanced and the algorithms can work more efficiently. By keeping this in mind, two features, fitness (cost) based mutation strategy and random walk i.e. Lévy–flight distribution is being introduced in the proposed nature–inspired metaheuristic algorithms, which is the main technical contribution of this paper. In modified algorithms the mutation rate probability is determined based on the fitness value. The mutation rate probability MR_i^t of each solution x_i at running iteration index t, mathematically is:

$$MR_i^t = \frac{f_i^t}{Max(f^t)} \tag{3.1}$$

where f_i^t is the fitness value of each solution x_i at the iteration index t, and $Max(f^t)$ is the maximum fitness value of the population at iteration t. For all proposed algorithms, the mutation equation [69, 70] use throughout this paper is:

$$x_i^t = x_i^{t-1} + p_m(x_{best}^{t-1} - x_i^{t-1}) + p_m(x_{r_1}^{t-1} - x_{r_2}^{t-1}) \tag{3.2}$$

where x_i^t is the population at running iteration index t, $x_{best}^{t-1} = x_\star^{t-1}$ is the current global best solution at iteration one less than running iteration index t, p_m is mutation operator, r_1 and r_2 are uniformly distributed random integer numbers between 1 to size of the given problem. The numbers r_1 and r_2 are different from running index. Typical values of p_m are the same as in GAs, i.e. 0.001 to 0.05. The mutation strategy increases the chances for a good solution, but a high mutation rate (p_m = 0.5 and 1.0) results in too much exploration and is disadvantageous to the improvement of candidate solutions. As p_m decreases from 1.0 to 0.01, optimization ability increases greatly, but as p_m continues to decrease to 0.001, optimization ability decreases rapidly. A small value of p_m is not able to sufficiently increase the solution diversity [30].

The Lévy flight distribution [71] used for all proposed algorithms in this paper mathematically is given by:

$$L(\lambda) \sim \frac{\lambda\Gamma(\lambda)\sin(\pi\lambda/2)}{\pi}\frac{1}{s^{1+\lambda}}, \qquad (s >> s_0 > 0) \tag{3.3}$$

Here, $\Gamma(\lambda)$ is the standard gamma distribution, valid for large steps, i.e. for $s > 0$. Throughout the paper, λ = 3/2 is used. In theory, it is required that $|s_0| \gg 0$, but in practice s_0 can be as small as 0.1 [71].

By introducing these two features in the simplified forms of proposed nature–inspired algorithms, the basic concept of the search space is modified, i.e. the proposed algorithms can explore new search space by the mutation and random walk. A fundamental benefit of using mutation and Lévy flight strategies with nature–inspired algorithms in this paper is their ability to improve its solutions over time, which does not seem in the existing algorithms [26–34, 37, 39] to find near–OGRs.

3.1 Bat Algorithm and its Modified forms

X. S. Yang [61–65, 72, 73], inspired by the echolocation characteristics of microbats, introduced an optimization algorithm called Bat algorithm (BA). In describing this new algorithm, the author in [73] uses the following three idealized rules:

1. To sense the distance, all bats use echolocation and they also know the surroundings in some magical way;
2. Bats fly randomly with a velocity v_i at position x_i, with a fixed frequency range $[f_{min}, f_{max}]$, fixed wavelength range $[\lambda_{min}, \lambda_{max}]$, varying its pulse emission rate $r \in [0, 1]$, and loudness A^0 to hunt for prey, depending on the proximity of their target;

3. Although the loudness can vary in different ways, it is assumed that the loudness varies from a minimum constant (positive) A_{min} to a large A^0.

In BA, each bat has its position x_i, velocity v_i, frequency f_i, loudness A_i, and the emission pulse rate r_i in a d−dimensional search space. Among all the bats, there is a current global best solution $x_\star$ which is located after comparing all the solutions among all the bats. The new velocities v_i^t and solutions x_i^t at step t are given by the following equations [61, 62, 73–76]:

$$f_i = f_{\min} + (f_{\max} - f_{\min})\beta \tag{3.4}$$

$$v_i^t = v_i^{t-1} + (x_i^{t-1} - x_\star)f_i \tag{3.5}$$

$$x_i^t = x_i^{t-1} + v_i^{t-1} \tag{3.6}$$

where $\beta \in [0, 1]$ is a random vector drawn from a uniform distribution. A random walk is used for local search that modifies the current best solution according to:

$$x_{new} = x_{best} + \varepsilon A^t \tag{3.7}$$

where $x_{best} = x_\star$, $\varepsilon \in [-1, 1]$ is a scaling factor and A^t is loudness. Further the loudness A and pulse rate r are updated according to the equations 3.8 and 3.9 respectively as the iterations proceed:

$$A_i^t = \alpha A_i^{t-1} \tag{3.8}$$

$$r_i^t = r_i^0[1 - e^{-\gamma t}] \tag{3.9}$$

where α and γ are constants and for simplicity, $\alpha = \gamma$ is chosen. For most of the simulation $\alpha = \gamma = 0.9$ is used [73–76].

By combining the characteristics of mutation strategy (equations 3.1 and 3.2), Lévy flights distribution (equation 3.3), with the simple BA, three new algorithms, namely, *Bat algorithm with mutation* (BAM), *Lévy flight Bat algorithm* (LBA), and *Lévy flight Bat algorithm with mutation* (LBAM) can be formulated. For LBA, the modification performed in equation 3.7 is given by:

$$x_{new} = x_{best} + \varepsilon A^t \oplus L(\lambda) \tag{3.10}$$

Based on these idealizations, the basic steps of BA can be described as a general pseudo−code shown in Figure 3. In Figure 3, if the concept of Lévy flights in lines 11, 12 and mutation (lines 17 to 22) are omitted, then Figure 3 corresponds to the general pseudo−code for simple BA. If only the concept of mutation is not used in Figure 3, then it corresponds to the pseudo−code for LBA, otherwise Figure 3 shows the general pseudo−code for LBAM algorithm.

3.2 Cuckoo Search Algorithm and its Modified form

X. S. Yang et al. [77–80], inspired by brood parasitism of some cuckoo species, developed a novel nature−inspired metaheuristic optimization algorithm called a Cuckoo search algorithm (CSA). In addition, CSA algorithm is enhanced by the Lévy flight trajectory of some birds, rather than by simple random walks. For describing this new algorithm, X. S. Yang et al. use the following three idealized rules:

1. Each cuckoo lays one egg at a time, and dumps it in a randomly chosen nest;
2. The best nest with high quality of eggs (solution) are carried over to the next iterations;

```
1. Modified Bat algorithm (MBA)
2. Begin
3.    /* MBA parameter initialization */
4.       Define objective function f(x);  x = (x1, ..., xd)^T;
5.      Generate initial bat population xi and initial velocities vi (i = 1,
6.       2,...n);
7.       Define pulse frequency fi at xi;
8.       Initialize pulse rates ri and the loudness Ai;
9.    /* End of MBA parameter initialization */
10.  While not t                          /* t is a termination criterion */
11.     Generate new solutions by adjusting frequency, and updating
12.     velocities and locations/solutions by Lévy flights;
13.     If (rand > ri)
14.        Select a solution among the best solutions;
15.        Generate a local solution around the selected best solution;
16.     Else
17.        /* Mutation */
18.          Compute mutation rate probability MRi via equation 3.1;
19.            If (MRi < rand)
20.               Perform mutation via equation 3.2;
21.            End if
22.        /* End of Mutation */
23.     End if
24.     Generate a new solution by flying randomly;
25.     If (rand < ri and f (xi) < f (x*))
26.        Accept the new solutions;
27.          Increase ri and reduce Ai;
28.     End if
29.     Rank the bats and find the current best;
30.   End while
31.   Post-process results and visualization;
32.  End
```

Figure 3: The general pseudo–code for MBA

```
1. Cuckoo search algorithm with mutation (CSAM)
2.    Begin
3.      /* CSAM parameter initialization */
4.          Define objective function f(x);   x = (x1, ..., xd)^T;
5.          Generate initial population of n host nests xi (i = 1, 2, ..., n);
6.      /* End of CSAM parameter initialization */
7.      While not t                          /* t is a termination criterion */
8.          Get a cuckoo (say i) randomly by Lévy flights;
9.          /* Mutation */
10.           Compute mutation rate probability MRi via equation 3.1;
11.             If (MRi < rand)
12.                Perform mutation via equation 3.2;
13.             End if
14.         /* End of mutation */
15.         Evaluate its quality/fitness Fi;
16.         Choose a nest among n (say j) randomly;
17.         If (Fi > Fj),
18.            Replace j by the new solution;
19.         End if
20.         A fraction (pa) of worse nests are abandoned and new ones
21.         are built;
22.         Keep the best solutions (or nests with quality solutions);
23.         Rank the solutions and find the current best;
24.     End while
25.     Post-process results and visualization;
26.  End
```

Figure 4: The general pseudo–code for CSAM.

3. The number of available host nests is fixed, and a host can discover an alien egg with probability $p_a \in [0,1]$. In this case, the host bird can either throw the egg away or simply abandon the nest so as to build a completely new nest in a new location.

For simplicity, the last assumption can be approximated by a fraction p_a of the n host nests being replaced by new nests (with new random solutions). For a maximization problem, the quality, i.e. fitness of a solution

can simply be proportional to the value of the objective function. When new solutions x^t are generating for, say, a cuckoo i, a Lévy flight is performed as [77]:

$$x_i^t = x_i^{t-1} + \alpha \oplus L(\lambda) \tag{3.11}$$

where $\alpha > 0$ is the step size, which should be related to the scale of the specified problem.

As the authors in [77], already introduced the Lévy flights distribution concept to enhance the performance, so only mutation strategy is applied to the simple CSA to explore the search space. The new modified algorithm so formulated is named as a *Cuckoo search algorithm with mutation* (CSAM). The basic steps of CSAM can be summarized as the pseudo–code shown in Figure 4. If the concept of mutation (lines 9 to 14) is withdrawn from Figure 4, then it corresponds to the pseudo–code for simple CSA.

3.3 Flower Pollination Algorithm and its Modified form

X. S. Yang et. al [71, 81, 82], inspired by the flow pollination process of flowering plants, introduced a novel nature–inspired optimization algorithm called Flower pollination algorithm (FPA). For describing this novel metaheuristic algorithm, the authors in [71] use the following four idealized rules:

1. For global pollination process, biotic cross–pollination is used and pollen–carrying pollinators obey Lévy flight movements;
2. For local pollination, abiotic and self–pollination are used;
3. Pollinators such as insects can develop flower constancy, which is equivalent to a reproduction probability that is proportional to the similarity of two flowers involved;
4. The interaction of local pollination and global pollination can be controlled by a switch probability $p \in [0,1]$, with a slight bias towards local polination.

In FPA, the global pollination and local pollination are two main steps. In the global pollination step, flower pollens are carried by pollinators such as insects, and pollens can travel over a long distance because insects can often fly and travel over a much longer range. The first rule and flower constancy (i.e. third rule) can be written mathematically as [71]:

$$x_i^t = x_i^{t-1} + \gamma L(\lambda)(x_* - x_i^{t-1}) \tag{3.12}$$

where x_i^t is the pollen i or solution vector x_i at iteration t, x_* is the current best solution (i.e. most fittest) found among all solutions at the current iteration and γ is a scaling factor to control the step size. The Lévy flight based step size $L(\lambda)$ corresponds to the strength of the pollination. Since insects may travel over a long distance with various distance steps, a Lévy flight can be used to mimic this characteristic efficiently. That is, $L > 0$ is drawn from a Lévy flight distribution.

For local pollination, the second rule and flower constancy can be written mathematically by [71]:

$$x_i^t = x_i^{t-1} + \in (x_j^{t-1} - x_k^{t-1}) \tag{3.13}$$

where x_j^{t-1} and x_k^{t-1} are pollens from different flowers of the same plant species. This essentially mimics the flower constancy in a limited neighborhood. Mathematically, if x_j^t and x_k^t are selected from the same population, this become a local random walk if $\in$ is drawn from a uniform distribution in [0,1]. Pollination may also occur in a flower from the neighboring flower than by the far away flowers. For this, a switch probability (i.e. fourth rule) or proximity probability p can be used to switch between global pollination and local pollination.

Like CSA, the author in [71] already introduced the concept of Lévy flight distributions in FPA, so only mutation based on the fitness value (equations 3.1 and 3.2) is added to simple FPA. The new algorithm so formed is named in this paper as a *Flower pollination algorithm with mutation* (FPAM) which is summarized as pseudo–code shown in Figure 5. The only difference in the pseudo–code for FPA and FPAM is only the addition of mutation (lines 19 to 24) in Figure 5. If lines 19 to 24 are not used in Figure 5 then it corresponds to the pseudo–code for simple FPA.

```
1. Flower pollination algorithm with mutation (FPAM)
2.  Begin
3.   /* FPAM parameter initialization */
4.     Define objective function f(x);   x = (x1, ..., xd)^T;
5.     Initialize a population of n flowers/pollen gametes with random
6.     solutions;
7.     Find the best solution g* in the initial population;
8.     Define a switch probability p ∈ [0,1] ;
9.   /* End of FPAM parameter initialization */
10.  While not t                          /* t is a termination criterion */
11.    For i = 1 : n                                   /*all n flowers */
12.       If (rand < p)
13.          Draw a (d–dimensional) step vector L via Lévy flights;
14.          Perform global pollination via equation 3.12;
15.       Else
16.            Draw  ∈ from a uniform distribution in [0,1];
17.          Perform local pollination via equation 3.13;
18.       End if
19.       /* Mutation */
20.        Compute mutation rate probability MRi via equation 3.1;
21.         If (MRi < rand)
22.              Perform mutation via equation 3.2;
23.         End if
24.       /* End of mutation */
25.       Evaluate new solutions;
26.       If new solutions are better, update them in the population;
27.    End for
28.    Rank the solutions and find the current best solution x* ;
29.  End while
30.  Post-process results and visualization;
31. End
```

Figure 5: The general pseudo–code for FPAM

4 Finding Near–OGRs: Problem Formulation

Both simplicity and efficiency attract researchers towards natural phenomenon to solve NP–complete and complex optimization problems. The first problem investigated in this research is to find Golomb ruler sequences for unequal channel allocation. The second problem is to obtain either optimal or near–optimal Golomb rulers through nature–inspired metaheuristic algorithms by optimizing the ruler length so as to conserve the total occupied optical channel bandwidth.

If each individual element in an obtained set (i.e. non–negative integer location) is a Golomb ruler, the sum of all elements of an individual set form the total occupied optical channel bandwidth. Thus, if the spacing between any pair of channels in a Golomb ruler set is denoted as CS, an individual element is as IE, and the total number of channels/marks is n, then the ruler length RL and the total optical channel bandwidth TBW are given mathematically by the equations 4.1 and 4.2, respectively.

Ruler Length (RL):

$$RL = \sum_{i=1}^{n-1} (CS)_i \tag{4.1a}$$

subject to $(CS)_i \neq (CS)_j$.

Alternatively, the equation 4.1a can also be re–written as:

$$RL = IE(n) - IE(1) \tag{4.1b}$$

Total Bandwidth (TBW):

$$TBW = \sum_{i=1}^{n} (IE)_i \tag{4.2}$$

subject to $(IE)_i \neq (IE)_j$ in both equations 4.1b and 4.2, where $i, j = 1, 2, \ldots, n$ with $i \neq j$ are distinct in both equations 4.1 and 4.2.

The OGR sequences as WDM channel allocation have two optimization objectives (bi–objective) functions f_1 and f_2 with nonlinear equality and inequality constraints. The OGRs as WDM channel–allocation problem having bi–objectives can be written in general as:

$$Minimize \rightarrow \quad f_1, f_2, \tag{4.3}$$

where

$$f_1 = RL = \sum_{i=1}^{n-1} (CS)_i = IE(n) - IE(1) \tag{4.4}$$

and

$$f_2 = TBW = \sum_{i=1}^{n} (IE)_i \tag{4.5}$$

Generally, a weighted sum approach [74, 81, 82] is used to combine the multiple objective functions (here f_1 and f_2) into a single composite objective function f.

$$f = w_1 f_1 + w_2 f_2 \tag{4.6}$$

$$\text{with } w_1 + w_2 = 1 \text{ and } w_1 > 0; w_2 > 0 \tag{4.7}$$

where w_1 and w_2 represent the randomly generated weight values from a uniform distribution. A set of random numbers u_i (here i = 1,2) are generated from a uniform distribution U [0,1]. Then, the weight values w_1 and w_2 are normalized by:

$$w_1 = \frac{u_1}{u_1 + u_2} \tag{4.8}$$

and

$$w_2 = \frac{u_2}{u_1 + u_2} \tag{4.9}$$

These normalized weight values ($w_1$and w_2) serve as preferences for optimizing the bi–objective functions f_1 and f_2. The weight values can vary with sufficient diversity so as to estimate the Pareto front [64, 74, 82] efficiently.

4.1 Nature–Inspired Algorithms to Find Near–OGRs

The general pseudo–code to find near–OGR sequences as WDM channel allocation by using nature–inspired optimization algorithms proposed in this paper is shown in Figure 6. The core of the proposed nature–inspired algorithms are lines 20 to 31 which find Golomb ruler sequences for a number of iterations or until either an optimal or near–to–optimal solution is found. Also the size of the generated population must be equal at the end of iteration to the initial population size (*Popsize*). Since there are many solutions, a replacement strategy must be performed as shown in Figure 6 to remove the worst individuals. This means that the proposed algorithms maintain a fixed population of rulers and performs a fixed number of iterations until either an optimal or near–to–optimal solution is found.

5 Simulation Results

This section presents the performance of proposed nature–inspired optimization algorithms and their performance compared with best known OGRs [17, 22, 25, 40, 42, 49, 57, 58], two of the conventional computing algorithms, i.e. Extended quadratic congruence (EQC) and Search algorithm (SA) [1, 15] and three existing nature–inspired algorithms, i.e. GAs [30], BBO [30–32] and BB–BC [33, 34], of finding unequal channel spacing. All the proposed algorithms to find near–OGRs have been coded and tested in Matlab–7.12.0 (R2011a) [83] language running under Windows 7, 64–bit operating system. The proposed algorithms have been executed on a processor working at 2.20GHz with a RAM of 3GB.

```
1. Nature–inspired optimization algorithms to find near–OGRs as WDM channel allocation
2.  Begin
3.      /* Parameter initialization */
4.          Define operating parameters for nature–inspired optimization algorithms;
5.          Initialize the number of channels, lower and upper bound on the ruler length;
6.        While not Popsize                                   /* Popsize is the population size input by the user */
7.              Generate a random set of candidates (integer population);
8.                                  /* Number of integers in candidates is being equal to the number of channels */
9.              Check Golombness of each candidates;
10.             If Golombness is satisfied
11.                 Retain that candidate;
12.             Else
13.                 Remove that particular candidate from the generated population;
14.           End if
15.       End while
16.         Compute the fitness value via equations 4.4 to 4.9;
17.                          /* Fitness value represents the cost value i.e. ruler length and total optical channel bandwidth */
18.         Rank the candidates from best to worst based on fitness values;
19.    /* End of the parameter initialization */
20.    While not t                                                  /* t is a termination criterion */
21.  A:    Call any nature–inspired optimization algorithm to determine new optimal set of candidates;
22.        Recheck Golombness of updated candidates;
23.         If Golombness is satisfied
24.             Retain that candidate and then go to B;
25.         Else
26.             Retain the previous generated candidate and then go to A;
27.                     /* Previous generated candidate is being equal to the candidate generated into the parameter initialization step */
28.       End if
29.  B:    Recompute the fitness value of the modified candidates;
30.        Rank the candidates from best to worst based on fitness values and find the current best;
31.    End while
32.    Display the near–OGR sequences;
33. End
```

Figure 6: The general pseudo–code to find near–OGRs as WDM channel allocation by using nature–inspired optimization algorithms

5.1 Simulation Parameters Selection for the Proposed Algorithms

To find either optimal or near–optimal solutions after a number of careful experimentation, following optimum parameter values of proposed nature–inspired algorithms have finally been settled as shown in Tables 1 to 3. The selection of a suitable parameter values of nature– inspired algorithms are problem specific as there are no concrete rules.

Table 1: Simulation parameters for BA and MBA

Parameter	*Value*
A^0	0.8
r^0	0.5
p_m	0.01

Table 2: Simulation parameters for CSA and CSAM

Parameter	*Value*
a	0.01
p_a	0.5
p_m	0.05

Table 3: Simulation parameters for FPA and FPAM

Parameter	*Value*
γ	1.0
P	0.8
p_m	0.01

For BA, CSA, FPA, and their modified forms all the parameter values were varied from 0.0 to 1.0. By looking precisely at the obtained results, it was noted that $A = 0.8$, $r = 0.5$, $p_m = 0.01$ are the best choices for BA and MBA (Table 1), $\alpha = 0.01$, $p_a = 0.5$, $p_m = 0.05$ are the best choices for CSA and CSAM (Table 2), and $\gamma = 1.0$, $p = 0.8$, $p_m = 0.01$ are the best choices for FPA and FPAM (Table 3) to find near–OGRs.

5.2 Near–OGR Sequences

With the above mentioned parameters setting, the large numbers of sets of trials for various marks/channels were conducted. Each algorithm was executed 25 times until near–optimal solution was found. A set of 10 trials with $n = 6$, 8 and 15 for the proposed nature–inspired algorithms are reported in Tables 4 and 5. The performance of all the sets is nearly the same as reported in Tables 4 and 5. The generated near–OGR sequences for different values of marks verify that they are Golomb rulers. Although the proposed algorithms find same near–OGR sequences, the difference is in a required maximum number of iterations, computational time, and computational complexities as discussed in the following subsections.

5.3 Influence of Selecting Different Population Size on the Performance of Proposed Algorithms

In this subsection, the influence of selecting the different population size (*Popsize*) on the performance of proposed nature–inspired optimization algorithms for different values of channels is examined. The increased *Popsize* increases the diversity of potential solutions, and helps to explore the search space. But as the *Popsize* increase, the computation time required to get either the optimal or near–optimal solutions increase slightly as the diversity of possible solutions increase. But after some limit, it is not useful to increase *Popsize*, because it does not help in solving the problem faster. The choice of the best *Popsize* for nature–inspired optimization algorithms depends on the type of the problem [84]. Table 6 shows the influence of *Popsize* on the performance of the proposed algorithms to find near–OGRs for 7, 9 and 14–marks. In this experiment, all the parameter settings for proposed nature–inspired algorithms are the same as mentioned in Tables 1 to 3.

Golomb ruler sequences realized from 10 to 16–marks by Tabu search algorithm [26], the maximum *Popsize* was set to 190. In the hybrid approach proposed in [29] to find Golomb rulers from 11 to 23–marks, the *Popsize* was set between 20 and 2000. The near–OGR sequences found by algorithms GAs and BBO [30], maximum *Popsize* was 30. The BB–BC algorithm proposed in [33, 34] finds near–OGRs upto 11–marks with a maximum *Popsize* of 10. For the hybrid evolutionary (HE) algorithms [39], to find near–OGRs, maximum *Popsize* was 50.

From Table 6, it is noted that *Popsize* has little effect on the performance of all proposed nature–inspired optimization algorithms. By carefully looking at the results, the *Popsize* of 10 in all proposed algorithms was found to be adequate for finding near–OGR sequences.

5.4 Influence of Increasing Iterations on Total Optical Channel Bandwidth

The choice of the best maximum iteration (*Maxiter*) for nature–inspired metaheuristic algorithms is always critical for specific problems. Increasing the numbers of iteration will increase the possibility of reaching

optimal solutions and promoting the exploitation of the search space. This means, the chance to find the correct search direction increases considerably.

Here, all the parameter values for proposed nature–inspired algorithms are the same as mentioned in above subsections. By increasing the number of iterations, the total optical channel bandwidth tends to decrease; it means that the rulers reach their optimal or near–to–optimal values after certain iterations. This is the point where no further improvement is seen. Tables 7 and 8 illustrate the influence of increasing iterations on the performance of proposed algorithms for various channels. It is noted that the iterations have little effect for low value marks (such as $n = 7$ and 8). But for higher order marks, the iterations have a great effect on the total optical channel bandwidth i.e. total optical bandwidth gets optimized after a certain number of iterations.

In the literatures [26, 29], the *Maxiter* for Tabu search algorithm to find Golomb ruler sequences were set to 10000 and 30000, respectively. For the hybrid approach proposed in [29] to find Golomb ruler sequences the maximum number of iterations set were 100000. In [30], it was noted that to find near–OGRs, GAs and BBO algorithms stabilized in and around 5000 iterations, while hybrid evolutionary algorithms [39] were stabilized in and around 10000 iterations. By carefully looking at the results, it is concluded that all the proposed optimization algorithms in this paper to find either optimal or near–optimal Golomb rulers, stabilized at or around 1000 iterations.

5.5 Performance Comparison of Proposed Algorithms with Previous Existing Algorithms in Terms of Ruler Length and Total Optical Channel Bandwidth

Table 9 enlists the ruler length and total occupied channel bandwidth by different sequences obtained from the proposed nature–inspired algorithms after 25 executions and their performance compared with best known OGRs (best solutions), EQC, SA, GAs, BBO and BB–BC. According to [1], the applications of EQC and SA is restricted to prime powers only, so the ruler length and total occupied channel bandwidth for EQC and SA are presented by a dash line in Table 9. Comparing the experimental results obtained from the proposed algorithms with best known OGRs and existing algorithms, it is noted that there is a significant improvement in the ruler length and thus the total occupied channel bandwidth, that is the results become improved.

From Table 9, it is also observed that simulation results are particularly impressive. First, observe that the algorithms BA, BAM and LBA can find best rulers up to 17–marks and near–optimal rulers for 18 to 20–marks. The algorithms LBAM, CSA, CSAM, FPA and FPAM can find best rulers up to 20–marks very efficiently and effectively in a reasonable computational time.

From the simulation results, it is concluded that modified forms of the proposed nature–inspired algorithms to find either optimal or near–OGRs slightly outperform the algorithms presented in their simplified forms. As illustrated in Table 9 for higher order marks, the proposed algorithms outperform the other existing algorithms in terms of both the ruler length and total occupied channel bandwidth.

5.6 Performance Comparison of Proposed Algorithms in Terms of Computational Time

Finding Golomb ruler sequences is an extremely challenging optimization problem. The OGRs generation by exhaustive parallel search algorithms for higher order marks is computationally very time consuming, which took several hours, months, even years of calculation on the network of several thousand computers [19, 25, 42, 43, 57, 58]. For example, rulers with 20 to 27–marks were found by distributed OGR project [43] which took several years of calculations on many computers to prove the optimality of the rulers.

This subsection is devoted to report the experimental average CPU time taken to find either optimal or near–optimal Golomb rulers by the proposed algorithms and their comparison with the computation time taken by existing algorithms [22, 25, 27, 29, 30, 33–36, 42, 43]. Table 10 reports the average CPU time taken by proposed algorithms to find near–OGRs up to 20–marks. The experimental CPU time taken by BB–BC

algorithm to find near–OGRs is not reported in [33]. Then, using the same parameter values as mentioned in [33], algorithm BB–BC to find near–OGRs was executed to obtain the average computational CPU time. Figure 7 illustrates the graphical representation of the Table 10.

In [27], it is identified that to find Golomb ruler sequences from heuristic based exhaustive search algorithm, the times varied from 0.035 seconds to 6 weeks for 5 to 13–marks ruler, whereas by non–heuristic exhaustive search algorithms took approximately 12.57 minutes for 10–marks, 2.28 years for 12–marks, 2.07e+04 years for 14–marks, 3.92e+09 years for 16–marks, 1.61e+15 years for 18–marks and 9.36e+20 years for 20–marks ruler. In [29], it is reported that CPU time taken by Tabu search algorithm to find OGRs is around 0.1 second for 5–marks, 720 seconds for 10–marks, 960 seconds for 11–marks, 1913 seconds for 12–marks and 2516 seconds (around 41 minutes) for 13–marks. The OGRs realized by hybrid Genetic algorithm [29] took around 5 hours for 11–marks, 8 hours for 12–marks, and 11 hours for 13–marks. The OGRs realized by the exhaustive search algorithms in [22] for 14 and 16–marks, took nearly one hour and hundred hours, respectively, while 17, 18 and 19–marks OGRs realized in [25] and [42], took around 1440, 8600 and 36200 CPU hours (nearly seven months) respectively on a Sun Sparc Classic workstation. Also, the near–OGRs realized up to 20–marks by algorithms GAs and BBO [30], the maximum execution time was approximately 31 hours i.e. nearly 1.3 days, while for BB–BC [33] the maximum execution time was around 28 hours, i.e. almost 1.1 days.

From Table 10, it is noted that for proposed algorithms, the average CPU time varied from 0.0 second for 3–marks ruler to approximately 23 hours for 20–marks ruler. The maximum execution time taken by the proposed algorithms BA and FPA for 20–marks ruler is about 23 and 19 hours respectively. By introducing the concept of mutation and Lévy flight strategies with the proposed nature–inspired algorithms, the minimum execution time for algorithm FPAM is reduced to approximately 18 hours i.e. less than one day. This represents the improvement achieved by the use of proposed optimization algorithms and their modified forms to find near–OGR sequences. From Table 10, it is further observed that algorithm FPAM outperforms the other proposed optimization algorithms in terms of computational time.

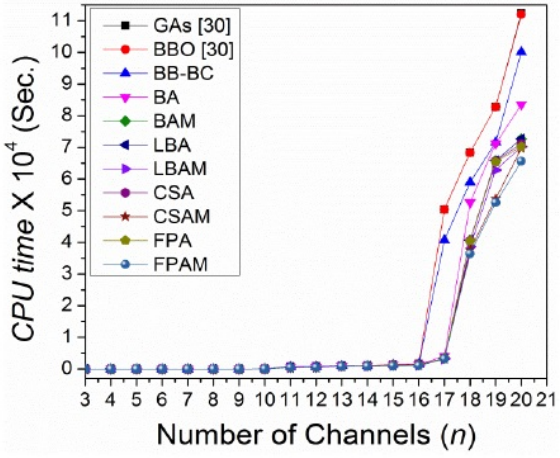

Figure 7: Performance comparison of proposed algorithms in terms of computational time

5.7 Performance Comparison of Proposed Algorithms in Terms of Computation Complexity

To arrive at optimal solutions, the proposed algorithms have an initialization stage and a subsequent stage of iterations. The computational complexity of proposed algorithms depends upon *Popsize* and *Maxiter*:

$$Computation\ Complexity = Popsize \times Maxiter \tag{5.1}$$

To find Golomb rulers by Tabu search algorithm [29], the maximum computation complexity was 57e+05 (190×30000). The maximum computation complexity for hybrid approach proposed in [29] was 2e+08 (2000×100000), whereas for GAs and BBO proposed in [30], the maximum computation complexity was 15e+04 (30×5000).

Thus, to generate 18–marks near–OGRs the computation complexity for BA, BAM, LBA, LBAM, CSA, CSAM, FPA, and FPAM is 8000, 7600, 7400, 7000, 6500, 6300, 6200, and 6100, respectively. Thus, it is clear that the algorithm FPAM outperforms the other proposed algorithms in terms of computation complexity. It is also further concluded that by introducing the concept of mutation and Lévy flight strategies in the simplified form of nature–inspired algorithms the computation complexity is reduced.

In general, the maximum computation complexity for all proposed algorithms to generate near–OGRs up to 20–marks is 10*1000=1e+04.

5.8 Statistical Analysis of Proposed Algorithms and their Comparison

In order to evaluate and compare the performance of the proposed different nature–inspired optimization algorithms, the statistical analysis test of each proposed algorithm is performed. For the statistical analysis, each proposed metaheuristic optimization algorithm was executed 25 times using the same parameter values as reported in the above subsections. The obtained optimum values of *RL* and *TBW* (Hz) along with computational *CPU time* (Sec.) after each trial were noted for the statistical analysis test. Tables 11 and 12 report statistical analysis in the form of mean ± the standard deviation of the proposed optimization algorithms to find optimal and near–optimal Golomb rulers for various channels (the best values are in bold).

From Tables 11 and 12, it is clear that the algorithm LBAM performs much better than BA, BAM and LBA, whereas the algorithm CSAM outperforms LBAM and CSA, while the algorithm FPAM is much superior to FPA and CSAM in terms of mean and standard deviation. It suggests that the algorithm FPAM performs better than all the other algorithms presented in the standard and their modified forms to find OGRs for optical WDM systems. In order to analyze whether the OGRs found by algorithm FPAM are significant or not, non–parametric Friedman's statistical test [85] on the ruler length, total bandwidth and CPU time is carried out to rank the proposed algorithms. For the statistical analysis, two hypotheses, the null hypothesis H_0 and the alternative hypothesis H_1, are defined. The null hypothesis states that there is no difference or no change, whereas the alternative hypothesis represents the presence of a difference or change among the proposed nature–inspired metaheuristic algorithms. In order to reject a null hypothesis H_0 in statistical analysis procedure, a 95 % confidence level of significance is used to estimate at which level the null hypothesis H_0 can be rejected. In statistical analysis, a p–value is determined to estimate whether a statistical hypothesis test for the proposed algorithms is significant or not, and it also provides information about how significant the result is. The aim to use Friedman's non–parametric statistical test for multiple comparison in this paper was to examine the statistical difference or change between the performances of all the proposed algorithms for OGRs problem in optical WDM systems. Table 13 reports the average ranking obtained by Friedman's statistical test in terms of *RL*, *TBW* (Hz) and *CPU time* (Sec.). At a 95 % confidence level of significance with 7 as degrees of freedom the critical value in χ^2 distribution is 2.167. As mentioned in Table 13, the estimated Friedman's statistic value for *RL*, *TBW* (Hz), and *CPU time* (Sec.) are larger than the critical values in χ^2 distribution. It further suggests that there is a significant difference between the performance of proposed optimization metaheuristic algorithms i.e. the null hypothesis H_0 is rejected. The algorithm FPAM is with the lowest rank and outperforms the other proposed algorithms. The estimated p–value in Friedman's statistic is <0.00001 for *RL*, *TBW* and *CPU time*.

In order to statistically estimate better performance of the algorithm FPAM to find OGRs, the Holm's post–hoc statistical test [86] is being conducted using FPAM as a control algorithm. The unadjusted and adjusted p–values estimated through the Holm's post–hoc statistical procedure is reported in Table 14. It can be seen from Table 14 that the algorithm FPAM is potentially better than all other proposed algorithms (standard and their modified forms) for OGRs problem in terms of both efficiency and success rate at 95% confidence level. This is no surprise as the aim of developing the new metaheuristic algorithms and their modified forms to find OGRs was to try to use the advantages of mutation strategy and random walk via. Lévy–flight distribution.

Table 4: Performance of Proposed BA and MBA Algorithms to find Near–OGRs for Various channels in a set of 10 trials

Trials	BA									MBA																				
										BAM									LBA						LBAM					
	n=6			n=8			n=15			n=6			n=8			n=15			n=6			n=8			n=15			n=6			n=8			n=15		
	RL	TBW (Hz)	CPU Time (Sec.)	RL	TBW (Hz)	CPU Time (Sec.)	RL	TBW (Hz)	CPU Time (Sec.)	RL	TBW (Hz)	CPU Time (Sec.)	RL	TBW (Hz)	CPU Time (Sec.)	RL	TBW (Hz)	CPU Time (Sec.)	RL	TBW (Hz)	CPU Time (Sec.)	RL	TBW (Hz)	CPU Time (Sec.)	RL	TBW (Hz)	CPU Time (Sec.)	RL	TBW (Hz)	CPU Time (Sec.)	RL	TBW (Hz)	CPU Time (Sec.)	RL	TBW (Hz)	CPU Time (Sec.)
1	17	44	0.4349	34	117	1.0168	151	1047	1.289e+03	17	44	0.0525	34	117	0.1416	151	1047	1.165e+03	17	44	0.0525	39	113	0.1411	151	1047	1.158e+03	18	42	0.0519	34	117	0.1378	151	1047	1.150e+03
2	17	44	0.4291	34	117	1.0169	151	1047	1.260e+03	17	44	0.0524	34	117	0.1408	151	1047	1.158e+03	17	44	0.0522	39	113	0.1389	151	1047	1.159e+03	18	42	0.0511	34	117	0.1381	151	1047	1.147e+03
3	17	44	0.4235	39	113	1.0186	151	1047	1.256e+03	17	44	0.0522	34	117	0.1412	151	1047	1.157e+03	17	44	0.0520	34	117	0.1388	151	1047	1.162e+03	18	42	0.0510	34	117	0.1379	151	1047	1.147e+03
4	18	42	0.4248	34	117	1.0216	151	1047	1.258e+03	17	44	0.0521	34	117	0.1407	151	1047	1.160e+03	17	44	0.0519	34	117	0.1421	151	1047	1.158e+03	17	44	0.0512	34	117	0.1378	151	1047	1.148e+03
5	17	44	0.4267	34	117	1.0175	151	1047	1.259e+03	17	44	0.0521	34	117	0.1410	151	1047	1.159e+03	17	44	0.0518	34	117	0.1367	151	1047	1.154e+03	17	44	0.0513	34	117	0.1379	151	1047	1.149e+03
6	18	42	0.4237	34	117	1.0177	151	1047	1.239e+03	17	44	0.0519	34	117	0.1411	151	1047	1.155e+03	17	44	0.0521	34	117	0.1411	151	1047	1.157e+03	17	44	0.0510	34	117	0.1378	151	1047	1.159e+03
7	18	42	0.4101	34	117	1.0158	151	1047	1.242e+03	18	42	0.0522	39	113	0.1413	151	1047	1.157e+03	17	44	0.0522	34	117	0.1369	151	1047	1.156e+03	17	44	0.0511	39	113	0.1375	151	1047	1.146e+03
8	17	44	0.4267	39	113	1.0169	151	1047	1.258e+03	17	44	0.0521	34	117	0.1414	151	1047	1.159e+03	18	42	0.0520	34	117	0.1367	151	1047	1.165e+03	17	44	0.0510	34	117	0.1379	151	1047	1.139e+03
9	17	44	0.4236	39	113	1.0177	151	1047	1.257e+03	17	44	0.0520	39	113	0.1408	151	1047	1.158e+03	17	44	0.0523	34	117	0.1385	151	1047	1.157e+03	17	44	0.0500	34	117	0.1382	151	1047	1.148e+03
10	17	44	0.4221	39	113	1.0178	151	1047	1.261e+03	17	44	0.0519	34	117	0.1410	151	1047	1.156e+03	18	42	0.0518	34	117	0.1382	151	1047	1.159e+03	17	44	0.051	39	113	0.1381	151	1047	1.146e+03
	Optimal *RL* =17 Optimal *TBW* = 42 Hz Minimum CPU Time = 0.4101 Sec. Average CPU Time = 0.42452 Sec.			Optimal *RL* =34 Optimal *TBW* = 113 Hz Minimum CPU Time = 1.0158 Sec. Average CPU Time = 1.01773 Sec.			Optimal *RL* =151 Optimal *TBW* = 1047 Hz Minimum CPU Time = 1.239e+03 Sec. Average CPU Time = 1.258e+03 Sec.			Optimal *RL* =17 Optimal *TBW* = 42 Hz Minimum CPU Time = 0.0519 Sec. Average CPU Time = 0.05214 Sec.			Optimal *RL* =34 Optimal *TBW* = 113 Hz Minimum CPU Time = 0.1407 Sec. Average CPU Time = 0.14109 Sec.			Optimal *RL* =151 Optimal *TBW* = 1047 Hz Minimum CPU Time = 1.155e+03 Sec. Average CPU Time = 1.158e+03 Sec.			Optimal *RL* =17 Optimal *TBW* = 42 Hz Minimum CPU Time = 0.0518 Sec. Average CPU Time = 0.05208 Sec.			Optimal *RL* =34 Optimal *TBW* = 113 Hz Minimum CPU Time = 0.1367 Sec. Average CPU Time = 0.1389 Sec.			Optimal *RL* =151 Optimal *TBW* = 1047 Hz Minimum CPU Time = 1.154e+03 Sec. Average CPU Time = 1.159e+03 Sec.			Optimal *RL* =17 Optimal *TBW* = 42 Hz Minimum CPU Time = 0.05 Sec. Average CPU Time = 0.05106 Sec.			Optimal *RL* =34 Optimal *TBW* = 113 Hz Minimum CPU Time = 0.1375 Sec. Average CPU Time = 0.1379 Sec.			Optimal *RL* =151 Optimal *TBW* = 1047 Hz Minimum CPU Time = 1.139e+03 Sec. Average CPU Time = 1.148e+03 Sec.		

Table 5: Performance of Proposed CSA, CSAM, FPA and FPAM Algorithms to find Near–OGRs for Various channels in a set of 10 trials

Trials	CSA									CSAM									FPA									FPAM								
	n=6			n=8			n=15			n=6			n=8			n=15			n=6			n=8			n=15			n=6			n=8			n=15		
	RL	TBW (Hz)	CPU Time	RL	TBW (Hz)	CPU Time (Sec.)	RL	TBW (Hz)	CPU Time (Sec.)	RL	TBW (Hz)	CPU Time (Sec.)	RL	TBW (Hz)	CPU Time (Sec.)	RL	TBW (Hz)	CPU Time (Sec.)	RL	TBW (Hz)	CPU Time (Sec.)	RL	TBW (Hz)	CPU Time (Sec.)	RL	TBW (Hz)	CPU Time (Sec.)	RL	TBW (Hz)	CPU Time (Sec.)	RL	TBW (Hz)	CPU Time (Sec.)	RL	TBW (Hz)	CPU Time (Sec.)
1	18	42	0.0517	39	113	0.1375	151	1047	1.135e+03	17	44	0.0508	34	117	0.1367	151	1047	1.090e+03	18	42	0.0505	34	117	0.1367	151	1047	1.085e+03	17	44	0.0499	39	113	0.1349	151	1047	1.032e+03
2	18	42	0.0518	39	113	0.1376	151	1047	1.136e+03	18	42	0.0509	34	117	0.1368	151	1047	1.089e+03	17	44	0.0507	34	117	0.1365	151	1047	1.076e+03	17	44	0.0484	34	117	0.1351	151	1047	1.034e+03
3	17	44	0.0510	39	113	0.1389	151	1047	1.132e+03	17	44	0.0511	34	117	0.1369	151	1047	1.078e+03	17	44	0.0504	34	117	0.1362	151	1047	1.077e+03	17	44	0.0494	34	117	0.1348	151	1047	1.033e+03
4	17	47	0.0490	34	117	0.1379	151	1047	1.135e+03	17	44	0.0507	34	117	0.1371	151	1047	1.084e+03	17	44	0.0498	34	117	0.1365	151	1047	1.078e+03	17	44	0.0491	34	117	0.1353	151	1047	1.030e+03
5	18	42	0.0517	34	117	0.1374	151	1047	1.137e+03	17	44	0.0511	34	117	0.1368	151	1047	1.085e+03	17	44	0.0521	34	117	0.1364	151	1047	1.081e+03	17	44	0.0480	39	113	0.1347	151	1047	1.029e+03
6	17	44	0.0511	34	117	0.1378	151	1047	1.136e+03	17	44	0.0490	34	117	0.1364	151	1047	1.084+03	17	44	0.0489	34	117	0.136	151	1047	1.069e+03	17	44	0.0495	34	117	0.1346	151	1047	1.035e+03
7	17	44	0.0515	34	117	0.1380	151	1047	1.137e+03	17	44	0.0505	39	113	0.1369	151	1047	1.088e+03	17	44	0.0504	34	117	0.1371	151	1047	1.075e+03	17	44	0.0492	34	117	0.1341	151	1047	1.029e+03
8	18	42	0.0513	39	113	0.1377	151	1047	1.134e+03	17	44	0.0506	34	117	0.1368	151	1047	1.084e+03	17	44	0.0502	34	117	0.1365	151	1047	1.079e+03	17	44	0.0512	34	117	0.1346	151	1047	1.028e+03
9	17	44	0.0512	34	117	0.1379	151	1047	1.129e+03	17	44	0.0502	34	117	0.1368	151	1047	1.086e+03	17	44	0.0503	39	113	0.1364	151	1047	1.078e+03	17	42	0.0497	34	117	0.1348	151	1047	1.029e+03
10	18	42	0.0570	34	117	0.1372	151	1047	1.133e+03	17	44	0.0508	34	117	0.1369	151	1047	1.085e+03	17	47	0.0508	34	117	0.1367	151	1047	1.077e+03	18	42	0.0493	34	117	0.1349	151	1047	1.026e+03
	Optimal *RL* =17 Optimal *TBW* = 42 Hz Minimum *CPU Time* = 0.0490 Sec. Average *CPU Time* = 0.05173 Sec.			Optimal *RL* =34 Optimal *TBW* =113 Hz Minimum *CPU Time* = 0.1372 Sec. Average *CPU Time* = 0.13779 Sec.			Optimal *RL* =151 Optimal *TBW* = 1047 Hz Minimum *CPU Time* = 1.129e+03 Sec. Average *CPU Time* = 1.134e+03 Sec.			Optimal *RL* =17 Optimal *TBW* = 42 Hz Minimum *CPU Time* = 0.0490 Sec. Average *CPU Time* = 0.05057 Sec.			Optimal *RL* =34 Optimal *TBW* = 113 Hz Minimum *CPU Time* = 0.1364 Sec. Average *CPU Time* = 0.13681 Sec.			Optimal *RL* =151 Optimal *TBW* = 1047 Hz Minimum *CPU Time* = 1.078e+03 Sec. Average *CPU Time* = 1.085e+03 Sec.			Optimal *RL* =17 Optimal *TBW* = 42 Hz Minimum *CPU Time* = 0.0489 Sec. Average *CPU Time* = 0.05041 Sec.			Optimal *RL* =34 Optimal *TBW* = 113 Hz Minimum *CPU Time* = 0.1360 Sec. Average *CPU Time* = 0.1365 Sec.			Optimal *RL* =151 Optimal *TBW* = 1047 Hz Minimum *CPU Time* = 1.069e+03 Sec. Average *CPU Time* = 1.078e+03 Sec.			Optimal *RL* =17 Optimal *TBW* = 42 Hz Minimum *CPU Time* = 0.0480 Sec. Average *CPU Time* = 0.04937 Sec.			Optimal *RL* =34 Optimal *TBW* = 113 Hz Minimum *CPU Time* = 0.1341 Sec. Average *CPU Time* = 0.13478 Sec.			Optimal *RL* =151 Optimal *TBW* = 1047 Hz Minimum *CPU Time* = 1.026e+03 Sec. Average *CPU Time* = 1.031e+03 Sec.		

Table 6: Influence of Population Size on the Performance of BA, MBA, CSA, CSAM, FPA and FPAM Algorithms to find Near–OGRs for Various Channels

Popsize	BA						MBA																		CSA						CSAM						FPA						FPAM					
							BAM						LBA						LBAM																															
	n=7		n=9		n=14		n=7		n=9		n=14		n=7		n=9		n=14		n=7		n=9		n=14		n=7		n=9		n=14		n=7		n=9		n=14		n=7		n=9		n=14		n=7		n=9		n=14	
	RL	TBW (Hz)	RL	TBW (Hz)	RL	TBW (Hz)	RL	TBW (Hz)	RL	TBW (Hz)	RL	TBW (Hz)	RL	TBW (Hz)	RL	TBW (Hz)	RL	TBW (Hz)	RL	TBW (Hz)	RL	TBW (Hz)	RL	TBW (Hz)	RL	TBW (Hz)	RL	TBW (Hz)	RL	TBW (Hz)	RL	TBW (Hz)	RL	TBW (Hz)	RL	TBW (Hz)	RL	TBW (Hz)	RL	TBW (Hz)	RL	TBW (Hz)	RL	TBW (Hz)	RL	TBW (Hz)	RL	TBW (Hz)
10	25	77	44	206	127	927	25	77	44	206	127	927	25	77	44	206	127	927	25	77	44	206	127	927	25	77	44	206	127	927	25	77	44	206	127	927	25	77	44	206	127	927	25	77	44	206	127	927
20	25	77	44	206	127	927	25	77	44	206	127	927	25	77	44	206	127	927	25	77	44	206	127	927	25	77	44	206	127	927	25	77	47	185	127	927	25	77	44	206	127	927	25	77	44	206	127	927
50	25	77	44	206	127	927	25	77	44	206	127	927	25	77	58	177	127	927	25	77	44	206	127	927	25	77	44	206	127	927	25	77	44	206	127	927	25	77	44	206	127	927	25	77	44	206	127	927
80	25	77	44	206	127	927	27	73	44	206	127	927	25	77	44	206	127	927	25	77	44	206	127	927	25	77	44	206	127	927	25	77	44	206	127	927	25	77	44	206	127	927	25	77	55	176	127	927
100	28	74	44	206	127	927	27	73	57	183	127	927	27	73	44	206	127	927	25	77	47	185	127	927	27	73	44	206	127	927	27	73	44	206	127	927	25	77	44	206	127	927	25	77	44	206	127	927

Table 7: Influence of Increasing Iterations on the Performance of BA and MBA to find Near–OGRs for Various Channels

Iterations	TBW (Hz)																																														
	BA										MBA																																					
											BAM										LBA										LBAM																			
	n=7	n=8	n=9	n=10	n=11	n=12	n=13	n=14	n=16	n=18	n=7	n=8	n=9	n=10	n=11	n=12	n=13	n=14	n=16	n=18	n=7	n=8	n=9	n=10	n=11	n=12	n=13	n=14	n=16	n=18	n=7	n=8	n=9	n=10	n=11	n=12	n=13	n=14	n=16	n=18	n=7	n=8	n=9	n=10	n=11	n=12	n=13	n=14	n=16	n=18
5	77	117	342	382	689	870	1156	2187	2519	6111	73	113	206	285	550	670	1120	1583	2347	6024	73	113	342	286	509	666	1134	1775	2233	5756	73	113	216	267	490	628	1064	1308	2115	5704												
50	74	113	296	376	593	736	1022	2081	2519	5820	73	113	183	282	450	605	1046	1290	2347	5601	73	113	206	283	397	643	970	1390	2233	5723	73	113	185	249	425	612	1064	1166	1960	5704												
100	74	113	206	249	498	711	923	2081	2484	5775	73	113	183	249	437	581	935	1246	2215	5258	73	113	177	249	395	611	969	1390	2149	5656	73	113	185	249	386	609	951	1166	1960	5515												
150	74	113	206	249	386	682	886	1937	2407	5302	73	113	183	249	386	503	861	1246	2143	5098	73	113	177	249	391	503	969	1290	2143	4809	73	113	185	249	386	503	884	991	1845	4675												
250	74	113	206	249	386	503	794	1332	2149	5205	73	113	183	249	386	503	786	1177	1985	4579	73	113	177	249	391	503	880	1193	1960	4488	73	113	185	249	386	503	794	1001	1845	3680												
350	74	113	206	249	386	503	660	1332	2149	4809	73	113	183	249	386	503	660	991	1985	4108	73	113	177	249	391	503	660	1193	1960	4191	73	113	185	249	386	503	660	1001	1540	3488												
500	74	113	206	249	386	503	660	1159	1958	3927	73	113	183	249	386	503	660	924	1804	3822	73	113	177	249	391	503	660	924	1834	3927	73	113	185	249	386	503	660	924	1365	3213												
600	74	113	206	249	386	503	660	924	1958	3595	73	113	183	249	386	503	660	924	1298	3272	73	113	177	249	391	503	660	924	1298	3680	73	113	185	249	386	503	660	924	1298	2912												
700	74	113	206	249	386	503	660	924	1298	3310	73	113	183	249	386	503	660	924	1298	2934	73	113	177	249	391	503	660	924	1298	3264	73	113	185	249	386	503	660	924	1298	1894												
800	74	113	206	249	386	503	660	924	1298	3079	73	113	183	249	386	503	660	924	1298	2566	73	113	177	249	391	503	660	924	1298	2912	73	113	185	249	386	503	660	924	1298	1894												
900	74	113	206	249	386	503	660	924	1298	3079	73	113	183	249	386	503	660	924	1298	2566	73	113	177	249	391	503	660	924	1298	2912	73	113	185	249	386	503	660	924	1298	1894												
1000	74	113	206	249	386	503	660	924	1298	3079	73	113	183	249	386	503	660	924	1298	2566	73	113	177	249	391	503	660	924	1298	2912	73	113	185	249	386	503	660	924	1298	1894												

Table 8: Influence of Increasing Iterations on the Performance of CSA, CSAM, FPA, and FPAM Algorithms to find Near–OGRs for Various Channels

Iterations	TBW (Hz)																																														
	CSA										CSAM										FPA										FPAM																	
	n=7	n=8	n=9	n=10	n=11	n=12	n=13	n=14	n=16	n=18	n=7	n=8	n=9	n=10	n=11	n=12	n=13	n=14	n=16	n=18	n=7	n=8	n=9	n=10	n=11	n=12	n=13	n=14	n=16	n=18	n=7	n=8	n=9	n=10	n=11	n=12	n=13	n=14	n=16	n=18										
5	74	117	342	392	609	681	1047	1544	2061	5601	73	113	183	283	521	693	1048	1439	2059	5515	73	113	262	278	544	653	1025	1286	1985	5483	73	113	177	292	488	611	1010	1172	1978	5217										
50	73	113	216	258	499	648	987	1436	2058	5483	73	113	176	249	489	679	923	1166	1958	5302	73	113	206	249	392	581	998	1286	1804	5258	73	113	176	249	386	609	904	1172	1799	4953										
100	73	113	206	249	399	606	857	1210	1877	4779	73	113	176	249	448	503	727	1001	1747	4649	73	113	206	249	386	503	876	1159	1665	5065	73	113	176	249	378	503	786	1001	1667	4740										
150	73	113	206	249	391	503	756	1210	1647	3822	73	113	176	249	378	503	669	1001	1544	4458	73	113	206	249	386	503	774	1012	1542	3927	73	113	176	249	378	503	698	993	1539	3822										
250	73	113	206	249	391	503	660	987	1427	3272	73	113	176	249	378	503	660	996	1497	3673	73	113	206	249	386	503	660	956	1355	3698	73	113	176	249	378	503	660	991	1342	3665										
350	73	113	206	249	391	503	660	924	1323	3221	73	113	176	249	378	503	660	924	1365	3488	73	113	206	249	386	503	660	924	1323	3412	73	113	176	249	378	503	660	924	1321	3405										
500	73	113	206	249	391	503	660	924	1298	3083	73	113	176	249	378	503	660	924	1298	3079	73	113	206	249	386	503	660	924	1298	3100	73	113	176	249	378	503	660	924	1298	3062										
600	73	113	206	249	391	503	660	924	1298	2806	73	113	176	249	378	503	660	924	1298	2725	73	113	206	249	386	503	660	924	1298	2678	73	113	176	249	378	503	660	924	1298	2566										
700	73	113	206	249	391	503	660	924	1298	1894	73	113	176	249	378	503	660	924	1298	1894	73	113	206	249	386	503	660	924	1298	1894	73	113	176	249	378	503	660	924	1298	1894										
800	73	113	206	249	391	503	660	924	1298	1894	73	113	176	249	378	503	660	924	1298	1894	73	113	206	249	386	503	660	924	1298	1894	73	113	176	249	378	503	660	924	1298	1894										
900	73	113	206	249	391	503	660	924	1298	1894	73	113	176	249	378	503	660	924	1298	1894	73	113	206	249	386	503	660	924	1298	1894	73	113	176	249	378	503	660	924	1298	1894										
1000	73	113	206	249	391	503	660	924	1298	1894	73	113	176	249	378	503	660	924	1298	1894	73	113	206	249	386	503	660	924	1298	1894	73	113	176	249	378	503	660	924	1298	1894										

Table 9: Performance Comparison of Proposed Nature–Inspired Optimization Algorithms to find Near–OGRs for Various Channels

n	Best Known OGRs [17, 22, 25, 40] [42, 49, 57, 58]		ALGORITHMS																									
			Conventional Algorithms				Existing Nature–Inspired Algorithms										Proposed Nature–Inspired Algorithms											
			EQC[1, 15]		SA[1, 15]		GAs[30]		BBO[30]		BB–BC[33, 34]		BA		BAM		LBA		LBAM		CSA		CSAM		FPA		FPAM	
	RL	*TBW* (Hz)	*RL*	*TBW* (Hz)	*RL*	*TBW* (Hz)	*RL*	*TBW* (Hz)	*RL*	*TBW* (Hz)	*RL*	*TBW* (Hz)	*RL*	*TBW* (Hz)	*RL*	*TBW* (Hz)	*RL*	*TBW* (Hz)	*RL*	*TBW* (Hz)	*RL*	*TBW* (Hz)	*RL*	*TBW* (Hz)	*RL*	*TBW* (Hz)	*RL*	*TBW* (Hz)
3	3	4	6	10	6	4	3	4	3	4	3	4	3	4	3	4	3	4	3	4	3	4	3	4	3	4	3	4
4	6	11	15	28	15	28	6 7	11	6 7	11	6 7	11	6 7	11	6 7	11	6 7	11	6	11	6 7	11	6	11	6 7	11	6	11
5	11	25 28	–	–	–	–	12 13	23 25 29	12 13	23 24	11 12	23 25	11 12	23 25	11 12 13	23 25	11 12	23 25 28	11 12 13	23 24 25	11 12	23 25	11 12	23 25	11 12	23 25	11 12	23 25
6	17	44 47 50 52	45	140	20	60	17 18 21	42 44 45	17 18 20 21	42 43 44 45 49	17 18	42 44	17 18	42 44	17 18	42 44	17 18	42 44 47 50	17 18	42 44 47 50	17 18	42 44 47 50	17 18	42 44 47	17 18	42 44 47	17 18	42 44 47 50
7	25	77 81 87 90 95	– – –	–	–	–	27 28 29 30 31 32	73 78 79 80 83 86 95	27 29 30 31 32 33	73 82 83 84 91 95	25 26 28 30	73 74 77 81	25 28	74 77 81 90	25 26 27	73 77	25 27	73 77	25 26 27	73 77	25 26 27	73 77 80	25 26 27	73 77	25 26 27	73 77 80 81	25 26 27	73 77
8	34	117	91	378	49	189	35 41 42 45 46	121 126 128 129 131 133	34 39 40 42	121 125 127 131	39 41 42	113 118 119	34 39	113 117	34 39	113 117	34 39	113 117	34 39	113 117	34 39	113 117	34 39	113 117	34 39	113 117	34 39	113 117
9	44	206	–	–	–	–	52 56 59 61 63 65	192 193 196 203 225	49 56 61 62 64	187 200 201 206 215	44 45 46 61	179 248 253 262	44	206	44 57	183 206	44 58	177 206	44 47	185 206	44	206	44 47 55	176 185 206	44	206	44 47 55	176 185 206
10	55	249	–	–	–	–	75 76	283 287 301	74	274	77	258	55	249	55	249	55	249	55	249	55	249	55	249	55	249	55	249
11	72	386 391	–	–	–	–	94 96	395 456	86 103 104 114 118	378 435 440 491	72 105	377 490 456	72	386	72	386	72	391	72	386 391	72	391	72 103	378 386	72	386	72 103	378 386
12	85	503	231	1441	132	682	123 128 137	532 581 660	116 124 138	556 590 605	85 91	550 580 613	85	503	85	503	85	503	85	503	85	503	85	503	85	503	85	503
13	106	660	–	–	–	–	203 241	1015 1048	156 171 187	768 786 970	110 113	768 753	106	660	106	660	106	660	106	660	106	660	106	660	106	660	106	660
14	127	924	325	2340	286	1820	206 228 230	1172 1177 1285	169 206 221	991 1001 1166	221	1166	127	924	127	924	127	924	127	924	127	924	127	924	127	924	127	924
15	151	1047	–	–	–	–	275 298	1634 1653	260 267	1322 1554	267	1322	151	1047	151	1047	151	1047	151	1047	151	1047	151	1047	151	1047	151	1047
16	177	1298	–	–	–	–	316	1985	283	1804	316	1985	177	1298	177	1298	177	1298	177	1298	177	1298	177	1298	177	1298	177	1298
17	199	1661	–	–	–	–	355	2205	354 369	2201 2208	369	2201	199	1661	199	1661	199	1661	199	1661	199	1661	199	1661	199	1661	199	1661
18	216	1894	561	5203	493	5100	427 463	2599 3079	362 445	2566 2912	427	3079	427	3079	445	2566	362	2912	216	1894	216	1894	216	1894	216	1894	216	1894
19	246	2225	–	–	–	–	567 597	3432 5067	467 475 584	3337 3408 4101	584	4101	467	3337	467	3337	475	3408	246	2225	246	2225	246	2225	246	2225	246	2225
20	283	2794	703	7163	703	6460	615 673 680 691	4660 4826 4905 4941	578 593 649	4306 4517 4859	691	4941	578 615	4306 4660	578	4306	578 593	4306 4859	283	2794	283	2794	283	2794	283	2794	283	2794

Table 10: Comparison of Average CPU Time Taken by Proposed Optimization Algorithms to find Near–OGRs for Various Channels

n	Algorithms										
	GAs[30]	BBO[30]	BB–BC	BA	BAM	LBA	LBAM	CSA	CSAM	FPA	FPAM
	CPU time (Sec.)	*CPU time* (Sec.)	*CPU time* (Sec.)	*CPU time* (Sec.)	*CPU time* (Sec.)	*CPU time* (Sec.)	*CPU time* (Sec.)	*CPU time* (Sec.)	*CPU time* (Sec.)	*CPU time* (Sec.)	*CPU time* (Sec.)
3	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
5	0.021	0.020	0.009	0.0116	0.001	0.001	0.001	0.001	0.001	0.001	0.001
6	0.780	0.7432	0.659	0.4245	0.0521	0.0522	0.0510	0.0517	0.0505	0.0504	0.0494
7	1.120	1.180	1.170	0.845	0.0869	0.0866	0.0849	0.0819	0.0798	0.0798	0.0748
8	1.241	1.239	1.210	1.0172	0.1387	0.1378	0.1376	0.1369	0.1361	0.1359	0.1345
9	1.711	1.699	1.698	1.4829	1.187	1.185	1.177	1.1751	1.1681	1.086	1.0781
10	5.499e+01	5.491e+01	5.450e+01	4.290e+01	3.131e+01	3.130e+01	3.127e+01	3.121e+01	3.109e+01	3.091e+01	3.039e+01
11	7.200e+02	7.110e+02	6.990e+02	5.710e+02	4.760e+02	4.740e+02	4.558e+02	4.552e+02	4.545e+02	4.521e+02	4.478e+02
12	8.602e+02	8.600e+02	7.981e+02	6.852e+02	5.651e+02	5.647e+02	5.430e+02	5.341e+02	5.338e+02	5.318e+02	5.227e+02
13	1.070e+03	1.030e+03	1.020e+03	1.000e+03	8.748e+02	8.749e+02	8.711e+02	8.631e+02	8.435e+02	8.341e+02	8.239e+02
14	1.028e+03	1.027e+03	1.021e+03	1.017e+03	1.014e+03	1.010e+03	1.010e+03	1.007e+03	0.911e+03	0.890e+03	0.841e+03
15	1.440e+03	1.480e+03	1.291e+03	1.259e+03	1.157e+03	1.158e+03	1.149e+03	1.134e+03	1.084e+03	1.078e+03	1.031e+03
16	1.680e+03	1.677e+03	1.450e+03	1.429e+03	1.339e+03	1.341e+03	1.332e+03	1.230e+03	1.154e+03	1.145e+03	1.100e+03
17	5.048e+04	5.040e+04	4.075e+04	4.050e+03	3.459e+03	3.457e+03	3.434e+03	3.412e+03	3.225e+03	3.211e+03	3.196e+03
18	6.840e+04	6.839e+04	5.897e+04	5.262e+04	4.072e+04	4.075e+04	4.067e+04	4.041e+04	3.866e+04	3.767e+04	3.645e+04
19	8.280e+04	8.280e+04	7.158e+04	7.118e+04	6.586e+04	6.585e+04	6.571e+04	6.542e+04	6.288e+04	5.378e+04	5.258e+04
20	1.12428e+05	1.1196e+05	1.0012e+05	8.356e+04	7.271e+04	7.270e+04	7.118e+04	7.032e+04	7.020e+04	6.974e+04	6.564e+04

Table 11: Statistical Analysis of Proposed BA and MBA Algorithms to find Near–OGRs for different channels in terms of mean and standard deviation

n	PROPOSED NATURE–INSPIRED ALGORITHMS											
	BA			BAM			LBA			LBAM		
	RL	*TBW* (Hz)	*CPU Time* (Sec.)	*RL*	*TBW* (Hz)	*CPU Time* (Sec.)	*RL*	*TBW* (Hz)	*CPU Time* (Sec.)	*RL*	*TBW* (Hz)	*CPU Time* (Sec.)
4	6.4 ±0.5	11.16 ±0.8	**0 ±0**	6.12 ±0.33166	11.16 ±0.8	**0 ±0**	6.08 ±0.27689	11.08 ±0.07667	**0 ±0**	6.08 ±0.27689	**11 ±0.0**	**0 ±0**
5	11.32 ±0.5568	24.48 ±0.8718	1.16e-02 ±1.4e-03	11.44 ±0.6506	24.4 ±0.8660	1e-03 ±1.8708e-04	11.36 ±0.4899	24.28 ±0.9798	1e-03 ±1.8484e-04	11.36 ±0.4899	24.28 ±0.9798	1e-03 ±1.6401e-04
6	17.55999 ±0.86987	43.28 ±0.93631	0.42439 ±0.01419	17.28 ±0.67823	43.64 ±0.75719	5.25e-02 ±1.87e-03	17.28 ±0.67823	43.64 ±0.75719	5.194e-02 ±7.08284e-04	17.32 ±0.47609	43.36 ±0.95219	5.068e-02 ±1.21665e-03
8	36.8 ±2.5331	114.76 ±2.0265	1.0172 ±1.8e-03	34.8 ±1.8708	116.36 ±1.4967	0.1387 ±2.4e-03	34.8 ±1.8708	116.36 ±1.4967	0.1378 ±2.0e-03	34.6 ±1.6583	116.52 ±1.3266	0.1376 ±5.1501e-04
12	**85 ±0.0**	**503 ±0.0**	6.852e+02 ±1.7321	**85 ±0.0**	**503 ±0.0**	5.651e+02 ±0.6547	85 ±0.0 (100%)	503 ±0.0 (100%)	5.647e+02±0.9	**85 ±0.0**	**503 ±0.0**	5.430e+02 ±0.3824
14	**127 ±0.0**	**924 ±0.0**	1.017e+03 ±1.6902	**127 ±0.0**	**924 ±0.0**	1.014e+03 ±1.6698	127 ±0.0	924 ±0.0 (100%)	1.010e+03 ±1.6513	**127 ±0.0**	**924 ±0.0**	1.010e+03 ±1.388
16	**177 ±0.0**	**1298 ±0.0**	1.429e+03 ±0.4397	**177 ±0.0**	**1298 ±0.0**	1.339e+03 ±1.1504	**177 ±0.0**	**1298 ±0.0**	1.341e+03 ±0.4397	**177 ±0.0**	**1298 ±0.0**	1.332e+03 ±0.2887
18	427 ±0.0	3079 ±0.0	5.262e+04 ±1.5133	445 ±0.0	2566 ±0.0	4.072e+04 ±1.2152	362 ±0.0	2912 ±0.0	4.075e+04 ±1.0693	**216 ±0.0**	**1894 ±0.0**	4.067e+04 ±1.0614
20	582.44 ±12.27151	4348.48 ±117.409	8.356e+04 ±1.0909	578 ±0.0	4306 ±0.0	7.271e+04 ±0.8	579.2 ±4.14331	4350.24 ±153.11877	7.270e+04 ±0.6	**283 ±0.0**	**2794 ±0.0**	7.118e+04 ±0.49329

Table 12: Statistical Analysis of Proposed CSA, CSAM, FPA, and FPAM Algorithms to find Near–OGRs for different channels in terms of mean and standard deviation

n	PROPOSED NATURE–INSPIRED ALGORITHMS											
	CSA			CSAM			FPA			FPAM		
	RL	*TBW* (Hz)	*CPU Time* (Sec.)	*RL*	*TBW* (Hz)	*CPU Time* (Sec.)	*RL*	*TBW* (Hz)	*CPU Time* (Sec.)	*RL*	*TBW* (Hz)	*CPU Time* (Sec.)
4	6.08 ±0.27689	**11 ±0.0**	**0 ±0**	6.04 ±0.2	**11 ±0.0**	**0.0 ±0.0**	**6.0 ±0.0**	**11 ±0.0**	**0.0 ±0.0**	**6.0 ±0.0**	**11 ±0.0**	**0.0 ±0.0**
5	11.04 ±0.2	24.92 ±0.4	1.0e-03±1e-04	11.04 ±0.2	24.92 ±0.4	1.0e-03 ±8e-05	**11.0 ±0.0**	**25 ±0.0**	1.0e-03 ±6e-05	**11.0 ±0.0**	**25 ±0.0**	**1.0e-03 ±0.0**
6	17.19 ±0.40825	43.72 ±1.06145	4.888e-02 ±3.59447e-03	17.08 ±0.27689	43.84 ±0.55377	4.849e-02 ±3.28417e-03	**17.04 ±0.2**	44.04 ±0.73485	4.846e-02 ±3.26982e-03	**17.04 ±0.2**	**43.84 ±0.5538**	**4.76e-02 ±2.5e-03**
8	34.8 ±1.8708	116.36 ±1.4967	0.1369 ±1.1e-03	34.4 ±1.3844	116.84 ±0.8	0.1361 ±8.27e-04	**34.2 ±1.0**	116.84 ±0.8	0.1359 ±4.7145e-04	**34.2 ±1.0**	**116.68 ±0.6511**	**0.1345 ±4.4878e-04**
12	**85 ±0.0**	**503 ±0.0**	5.341e+02 ±0.3211	**85 ±0.0**	**503 ±0.0**	5.338e+02 ±0.3161	**85 ±0.0**	**503 ±0.0**	5.318e+02 ±0.2002	**85 ±0.0**	**503 ±0.0**	**5.227e+02 ±0.1952**
14	**127 ±0.0**	**924 ±0.0**	1.0073e+03 ±1.036	**127 ±0.0**	**924 ±0.0**	0.911e+03 ±0.4397	**127 ±0.0**	**924 ±0.0**	0.890e+03±0.2457	**127 ±0.0**	**924 ±0.0**	**0.841e+03 ±0.2**
16	**177 ±0.0**	**1298 ±0.0**	1.230e+03 ±0.2769	**177 ±0.0**	**1298 ±0.0**	1.154e+03 ±0.2	**177 ±0.0**	**1298 ±0.0**	1.145e+03 ±0.2	**177 ±0.0**	**1298 ±0.0**	**1.100e+03 ±0.18**
18	**216 ±0.0**	**1894 ±0.0**	4.041e+04 ±0.9798	**216 ±0.0**	**1894 ±0.0**	3.866e+04 ±0.8	**216 ±0.0**	**1894 ±0.0**	3.767e+04 ±0.6	**216 ±0.0**	**1894 ±0.0**	**3.645e+04 ±0.2**
20	**283 ±0.0**	**2794 ±0.0**	7.032e+04 ±0.4	**283 ±0.0**	**2794 ±0.0**	7.020e+04 ±0.27689	**283 ±0.0**	**2794 ±0.0**	6.974e+04 ±0.23999	**283 ±0.0**	**2794 ±0.0**	**6.564e+04 ±0.2**

Table 13: Average rankings of all the proposed algorithms estimated by the Friedman's non-parametric test for OGRs problem

Algorithm	FRIEDMAN's NON–PARAMETRIC STATISTICAL TEST		
	Average Ranking		
	RL	*TBW* (Hz)	*CPU Time* (Sec.)
FPAM	**3.7500**	**3.7500**	**1.5556**
FPA	**3.7500**	3.9444	2.4167
CSAM	**3.7500**	4.0000	3.1944
CSA	4.1111	4.0278	4.1111
LBAM	4.1111	4.1389	4.8611
LBA	5.2778	5.0833	5.9722
MBA	5.5833	5.3889	6.2778
BA	5.6667	5.6667	7.6111
Friedman's statistic value	10.116	6.1884	89.1835
***p*-value**	<0.00001	<0.00001	<0.00001

Table 14: Unadjusted and adjusted *p*-values found for *RL*, *TBW*, and *CPU time* through Holm's post-hoc procedure with FPAM as control algorithm.

Algorithm	*RL*		*TBW* (Hz)		*CPU Time* (Sec.)	
	Unadjusted *p*–value	Adjusted *p*–value	Unadjusted *p*–value	Adjusted *p*–value	Unadjusted *p*–value	Adjusted *p*–value
FPA	**6.121536e–10**	**2.040512e–10**	**2.893099e–08**	**2.893099e–08**	**1.5707982e–47**	**1.5707982e–47**
CSAM	2.829387e–09	5.281522e–10	0.000003	5.708163e–07	2.930598e–37	1.052010e–37
CSA	7.224396e–07	9.194686e–08	0.000192	0.000028	1.099350e–34	2.462543e–35
LBAM	1.0	0.211576	1.0	0.298251	1.175332e–24	1.828294e–25
LBA	1.0	0.211576	1.0	0.473943	1.866960e–17	2.513215e–18
MBA	1.0	1.0	1.0	0.518484	6.125462e–09	8.575646e–10
BA	1.0	1.0	1.0	0.611701	0.002294	0.000642

The performance evaluation of proposed nature–inspired metaheuristic algorithms implies that the proposed algorithm FPAM is potentially more powerful and thus should be investigated further in many applications of industrial and engineering optimization problems.

6 Conclusions and Future Work

In this paper, WDM channel allocation algorithm by considering the concept of OGR sequence was presented. Finding either optimal or near–OGR sequences through conventional computing algorithms is computationally hard problem. The aim to use nature–inspired metaheuristic algorithms was not necessarily to produce perfect results, but to produce the near–to–optimal results under the given constraints. This paper presented the application of three nature–inspired algorithms (BA, CSA and FPA) and their modified forms (MBA, CSAM and FPAM) to solve near–OGRs problem. The proposed algorithms were validated and compared with other existing algorithms to find near–OGRs. It was observed that modified forms, enumerate near–OGRs very efficiently and more effectively than their simplified forms. Simulated results showed that the proposed algorithms are superior to the existing algorithms in terms of ruler length, total optical channel bandwidth, computational complexity and computation time. From preliminary results it was also concluded that algorithm MBA outperforms BA, CSAM outperforms both MBA and CSA, whereas FPAM slightly outperforms MBA, CSAM and FPA in terms of experimental computation time, computational complexity and maximum number of iterations needed to find optimal and near–optimal Golomb rulers. This implies that FPAM is potentially more superior to all other algorithms for solving such NP–complete problems in terms of both efficiency and success rate.

To date, the research done in [1, 3–7, 10–16, 30, 33] do not show the implementation of their algorithm in real WDM systems. Although numerous algorithms have been suggested for finding near–OGRs, there is no uniformly accepted formulation yet. So, in order for these algorithms to be of practical use, it is desired that the performance of these algorithms for higher order OGRs up to about several thousand channels is evaluated and used to provide unequal channel spacing in the real WDM system. Though this process will be very time consuming, this needs to be done for this work to be of some use in the field of communication engineering.

References

[1] Kwong W. C., Yang G. C., An algebraic approach to the unequal–spaced channel–allocation problem in WDM lightwave systems, IEEE Trans. Commun., 1997, 45(3), 352–359.
[2] Chraplyvy A. R., Limitations on lightwave communications imposed by optical–fiber nonlinearities, J. Lightwave Technol., 1990, 8, 1548–1557.
[3] Aggarwal G. P., Nonlinear fiber optics, 2nd ed., Academic Press, San Diego, CA, 2001.
[4] Thing V. L. L., Shum P., Rao M. K., Bandwidth–efficient WDM channel allocation for four–wave mixing–effect minimization, IEEE Trans. Commun., 2004, 52(2.12), 2184–2189.
[5] Saaid N. M., Nonlinear optical effects suppression methods in WDM systems with EDFAs: A review, Proceedings of International Conference on Computer and Communication Engineering (ICCCE–2010, May–2010, Kuala Lumpur, Malaysia), 2010.
[6] Forghieri F., R. Tkach W., Chraplyvy A. R., Marcuse D., Reduction of four–wave mixing crosstalk in WDM systems using unequally spaced channels, IEEE Photon. Technol. Lett., 1994, 6(6), 754-756.
[7] Babcock W., Intermodulation Interference in Radio Systems, Bell Syst. Tech. J., 1953, 63–73.
[8] Sugumaran S., Sharma N., Chitranshi S., Thakur N., Arulmozhivarman P., Effect of four–wave mixing on WDM system and its suppression using optimum algorithms, Int. J. Engineer. Technol. (IJET), 2013, 5(2), 1432–1444.
[9] Singh K., Bansal S., Suppression of FWM crosstalk on WDM systems using unequally spaced channel algorithms–A survey, Int. J. Advan. Res. Comput. Sci. Soft. Eng. (IJARCSSE), 2013, 3(12), 25–31.
[10] Sardesai H. P., A simple channel plan to reduce effects of nonlinearities in dense WDM systems, Lasers and Electro–Optics, (23–28 May 1999, CLEO '99 Baltimore, MD, USA), 1999, 183–184, DOI: 10.1109/CLEO.1999.834058.
[11] Forghieri F., Tkach R. W., Chraplyvy A. R., WDM systems with unequally spaced channels, J. Lightwave Technol., 1995, 13, 889–897.

[12] Hwang B., Tonguz O. K., A generalized suboptimum unequally spaced channel allocation technique—Part I: In IM/DDWDM systems, IEEE Trans. Commun., 1998, 46, 1027–1037.
[13] Tonguz O. K., Hwang B., A generalized suboptimum unequally spaced channel allocation technique—Part II: In coherent WDM systems," IEEE Trans. Commun., 1998, 46, 1186–1193.
[14] Atkinson M. D., Santoro N., Urrutia J., Integer sets with distinct sums and differences and carrier frequency assignments for nonlinear repeaters, IEEE Trans. Commun., 1986, 34(6), 614–617.
[15] Randhawa R., Sohal J. S., Kaler R. S., Optimum algorithm for WDM channel allocation for reducing four–wave mixing effects, Optik 120, 2009, 898–904.
[16] http://www.compunity.org/events/pastevents/ewomp2004/jaillet_krajecki_pap_ew04.pdf.
[17] Bloom G. S., Golomb S. W., Applications of numbered undirected graphs, Proc. IEEE, 1977, 65(4), 562–570.
[18] Thing V. L. L., Rao M. K., Shum P., Fractional optimal Golomb ruler based WDM channel allocation, Proceedings of The 8th Opto-Electronics and Communication Conference (OECC–2003, October–2003), 2003, 23, 631–632.
[19] Shearer J. B., Some new disjoint Golomb rulers, IEEE Trans. Inf. Theory, 1998, 44(7), 3151–3153.
[20] http://theinf1.informatik.uni-jena.de/teaching/ss10/oberseminar-ss10.
[21] Robinson J. P., Optimum Golomb rulers, IEEE Trans. Comput., 1979, 28(12), 183–184.
[22] Shearer J. B., Some new optimum Golomb rulers, IEEE Trans Inf. Theory, 1990, 36, 183–184.
[23] Galinier P., Jaumard B., Morales R., Pesant G., A constraint–based approach to the Golomb ruler problem, Proceedings of 3rd International Workshop on Integration of AI and OR Techniques (CP–AI–OR 2001), 2001.
[24] Leitao T., Evolving the maximum segment length of a Golomb ruler, Proceedings of Genetic and Evolutionary Computation Conference (June–2004, USA), 2004.
[25] Rankin W. T., Optimal Golomb rulers: An exhaustive parallel search implementation, M.S. Thesis, Duke University, 1993, http://people.ee.duke.edu/~wrankin/golomb/golomb.html.
[26] Cotta C., Dotú I., Fernández A. J., Hentenryck P. V., A memetic approach to Golomb rulers, parallel problem solving from nature–PPSN IX, Lecture Notes in Computer Science, Springer–Verlag Berlin Heidelberg, 2006, 4193, 252–261.
[27] Soliday S. W., Homaifar A., Lebby G. L., Genetic algorithm approach to the search for Golomb rulers, Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA–95, Morgan Kaufmann), 1995, 528–535.
[28] Robinson J. P., Genetic search for Golomb arrays, IEEE Trans Inf. Theory, 2000, 46(3), 1170–1173.
[29] Ayari N., Luong T. V., Jemai A., A hybrid genetic algorithm for Golomb ruler problem, Proceedings of ACS/IEEE International Conference on Computer Systems and Applications (AICCSA–2010, May 16–19, 2010), 2010, 1–4.
[30] Bansal S., Optimal Golomb ruler sequence generation for FWM crosstalk elimination: Soft computing versus conventional approaches", Appl. Soft Comput., 2014, 22, 443–457.
[31] Bansal S., Kumar S., Sharma H., P. Bhalla, Golomb ruler sequences optimization: A BBO approach, Int. J. Comput. Sci. Inf. Secur. (IJCSIS), 2011, 9(5), 63–71.
[32] Bansal S., Golomb ruler sequences optimization: Soft computing approaches, M.Tech. Thesis, Maharishi Markandeshwar Engineering College, Deemed University, Mullana, India, 2011.
[33] Kumar S., Bansal S., Bhalla P., Optimal Golomb ruler sequence generation for FWM crosstalk elimination: A BB–BC approach, Proceedings of 6th International Multi Conference on Intelligent Systems, Sustainable, New and Renewable Energy Technology and Nanotechnology (IISN–2012, March 16–18, 2012, Institute of Science and Technology Klawad, Haryana), India, 2012, 255–262.
[34] Bansal S., Kumar S., Bhalla P., A novel approach to WDM channel allocation: Big bang–big crunch optimization, Proceedings of Zonal Seminar on Emerging Trends in Embedded System Technologies (ETECH–2013, The Institution of Electronics and Telecommunication Engineers (IETE), Chandigarh Centre, Chandigarh), India, 2013, 80–81.
[35] Bali S., Bansal S., Kamboj A., A novel hybrid multi–objective BB–BC based channel allocation algorithm to reduce FWM crosstalk and its comparative study, Int. J. Comput. Appl. (IJCA), 2015, 124(12), 38–45.
[36] Vyas J., Bansal S., Sharma K., Generation of optimal Golomb rulers for FWM crosstalk reduction: BB–BC and FA approaches, Proceedings of 2016 International Conference on Signal Processing and Communication (ICSC–2016, December 26–28, 2016), Jaypee Institute of Information Technology, Noida, India, 2016, 74–78.
[37] Bansal S., Singh K., A novel soft–computing algorithm for channel allocation in WDM systems, Int. J. Comput. Appl. (IJCA), 2014, 85(9), 19–26.
[38] Bansal S., Jain P., Singh A. K., Gupta N., Improved multi–objective firefly algorithms to find OGR sequences for WDM channel–allocation, World Academy of Science, Engineering and Technology, International Science Index, Int. J. Math., Comput., Phy., Elect. Comput. Eng., 2016, 10(7), 315–322.
[39] Dotú I., Hentenryck P. V., A simple hybrid evolutionary algorithm for finding Golomb rulers, Proceedings of Evolutionary Computation, (2–5 September, 2005, The 2005 IEEE Congress on), 2005, 3, 2018–2023, DOI: 10.1109/CEC.2005.1554943.
[40] Colannino J., Circular and modular Golomb rulers, 2003. http://cgm.cs.mcgill.ca/~athens/cs507/Projects/2003/JustinColannino/.
[41] Dimitromanolakis A., Analysis of the Golomb ruler and the sidon set problems, and determination of large, near–optimal Golomb rulers, Master's Thesis, Technical University of Crete, 2002.
[42] Dollas A., Rankin W. T., McCracken D., A new algorithm for Golomb ruler derivation and proof of the 19 mark ruler, IEEE Trans. Inf. Theory, 1998, 44(1), 379–382.

[43] Distributed.net, Project OGR. http://www.distributed.net/ogr.
[44] Cotta C., Dotú I., Fernandez A. J., Hentenryck P. V., Local search–based hybrid algorithms for finding Golomb rulers, Kluwer Academic Publishers, Boston, 2007, 12(3), 263–291.
[45] Drakakis K., S. Rickard, On the construction of nearly optimal Golomb rulers by unwrapping costas arrays, Contemp. Eng. Sci., 2010, 3(7), 295–309.
[46] Drakakis K., A review of the available construction methods for Golomb rulers, Adv. in Math. Commun., 2009, 3(3), 235–250.
[47] Caicedo Y., Martos C. A., Trujillo C. A., g-Golomb rulers, Revista Integración Temas Mat., 2015, 33(2), 161–172, DOI: http://dx.doi.org/10.18273/revint.v33n2-2015006.
[48] http://mathworld.wolfram.com/PerfectRuler.html.
[49] http://mathworld.wolfram.com/GolombRuler.html.
[50] Lam A. W., Sarwate D. V., On optimal time–hopping patterns, IEEE Trans. Commun., 1988, 36, 380–382.
[51] Lavoie P., Haccoun D., Savaria Y., New VLSI architectures for fast soft-decision threshold decoders, IEEE Trans. Commun., 1991, 39(2), 200–207.
[52] Robinson J. P., Bernstein A. J., A class of binary recurrent codes with limited error propagation, IEEE Trans. Inf. Theory,1967, IT–13, 106–113.
[53] Cotta C., Fernández A. J., Analyzing fitness landscapes for the optimal Golomb ruler problem, Evolutionary Computation in Combinatorial Optimization, (Eds. J. Gottlieb,G. Raidl), Lecture Notes in Computer Science, 2005, Springer–Verlag Berlin, 3448, 68–79.
[54] Fang R. J. F., Sandrin W. A., Carrier frequency assignment for non–linear repeaters, Comsat Tech.l Rev., 1977, 7, 227–245.
[55] Blum E. J., Biraud F., Ribes J. C., On optimal synthetic linear arrays with applications to radio astronomy, IEEE Trans. Antennas Propag., 1974, 22, 108–109.
[56] Memarsadegh N., Golomb patterns: Introduction, applications, and citizen science game, Information Science and Technology (IS&T), Seminar Series NASA GSFC, 11September, 2013. http://istcolloq.gsfc.nasa.gov/fall2013/presentations/memarsadeghi.pdf.
[57] Shearer J. B., Golomb ruler table, Mathematics Department, IBM Research, http://www.research.ibm.com/people/s/shearer/grtab.html.
[58] Shearer J. B., Smallest known Golomb rulers, Mathematics Department, IBM Research, http://www.research.ibm.com/people/s/shearer/gropt.html.
[59] Dewdney A., Computer recreations, Scientific American, 1985, 16–26.
[60] Dewdney A., Computer recreations, Scientific American, 1986, 14–21.
[61] Cotta C., Hemert J. V., Recent advances in evolutionary computation for combinatorial optimization, Studies in Computational Intelligence, Springer, 153.
[62] Yang X. S., Optimization and metaheuristic algorithms in engineering, in: Metaheursitics in Water, Geotechnical and Transport Engineering (Eds. X. S. Yang, A. H. Gandomi, S. Talatahari, A. H. Alavi), Elsevier, 2013, 1–23, http://dx.doi.org/10.1016/B978-0-12-398296-4.00001-5.
[63] Yang X. S., Nature–inspired metaheuristic algorithms, 2nd Edition, Luniver Press, 2010.
[64] Koziel S., Yang X. S., Computational optimization, methods and algorithms, Studies in Computational Intelligence, Springer, 2011, 356.
[65] Yang X. S., Nature–inspired mateheuristic algorithms: success and new challenges, J. Comput. Eng. Inf. Tech. (JCEIT), 2012, 1(1), 1–3, doi:104172/2324-9307.1000e101.
[66] Rajasekaran S., and Pai G. A. V., Neural networks, fuzzy logic, and genetic algorithms–synthesis and applications, Prentice Hall of India Pvt. Ltd., New Delhi, 2004.
[67] Mitchell M., An introduction to genetic algorithms, Prentice Hall of India Pvt. Ltd., New Delhi, 2004.
[68] Goldberg D. E., Genetic algorithms in search, optimization, and machine learning, Addison Wesley, USA, 1989.
[69] Storn R., Price K. V., Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces, J. Glob. Optim., 1997, 11(4), 341–359.
[70] Price K., Storn R., Lampinen J., Differential evolution–A practical approach to global optimization, Springer, Berlin, Germany, 2005.
[71] Yang X. S., Flower pollination algorithm for global optimization, in: Unconventional Computation and Natural Computation 2012, Lecture Notes in Computer Science, Springer, Berlin, 2012, 7445, 240–249.
[72] Yang X. S., Review of metaheuristics and generalized evolutionary walk algorithm, Int. J. Bio–Inspir. Comput., 2011, 3(2), 77–84.
[73] Yang X. S., A new metaheuristic bat–inspired algorithm, in: Nature Inspired Coop-erative Strategies for Optimization (NISCO–2010) (Eds. J. R. Gonzalez et al.), Studies in Computational Intelligence, Springer Berlin, 2010, 284, 65–74.
[74] Yang X. S., Bat algorithm for multi–objective optimization, Int. J. Bio–Inspir. Comput., 2011, 3(5), 267–274.
[75] Yang X. S., Bat algorithm: literature review and applications, Int. J. Bio–Inspir. Comput., 2013, 5(3), 141–149, DOI: 10.1504/IJBIC.2013.055093.
[76] Yang X. S., Gandomi A. H., Bat algorithm: A novel approach for global engineering optimization, Eng. Comput., 2012, 29(5), 464–483.
[77] Yang X. S., Deb S., Engineering optimisation by cuckoo search, Int. J. Math. Model. Numer. Optim., 2010, 1(4), 330–343.

[78] Yang X. S., Deb S., Cuckoo search via levy flights, in: Proc. of World Congress on Nature & Biologically Inspired Computing (NABIC–2009), IEEE Publications, USA, 2009, 210-214.
[79] Gandomi A. H., Yang X. S., Alavi A. H., Cuckoo search algorithm: A metaheuristic approach to solve structural optimization problems, Eng. with Comput. An Int. J. Sim. Based Eng., Springer Verlang, London, 2013, 29(1), 17–35, DOI: 10.1007/s00366-011-0241-y.
[80] Yang X. S., Deb S., Cuckoo search: recent advances and applications, Neural Computing and Applications, 2014, 24(1), 169–174.
[81] Yang X. S., Karamanoglu M., He.X. S., Multi–objective flower algorithm for optimization, Proceedings of International Conference on Computational Science (ICCS–2013, Procedia Computer Science), 2013, 18, 861–868.
[82] Yang X. S., Karamanoglu M., He. X. S., Flower pollination algorithm: A novel approach for multiobjective optimization, Eng. Optim., 2014, 46(9), 1222–1237, DOI: 10.1080/0305215X.2013.832237.
[83] Pratap R., Getting started with Matlab a quick introduction for scientists and engineers, Oxford University Press, New York, 2010.
[84] http://www.myreaders.info/html/artificial_intelligence.html.
[85] Derrac J., García S., Molina D., Herrera F., A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, Swarm and Evolut. Comput., 2011, 1, 3–18.
[86] Holm S., A simple sequentially rejective multiple test procedure, Scand. J. Stat., 1979, 6, 65–70.