

# Nature-Inspired Algorithms for Solving Weighted Constraint Satisfaction Problems

Mehdi Bidar and Malek Mouhoub<sup>1</sup>

*Department of Computer Science, University of Regina, Regina, Canada*

**Keywords:** Constraint Satisfaction Problems (CSPs), Nature-Inspired Techniques, Soft Constraints, Combinatorial Optimization.

**Abstract:** Several applications such as timetabling, scheduling and resource allocation, can be represented as a Constraint Satisfaction Problem (CSP). Solving a CSP consists in finding a complete assignment of values to variables satisfying all the constraints. In many real-life scenarios (including over-constrained problems), some constraints (called soft constraints) can be violated according to some penalty function. In this regard, the Weighted CSP (WCSP) can be used as an extension of the CSP where each constraint comes with a cost function. Solving a WCSP consists in finding an optimal solution minimizing the total costs related to all constraints. Searching for an optimal solution to a WCSP is usually dealt with classical complete methods like backtracking and bucket elimination techniques. However, since WCSPs are NP-hard, complete methods will require exponential time cost. Therefore, approximation methods such as metaheuristics are appropriate alternatives as they are capable of providing a good compromise between the quality of the solution and the corresponding running time. We study the applicability of several nature-inspired techniques including; Particle Swarm Optimization (PSO), Firefly, Genetic Algorithms (GAs), Artificial Bee Colony (ABC), Mushroom Reproduction Optimization (MRO), Harmony Search (HS) and Focus Group (FG). While these methods do not guarantee the optimality of the solution returned, they are in general successful in returning a good solution in a desirable time cost. This statement has been demonstrated through the experimental results we conducted on randomly generated WCSP instances following the known RB model. The latter has been adopted as it has the ability to produce hard-to-solve random problem instances. The obtained results are promising and show the potential of the considered nature-inspired techniques.


## 1 INTRODUCTION

A CSP consists of a set of variables, each defined over a discrete and finite set of values, and a set of constraints restricting the values that the variables can simultaneously take (Dechter and Cohen, 2003; Kumar, 1992; Tsang, 2014). Solving a CSP consists in finding a complete assignment of values to variables such that all the constraints are satisfied. A CSP can be tackled using a systematic search technique, i.e. backtracking. Due to the exponential time cost of the backtrack search algorithm, constraint propagation can be used to improve this running time in practice. Another alternative is to use metaheuristics as the latter are capable of trading running time for the quality of the returned solution (Talbi, 2009; Glover and Kochenberger, 2006; Blum and Roli, 2003; Bidar et al., 2020b; Korani and Mouhoub, 2021; Abbasian

and Mouhoub, 2016).

For many real-world applications, we might need to look for a good or a best solution satisfying all the constraints while optimizing some objectives. This has motivated the research community to develop extensions of CSPs taking into account penalties and costs. The Weighted Constraint Satisfaction Problems (WCSP) (Cooper et al., 2010; Lee and Leung, 2009) is an example of such extended models. In a WCSP, we consider two types of constraints: soft constraint that can be violated with associated costs and hard constraints that must be satisfied. Solving a WCSP is about finding a solution that satisfies all the hard constraints and minimizes the total cost related to soft constraints.

Like for CSPs, WCSPs can also be tackled by systematic or approximation methods (Gallardo et al., 2009; Bidar and Mouhoub, 2019b). In the case of systematic search techniques, the most known ones

<sup>1</sup>  <https://orcid.org/0000-0001-7381-1064>

are a variant of backtracking, i.e. branch and bound (Lawler and Wood, 1966), and bucket elimination (Dechter and Cohen, 2003; Gallardo et al., 2009). As for CSPs, systematic search methods come with an exponential cost when solving WCSPs. To overcome this difficulty in practice, some heuristic search techniques such as beam search and mini-buckets have been introduced and added respectively to systematic search methods (Dechter and Cohen, 2003; Gallardo et al., 2009). Despite the success of these heuristics, there is still a challenge especially when the problem size scales up. Like for CSPs, we can rely on metaheuristics as an alternative to provide at least near to optimal solutions to WCSP instances within a reasonable time frame (Bidar et al., 2018a; Bidar and Mouhoub, 2019a; Talbi, 2009). This has motivated us to study the applicability of nature-inspired techniques for solving WCSPs. More precisely, we consider the following methods: Particle Swarm Optimization (PSO) (Kennedy and Eberhart, 1995; Bidar and Mouhoub, 2019a), Firefly Algorithm (FA) (Wang et al., 2017; Yang, 2008; Bidar and Mouhoub, 2019b), Genetic Algorithms (GAs) (Abbasian and Mouhoub, 2016), Artificial Bee Colony algorithm (ABC) (Karaboga, 2005a), Mushroom Reproduction Optimization (MRO) (Bidar et al., 2018b; Bidar et al., 2018a), Harmony Search (HS) (Geem et al., 2001) and Focus Group (FG) (Fattahi et al., 2018; Bidar et al., 2020a). The main reason to choosing these algorithms is the fact that some have already been adapted to successfully solve CSPs (Bidar et al., 2020a; Bidar and Mouhoub, 2019b; Bidar and Mouhoub, 2019a). In addition, we aim for a variety of algorithms in terms of their respective source of inspiration: swarm intelligence and social behavior (PSO and ABC for foraging behavior and FA for other behavior), biological evolution (GAs), organism structure (MRO), and physical/chemical process (HS). Finally, these algorithms have been successfully applied to a wide range of optimization problems including feature selection, antenna design optimization, scheduling and planning, classification and clustering, resource allocation and vehicle routing. In this regard, we propose a general methodology that allow a discretization of nature-inspired techniques. We then show the adaptation of each of the algorithm we consider. Finally, we report on the experiments we have conducted to evaluate the different methods on randomly generated WCSP instances. The latter have been produced using a new variant of the RB model (Xu and Li, 2000) that we have designed. The rationale for adopting the RB model is due to its ability to randomly produce hard-to-solve problem instances. Given the the Firefly Algorithm (FA) has already been

used to solve WCSPs (Bidar and Mouhoub, 2019b), we will only consider this method in the comparative experimentation section. Similarly, we use the same CSP GA solving approach reported in (Abbasian and Mouhoub, 2016) with the fitness function corresponding to the total weight to minimize, rather than the number of violated constraints.

## 2 PRELIMINARIES

A CSP is a tuple  $(X, D, C)$  where  $X$  is a set of variables, such that each variable  $X_i \in X$  is defined on a finite domain of values  $D_i \in D$ , and  $C$  is a set of constraints restricting the values that the variables can simultaneously take. Solving a CSP consists in finding a complete assignment of values to all the variables,  $S = \{X_1 = v_1, X_2 = v_2, \dots, X_n = v_n\}$ , such that all the constraints hold (Dechter and Cohen, 2003).

A WCSP is an extension of a CSP to deal with soft constraints. More formally, a WCSP is defined as a tuple  $t = (X, D, C, K)$  where the constraints in  $C$  are soft constraints and  $K$  is a positive integer. Each soft constraint  $c \in C$  corresponds to a cost function, over a set of variables (its scope), returning a value in  $\{0, \dots, K\}$ , for each instantiation (assignment of values to the variables in its scope). In particular,  $c$  returns 0 when its instantiation is completely satisfactory, and returns  $K$  when the instantiation is forbidden. Any other value between 0 and  $K$  corresponds to a degree of preference. Solving a WCSP consists of finding a complete assignment minimizing the total sum of all the costs related to all the constraints. In this regard, a WCSP can be seen as an discrete combinatorial problem and is identified as NP-hard (Haddouch et al., 2016; Cooper et al., 2010).

A binary WCSP is a special case of WCSPs where the constraints involve at most 2 variables. Here, constraints can be unary or binary.

- Unary constraints (cost functions) are defined as follows.  $C_i : a \in D_i \rightarrow \{0, \dots, K\}$  where  $D_i$  is the domain of variable  $X_i$ .
- Binary cost functions are defined as follows.  $C_{ij} : (a, b) \rightarrow \{0, \dots, K\}$ , where  $a \in D_i$  and  $b \in D_j$ ,  $D_i$  and  $D_j$  are respectively the domains of variables  $X_i$  and  $X_j$ .

The total cost of a complete assignment  $S$  will then be equal to the cost of  $\mathcal{T}(S)$  defined as follows.

$$\mathcal{T}(S) = \sum_{C_{ij} \in C, \{X_i, X_j\} \subset X} C_{ij} \oplus \sum_{C_i \in C, \{X_i\} \subset X} C_i \quad (1)$$

where  $\oplus$  is an operator defined through the following equation.

$$a \oplus b = \min\{K, a + b\} \quad (2)$$

An inconsistent assignment  $S$  corresponds to a total cost,  $\mathcal{T}(S) = K$ . A solution with the minimum total cost is the optimal solution (most preferred one). Therefore, solving a WCSP consists in finding the solution that minimizes the total cost, and is formulated as follows.

$$\text{Minimize}(\mathcal{T}(S)) \quad (3)$$

Here,  $S$  belongs to the set of potential solutions.

### Example 1

Let us consider a WCSP with four variables,  $A$ ,  $B$ ,  $C$ , and  $D$ , defined on the same domain  $D = \{a, b, c, d\}$ . The constraints (excluding those with cost 0) are listed as follows.

**Hard Constraints:** Each of these constraints is defined as a set of forbidden tuples, each corresponding to a cost function with a value equal to  $K = 10$ . They are listed as follows.

$$(A, B) \leftarrow (b, b), (b, c), (d, d), (c, c), (c, d)$$

$$(B, C) \leftarrow (a, b), (b, c), (c, c), (d, d), (d, c)$$

$$(C, D) \leftarrow (a, a), (a, b), (b, b), (b, d), (d, d), (d, b)$$

**Soft Constraints:** These correspond to all the other unary and binary soft constraints.

$$\text{Unary Constraints: } C_i \leftarrow C_a = 1, C_b = 3, C_c = 2, C_d = 4$$

**Binary Constraints:**

$$C_{AB} \leftarrow C(a, a) = 3, C(a, b) = 1, C(b, d) = 1, C(d, c) = 5$$

$$C_{BC} \leftarrow C(a, a) = 2, C(a, c) = 1, C(b, d) = 7, C(c, d) = 3$$

$$C_{CD} \leftarrow C(a, c) = 1, C(c, c) = 9, C(c, d) = 6$$

Figure 1 shows the WCSP with all the constraints listed in Example 1. Here, hard constraints are displayed in dash lines.

The following are potential solutions with their total cost functions.

$$1. S_1 = [a b d c], \text{ Cost} = 1 + 7 + 0 + 1 + 3 + 4 + 2 = 18$$

$$2. S_2 = [b d b a], \text{ Cost} = 1 + 0 + 0 + 3 + 4 + 3 + 1 = 12$$

One optimal solution, minimizing the total cost, is presented as follows.

$$1. ch = [a a a c], \text{ Cost} = 3 + 2 + 0 + 1 + 1 + 1 + 2 = 10$$

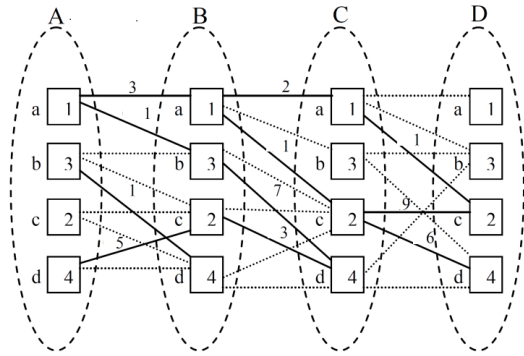


Figure 1: The WCSP of Example 1.

## 3 RELATED WORK

In (Gallardo et al., 2009), Jose et al. proposed a new method for solving WCSPs, using memetic algorithms with bucket elimination. The authors also investigated the combination of memetic algorithms with other exact methods like branch-and-bound and mini-buckets techniques. Based on the experiments they conducted, the authors claimed that their method outperforms the other existing approaches in terms of running time. Javier proposed a new approach to maintain local consistency while solving a WCSP (Larrosa and Schiex, 2003). Javier's approach was developed considering two main variants of the arc consistency algorithm: Directional Arc Consistency (DAC) and Full Directional Arc Consistency (FDAC). These two algorithms have been introduced by Cooper for binary WCSPs. Javier's proposed approach was able to maintain either DAC or FDAC while searching for a solution for a given WCSP.

In (Delisle and Bacchus, 2013), the authors presented a new approach for solving WCSPs. The WCSP is first converted into an ordinary CSP by converting all soft constraints to hard constraints. Then, the algorithm performs some constraint relaxations on the problem and checks if the latter has a solution or not. If there is no solution for the relaxed version of the problem, the algorithm will compute a set of forbidden tuples to rule out all the generated assignments (solutions). From this set of tuples, the algorithm forms a new relaxation and then checks again if it has a solution. The main goal here is to find the optimal relaxation which will result in incurring a minimal cost. Therefore, any solution meeting the requirements of this relaxation will be a solution to the original WCSP. In (Larrosa and Schiex, 2003), the authors introduced two new arc consistency enforcing algorithms based on AC-2001/3.1 with improved time

complexity  $O(e \times d^3)$  ( $e$  is the number of constraints and  $d$  is the domain size) in comparison with the arc consistency enforcing method which was proposed by Schiex in (Schiex, 2000). The latter method has a complexity  $O(e^2 \times d^4)$  for solving WCSPs. Then, Javier introduced a stronger alternative for arc consistency called AC\* along with its enforcing algorithm which has the complexity of  $O(n^2 \times d^2 + e \times d^3)$  where  $n$  is the number of variables.

In (Lau, 2002), Lau proposed a new approach for solving WCSPs based on semi-definite programming. The main goal is to take the respective advantage of complete and incomplete methods (guaranteeing the optimal solution and faster running time). The main feature of the proposed algorithm is its ability to find a solution that has provable worst case bound in terms of weight, comparing to the optimal solution. This is in contrast with conventional incomplete methods which have acceptable performance in practice but do not guarantee to perform well in worst case situations.

## 4 DISCRETE REPRESENTATION FOR WCSPs

In order to apply these techniques and the other methods we are considering to solve WCSPs, we propose a unified approach for the discretization of any nature-inspired method. Our discretization will include the representation/definition of the potential solution, fitness function and distance between two potential solutions.

### 4.1 Potential Solution and Fitness Function

As done with nature-inspired techniques for CSPs, we represent a WCSP potential solution with a chromosome, where each entry corresponds to the value assigned to each variable. Table 1 shows a chromosome corresponding to a potential solution of the WCSP presented in Example 1.

Table 1: A chromosome representing a potential solution.

Variables	$X_1$	$X_2$	$X_3$	$X_4$
Chromosome	b	d	b	a

The fitness function of a chromosome (potential solution) corresponds to the total cost function  $\mathcal{T}(S)$ . For instance, the fitness function of the potential solution depicted in Table 1 is equal to 12, as described in Example 2.

### 4.2 Similarity and Distance Between Potential Solutions

One important parameter needed by nature-inspired techniques, especially when conducting exploitation, is the similarity between two potential solutions. In this regard, we employ the Hamming distance which corresponds to the number of values that both solutions do not share. The Hamming distance between two solution  $S_i$  and  $S_j$  with  $n$  variables,  $d_H(S_i, S_j)$ , is calculated, according to the fitness function, as follows.

$$d_H(S_i, S_j) = \sum_{k=1}^n (S_{i,k} \neq S_{j,k}) \quad (4)$$

In the above equation,  $S_{i,k}$  and  $S_{j,k}$  represent the values assigned to the  $k^{th}$  variable of  $S_i$  and  $S_j$ , respectively. Basically, the Hamming distance corresponds to the number of values that the two solutions do not have in common. In this regard, the Hamming distance between two identical solutions is equal to 0, while the Hamming distance between two solutions not having any value in common is equal to the number of variables,  $n$ . Table 2 shows the computation of the Hamming distance for two potential solutions.

Table 2: The Hamming distance between two solutions.

Variables	$X_1$	$X_2$	$X_3$	$X_4$
$S_i$	a	b	d	c
$S_j$	a	c	d	a
$\neq$ values		X		X
$d_H$	=	1	+	1

As we will show in the next section, the Hamming distance will be used to implement one of the operators required by the nature-inspired techniques we are considering.

## 5 NATURE-INSPIRED TECHNIQUES FOR WCSPs

### 5.1 Particle Swarm Optimization (PSO)

Particle swarm optimization (PSO) algorithm is one of the most popular nature-inspired swarm-based optimization algorithms, developed based on the collective intelligent behavior of systems like fish schooling and birds flocking (Kennedy and Eberhart, 1995). PSO has been proven to be very effective in dealing with a wide range of combinatorial optimization problems including continuous and discrete problems (Poli et al., 2007; Bidar and Mouhoub, 2019a; Korani



and Mouhoub, 2020). In PSO, every particle position is influenced by its own best position,  $pbest$ , as well as the best position made by other particles so far,  $gbest$ . In (Bidar and Mouhoub, 2019a) a discrete version of PSO (called Discrete PSO, or DPSO) has been used to solve Dynamic CSPs (DCSPs). This specific problem consists in solving CSPs when constraints are added dynamically. We use a similar version of DPSO in this paper with a new definition for the fitness function. While this function has been defined as the number of violated constraints in (Bidar and Mouhoub, 2019a), we define it as the total cost function as described in the previous section. Otherwise, we follow the same DPSO algorithm, including the following equations defined respectively to update the velocity and position of the different particles corresponding to potential solutions (Bidar and Mouhoub, 2019a).

$$V_i^{t+1} = \underbrace{w \otimes V_i^t}_{\text{exploration}} \oplus \underbrace{c_1 r_1 \otimes (pbest_i^t \ominus X_i^t) \oplus c_2 r_2 \otimes (gbest_i^t \ominus X_i^t)}_{\text{exploitation}} \quad (5)$$

$$X_i^{t+1} = X_i^t \oplus V_i^{t+1}, \quad i = 1, \dots, n \quad (6)$$

In DPSO (Bidar and Mouhoub, 2019a), exploitation and exploration strategies (Črepinšek et al., 2013) are controlled through the above two equations, using three controlling parameters,  $w$ ,  $c_1$  and  $c_2$ . A high value of  $w$  encourages more exploration, while a low value favors exploitation (Nickabadi et al., 2011). Similarly, low values of  $c_1$  and  $c_2$  allow for more exploration for promising areas, while high values for these parameters favor more local intensification.

When using the operator  $\otimes$ ,  $w$  determines the percentage of the variable values that will be passed from  $V_i^t$  to  $V_i^{t+1}$ . The rest of  $V_i^{t+1}$  values will be generated randomly. Similarly,  $c_1 r_1$  and  $c_2 r_2$  correspond to the percentage of values to keep from  $(pbest_i^t \ominus X_i^t)$  and  $(gbest_i^t \ominus X_i^t)$ , respectively. This operation is again enforced through  $\otimes$ . The operator  $\oplus$  allows the selection of the values that are in  $pbest_i^t$  (respectively  $gbest_i^t$ ) and not in  $X_i^t$  ( $\ominus$  can be seen as a set difference operation).

## 5.2 Discrete MRO (DMRO)

The initial version of MRO (Bidar et al., 2020b) tackles continuous optimization problems ( $X \in R^n$ ). To apply a discrete version of MRO to solve WCSPs, we have used the definitions stated in the previous section. MRO is biologically inspired by the natural reproduction and growth mechanisms of mushrooms.

A potential solution corresponds to either a parent mushroom or a spore, and a population is a mix of both. In the case of WCSPs, this population is initially a set of randomly generated solutions. The search process will then follow an exploration/exploitation process and will be guided by the following equations respectively corresponding to local and global movements. More precisely, a local movement corresponds to a local search with the goal to make local improvements to potential solutions. This is done by spreading spores by parent mushrooms in their respective colonies. Equation 7 reflects this local movement. This equation updates the new location of spore  $j$  of colony (parent mushroom)  $i$  ( $X_{i,j}$ ).  $X_i^{parent}$  is the location of the parent  $i$  and  $Rand(0,1)$  generates a random number in  $(0,1)$ .

$$X_{i,j} = X_i^{parent} + Rand(0,1) \quad (7)$$

The global movement follows the natural phenomena of a wind factor. If the wind is blowing, spores are moved to different parts of the problem space and land in new locations. Next, spores grow and become mature mushrooms. Equations (8) and (9) define the global movement of the spores induced by wind.

$$X_{i,j} = X_i^{parent} + Move_j^{wind} \quad (8)$$

$$Move_j^{wind} = (X_i^* - X_k^*) \times \left( \frac{Ave(i)}{T_{ave}} \right)^m \times Rand(-\delta, \delta) \times rs + Rand(-r, r) \quad (9)$$

Here,  $X_i^*$  and  $X_k^*$  are the parent solutions of the colonies  $i$  and  $k$ ,  $Ave(i)$  is the average of solutions quality (fitness function) of colony  $i$ ,  $T_{ave}$  is the total average of all colonies,  $Rand(-\delta, \delta)$  is a vector that determines direction movement of the wind,  $rs \in (0,1)$  is the size of random step and  $Rand(-r, r)$  is the random movement of the spores to their neighboring areas with radius  $r$ .

## 5.3 Discrete Focus Group Optimization Algorithm (DFGOA)

FGOA is inspired by the collaborative behavior of a group's members sharing their ideas on a given subject (Fattahi et al., 2018). This algorithm has been initially proposed for continuous optimization problems. A discrete version of FGOA, called Discrete FGOA (DFGOA), has been proposed to deal with CSPs (Bidar et al., 2020b). In DFGOA, the search process is guided by the impact factor parameter for each potential solution based on its quality as shown in equation 10.

$$IF^{t+1}(i) = IF^t(i) + \sum_{j=1}^{nPop} \left( \frac{rand(l) \times (|F(S(i)) - F(S(j))|) \times IC(j)}{Nvar} \right)^m \quad (10)$$

Here,  $IF(i)$  is the impact factor of participant  $i$  which will take an important role in the next steps to affect the other participants' solutions,  $IF^{t+1}(i)$  is the new impact factor of participant  $i$ ,  $nPop$  is the population size,  $Nvar$  is the number of variables of the problem,  $rand(l)$  generates a random number in (0,1) and  $F(S_i)$  and  $F(S_j)$  are the qualities of solutions  $i$  and  $j$  respectively.  $IC(j)$ , the impact coefficient, is a random number in (0,1) and is assigned to each solution. In this regard, a set of  $nPop$  random numbers is generated and based on the qualities of solutions are assigned to each solution (the more quality a solution is, the larger the value will be assigned to). In a discrete problem space like for CSPs and WCSPs, affecting a solution can be interpreted as replacing its variables' values with the corresponding variables' values of the better solution with an appropriate probability, in order to avoid the immature convergence of the algorithm. In this regard, this replacement is done by considering  $IF()$  as the probability of this replacement. We normalize the Impact Factor between 0 and 1 according to (11).

$$IF(i)_{Normalized} = 1 - \frac{F(S(i)) - F(Bestsolution)}{F(Worstsolution) - F(Bestsolution)} \quad (11)$$

Here,  $F(Bestsolution)$  and  $F(Worstsolution)$  are the expected qualities of the best and the worst solutions. In fact, the larger  $IF(i)$  is, the more chance participant  $i$  ( $S_i$ ) has to impact the other participant' solutions. This replacement is done according to (12).

$$Rep(S_i, S_j) = \begin{cases} S_j(k) \leftarrow S_i(k) \\ \text{if } S_j(k) \neq S_i(k) \\ \text{and } rnd < IF(I) \end{cases} \quad (12)$$

$Rep(S_i, S_j)$  is the replacement equation,  $rnd$  is a random number in (0,1). The controlling parameter,  $CP$ , is used to detect if the FGOA has been trapped in local optimum solutions. This parameter through (4) monitors the progress trend of the algorithm and if for some iterations not enough progress has been made by the algorithm, this parameter enables a randomization method to diversify the solutions.

$$CP = \frac{\sum_{i=IN-WS}^{IN} (GB(i) - GB(i-1))}{WS} \quad (13)$$

Here,  $IN$  is the current iteration number,  $WS$  is the window size,  $GB(i)$  is the global best solution in iteration  $i$ . The window size determines the number of iterations to be considered to determine if acceptable progress has been made by the algorithm. If  $CP$  is less than the user-defined threshold value the algorithm activates a new randomization method called  $IF$  Randomization (IFR).  $IF$  Randomization method is used for diversifying the solutions. According to this method based on the Impact Factor ( $IF$ ) of a solution, a variable values of a given solution is replaced with another value which is randomly chosen from its domain with probability  $(1-IF)^2$ . The probability  $(1-IF)^2$  causes more quality solutions to be subject of less changes in their variables values.

#### 5.4 Discrete Harmony Search (DHS) Algorithm

Harmony Search (HS) optimization algorithm is a population based metaheuristic algorithm which was developed by Geem et al. in 2001 (Geem et al., 2001) based on improvisation process of jazz musicians. Improvisation process stands for the attempt of a musician to find the best harmony that can be achieved in practice (Degertekin, 2008). Three options can be considered when a skilled musician aims at improvising a music instrument, a) to play a memorized piece of music exactly, b) to play a piece similar to what he/she has in their memory and c) to play newly composed notes (Yang, 2009). These three options were considered as the main components of the HS algorithm which were introduced as harmony memory, pitch adjustment and random search to the algorithm (Yang, 2009). The harmony memory has valuable role in HS algorithm and that is to ensure that good harmonies are considered when generating new solutions. This component is controlled by a parameter called harmony memory considering,  $HMCR \in [0, 1]$ . In fact this parameter determines the ratio of considering elite solutions (harmonies) in generating a new solution. If this parameter is set to a small value, the algorithm considers a small number of elite solutions, which causes convergence to the optimal solution too slowly. On the other hand if it is set to a large value, the emphasis of the algorithm will be on using the solutions in the memory and therefore other good solutions are not explored. This does not lead to discovering better solutions.

The next component is pitch adjustment which has the same application as the mutation operator in genetic algorithms and is defined as follows.

$$X_{new} = X_{old} + BW * \epsilon \quad (14)$$

Here,  $X_{old}$  is the solution (pitch) in the memory,  $BW$  is the band width,  $\epsilon$  is a random value in  $(0,1)$  and  $X_{new}$  is the new solution. This component generates solutions slightly different from those in the memory by adding small random values to the solutions in memory. The degree of pitch adjustment can be controlled by pitch adjustment rate parameter  $PAR$ . The low value of  $PAR$  together with the small value of  $BW$  can reduce the exploration which results in discovering a portion of the problem space instead of the whole problem space.

The third component of the HS algorithm is randomization. The main role of this component is to encourage the diversity of the solutions. Randomization ensures that all regions of the problem space are accessible by the algorithm.

HS algorithm was developed to tackle continuous optimization problems. In order to deal with WCSPs, HS features must be changed to suit the discrete problem spaces, following our definitions in Section 4. The steps of converting the HS algorithm to its discrete version (that we call Discrete HS or DHS) are reported below. At the initialization step, the algorithm randomly generates  $HMS$  (harmony search size) solutions as the initial population. In this step, potential solutions are generated by assigning random values (from the variables domains) to CSP variables.

The next step is to redefine the pitch adjustment equation presented in Equation 14. The new definition of the pitch adjustment is presented in Equation 15.

$$X_{new} = (BW \odot \epsilon) \otimes X_{old} \quad (15)$$

Through pitch adjustment, slight changes by adding small random values are made in the current solution in order to improve it. The new definition of the pitch adjustment component has the same impact on the current solutions. Through the new definition, the variables' values of the current solution will be replaced by randomly picking values from variables' domain in the hope to improve the current solution. In Equation 15, we defined  $\odot$  and  $\otimes$  as follows.  $\odot$  is the multiplier and  $BW \odot \epsilon$  is a probability value in  $(0,1)$ . In fact, this latter term determines the probability of replacing the variables' values of  $X_{old}$  with new values picked up randomly from variables domain.  $\otimes$  applies these changes to  $X_{old}$ .

The last component (randomization) encourages diversity of solutions which ensures that a larger search space will be considered. To boost the diversity of the solutions, we employ the GA mutation operator. In this regard, different mutation operators have been developed so far, including the following three: Random Resetting Mutation (RRM), Scramble Mutation (SM), and Inversion Mutation (IM).

## 5.5 Artificial Bee Colony (ABC)

Artificial Bee Colony optimization algorithm (ABC) is a population based optimization algorithm which simulates the foraging behavior of the real bee colonies in the nature (Karaboga, 2005b; Mernik et al., 2015). Agents of the ABC algorithm is divided into three class of bees, recruited bees, onlooker bees and scout bees and each class of bees shoulders responsibilities (Gao and Liu, 2012). Recruited bees search the food sources around the location they have in their memories and keep the onlooker bees updated about the quality of the food sources they are visiting. Onlooker bees based on the information are receiving from recruited bees select the new food sources (the most quality ones) and also search around the selected food sources in the hope of discovering new and more quality food sources. Scout bees are recruited bees that abandoned their food source in order to discover new food sources (Karaboga, 2005b). In order to apply the ABC algorithm for WCSPs, we need to adapt it to discrete spaces, using the definitions reported in Section 4. The initial population of bees is generated by assigning the random values from variables' domain to the variables of the solutions. After the generated solutions were assigned to recruited bees, they try to locally improve the solutions by searching neighboring areas of their solutions (food sources). This search process is guided by the following equations, redefined for discrete spaces.

$$v_{ij} = w \otimes (x_{ij} \ominus \forall_{ij}) \quad (16)$$

where

$$\forall_{ij} = \emptyset_{ij} \otimes (x_{ij} \ominus x_{kj}) \quad (17)$$

$v_{ij}, x_{ij}$  and  $\forall_{ij}$  correspond to the  $j^{th}$  variable value for potential solutions  $v_i, x_i$  and  $\forall_i$ , respectively. We have redefined the operators  $\otimes$  and  $\ominus$  to deal with discrete problems. The idea is that the algorithm randomly selects a food source site  $k$  for each recruited bee in its neighborhood. Then, the recruited bee needs to move towards that new site  $x_k$ . In discrete problem spaces, moving from one solution to another corresponds to sharing more identical values with that solution. Operator  $\ominus$  identifies the variables of the first solution that have different values from the one the bee is going to move toward. After identifying the variables with different values, these variables' values will be replaced by the corresponding variables' values of solution  $x_k$  with probability  $\emptyset_{ij}$  (a value randomly selected from  $[0,1]$ ). The application of  $\otimes$  is to replace the variables' values of  $x_i$  with those of  $x_k$  considering the replacement probability  $\emptyset_i$ . Then,  $\ominus$  compares  $x_{ij}$  and  $\forall_{ij}$ , for each  $j$ , to identify the different variables. When the differences have been identified, those different variables' values will be replaced

$x_i$	2	1	3	5	5	4	1	2	3	4
$x_k$	3	1	4	3	5	4	4	5	1	3
$\theta_{ij}$	0.1	0.4	0.1	0.6	0.3	0.3	0.6	0.5	0.7	0.4
$\gamma_i$	2	1	3	3	5	4	4	5	1	4

Figure 2: An example of calculating  $\gamma_i = \theta_{ij} \otimes (x_{ij} \ominus x_{kj})$ .

by corresponding variables' values of  $\gamma_i$  with probability  $w$ . Figure 2 shows an example of calculating  $\gamma_{ij} = \theta_{ij} \otimes (x_{ij} \ominus x_{kj})$ .

Finally,  $v_i$  will be assessed according to the fitness function. If it has higher quality than  $x_i$  then it will replace it. The improvement of developing every food site is monitored using a given parameter. If for a defined number of trials, the expected improvement has not been achieved, the algorithm replaces that site (solution) with a randomly generated site.

## 6 EXPERIMENTATION

To assess the performances of the nature-inspired techniques considered in this paper, we have conducted several comparative experiments on randomly generated WCSP instances using a variant of the RB model (Xu and Li, 2000). The original RB model has been designed to produce random CSP instances, based on the following four parameters:  $n$ ,  $p$ ,  $\alpha$  and  $r$ . Here,  $n$  is the number of variables,  $p$  ( $0 < p < 1$ ) determines the tightness of the constraints, and  $r$  and  $\alpha$  ( $0 < r, \alpha < 1$ ) are two positive constants used by the model RB (Xu and Li, 2000). Using these four parameters, we have adapted the RB model to generate a WCSP instance as follows.

1.  $d = n^\alpha$  is the domain size of each variable.
2. All the variables will be assigned the same domain corresponding to the first  $d$  natural numbers ( $0 \dots d-1$ ).
3. Select without repetition  $t = r \times n \times \ln(n)$  random constraints. Each random constraint is formed by selecting 2 of  $n$  variables (without repetition).
4. For each constraint, we uniformly select without repetition  $q = p \times d^2$  incompatible pairs of values. We then assign a large value (1000) to all the incompatible pairs. From the remaining pairs (the compatible ones), we select a percentage of  $sc$  soft constraints, and assign a random value to each of them, from  $[1, \dots, 999]$ . The rest of the constraints are hard constraints and will therefore each be assigned a value of 0.

According to (Xu and Li, 2000), the phase transition  $P_t$  (borderline between solvable and non solvable

Table 3: The controlling parameters of the nature-inspired techniques.

Algorithm	Controlling Parameters
MRO	population size: 50 $w = 0.6$ $nSpore = 2$ $rs = 1$
HS	Harmony Memory Size (HMS) = 50; Number of New Harmonies (nNew) = 60 Harmony Memory Consideration Rate (HMCR) = 0.6 Pitch Adjustment Rate (PAR) = 0.6;
PSO	Population size: 50 $w = 0.6$ $c1 = 2$ $c2 = 3$ $r1 = 0.2$ $r2 = 0.2$
GA	Population size: 50 Selection function: Tournament Crossover Percentage (pc) = 0.3 Mutation Percentage (MP) = 0.2 Mutation Rate (MR) = 0.3
FA	Population size: 50 $\gamma = 0.009$ $\Theta = 0.4$
ABC	Number of employed bees: 25 Number of onlooker bees: 25
DFGOA	Population size: 50 Inertia weight: 0.8

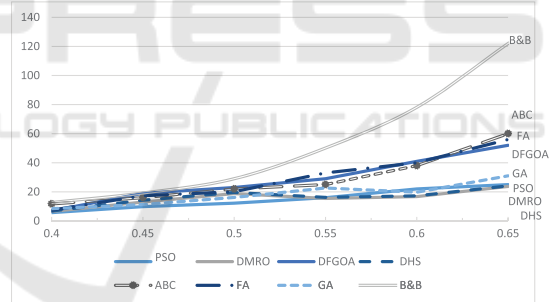


Figure 3: Solving WCPS instances with tightness between 0.45 and 0.65.

problems) is calculated as follows:  $P_t = 1 - e^{-\alpha/r}$ . In theory, if the constraints are all hard, solvable problems are generated with tightness  $p < P_t$ . In our experiments, we select the values for  $\alpha$  and  $r$  such that  $P_t = 0.7$ . In this regard, we randomly generate WCSP instances with 100 variables and a tightness  $p$  ranging from 0.4 to 0.65 (the latter value corresponds to the hardest problem to solve). We set  $sc$  to 0.3 which means that 30% of randomly chosen hard constraints will be converted to soft constraints. All methods have been implemented using MATLAB R2013b and all experiments have been performed on a PC with Intel Core i7-6700K, with 4.00 GHz processor and 32 GB RAM.

Figure 3 reports on the comparative results of the



experiments conducted on the nature-inspired techniques we adapted in this paper, namely PSO, DMRO, DFGOA, DHS and ABC. In addition, we use FA as reported in (Bidar and Mouhoub, 2019b), the GA approach in (Abbasian and Mouhoub, 2016), and a branch and bound (B&B) algorithm (Lawler and Wood, 1966) for comparative purposes. The latter algorithm has been improved using constraint propagation (Arc Consistency) before and during search, following a forward check strategy (Dechter and Cohen, 2003). This will help reducing the size of the search space by removing some of the inconsistent values (according to hard constraints). The controlling parameters of the nature-inspired techniques are tuned to their best, as shown in Table 3. The different methods are compared in terms of the running time (Y-axis in 3) needed to return the optimal solution, for each tightness value (X-axis). Note that in all the experiments conducted, all the methods were able to return the optimal solution, before a given timeout. The charts show the superiority of GAs, MTO, PSO and DHS over ABC, FA and DFGOA. Due to its inherent exponential time cost, B&B performs poorly comparing to all the other methods.

## 7 CONCLUSION AND FUTURE WORK

This work investigate the applicability of the metaheuristics for solving WCSPs. In this regard, we report on the adaptation of several known nature-inspired techniques in order to tackle these discrete optimization problems. To assess the performance of our developed methods, we conducted several experiments on WCSP instances, randomly generated using a variant of the model RB. The results are promising and demonstrate the fact that nature-inspired methods can be a good alternative for tackling these NP-hard problems. In the near future, we plan to apply our adapted nature-inspired techniques to solve those real-world problems that can be represented as WCSPs. These applications include Nurse Scheduling Problems (NSPs) (Said et al., 2021) and timetabling (Hmer and Mouhoub, 2016). Given that most of these real-world problems occur in an evolving environment where constraints can added or removed dynamically, we will need to adapt our techniques to operate in an incremental manner (Bidar and Mouhoub, 2019a; Mouhoub, 2004).

## REFERENCES

- Abbasian, R. and Mouhoub, M. (2016). A new parallel ga-based method for constraint satisfaction problems. *Int. J. Comput. Intell. Appl.*, 15(3):1650017:1–1650017:22.
- Bidar, M., Kanan, H. R., Mouhoub, M., and Sadaoui, S. (2018a). Mushroom reproduction optimization (MRO): A novel nature-inspired evolutionary algorithm. In *2018 IEEE Congress on Evolutionary Computation, CEC 2018, Rio de Janeiro, Brazil, July 8-13, 2018*, pages 1–10. IEEE.
- Bidar, M., Kanan, H. R., Mouhoub, M., and Sadaoui, S. (2018b). Mushroom reproduction optimization (MRO): A novel nature-inspired evolutionary algorithm. In *2018 IEEE Congress on Evolutionary Computation, CEC 2018, Rio de Janeiro, Brazil, July 8-13, 2018*, pages 1–10. IEEE.
- Bidar, M. and Mouhoub, M. (2019a). Discrete particle swarm optimization algorithm for dynamic constraint satisfaction with minimal perturbation. In *2019 IEEE International Conference on Systems, Man and Cybernetics, SMC 2019, Bari, Italy, October 6-9, 2019*, pages 4353–4360. IEEE.
- Bidar, M. and Mouhoub, M. (2019b). Solving weighted constraint satisfaction problems using a new self-adaptive discrete firefly algorithm. In *2019 IEEE International Conference on Systems, Man and Cybernetics, SMC 2019, Bari, Italy, October 6-9, 2019*, pages 2198–2205. IEEE.
- Bidar, M., Mouhoub, M., and Sadaoui, S. (2020a). Discrete focus group optimization algorithm for solving constraint satisfaction problems. In Rocha, A. P., Steels, L., and van den Herik, H. J., editors, *Proceedings of the 12th International Conference on Agents and Artificial Intelligence, ICAART 2020, Volume 2, Valletta, Malta, February 22-24, 2020*, pages 322–330. SCITEPRESS.
- Bidar, M., Mouhoub, M., Sadaoui, S., and Kanan, H. R. (2020b). A novel nature-inspired technique based on mushroom reproduction for constraint solving and optimization. *Int. J. Comput. Intell. Appl.*, 19(2):2050010:1–2050010:21.
- Blum, C. and Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, 35(3):268–308.
- Cooper, M. C., De Givry, S., Sánchez, M., Schiex, T., Zytnicki, M., and Werner, T. (2010). Soft arc consistency revisited. *Artificial Intelligence*, 174(7-8):449–478.
- Dechter, R. and Cohen, D. (2003). *Constraint processing*. Morgan Kaufmann.
- Degertekin, S. O. (2008). Optimum design of steel frames using harmony search algorithm. *Structural and Multidisciplinary Optimization*, 36(4):393–401.
- Delisle, E. and Bacchus, F. (2013). Solving weighted cpsps by successive relaxations. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming, CP'13*, page 273–281, Berlin, Heidelberg. Springer-Verlag.

- Fattahi, E., Bidar, M., and Kanan, H. R. (2018). Focus group: An optimization algorithm inspired by human behavior. *International Journal of Computational Intelligence and Applications*, 17:1–27.
- Gallardo, J. E., Cotta, C., and Fernández, A. J. (2009). Solving weighted constraint satisfaction problems with memetic/exact hybrid algorithms. *Journal of Artificial Intelligence Research*, 35:533–555.
- Gao, W.-f. and Liu, S.-y. (2012). A modified artificial bee colony algorithm. *Computers & Operations Research*, 39(3):687–697.
- Geem, Z. W., Kim, J. H., and Loganathan, G. (2001). A new heuristic optimization algorithm: Harmony search. *SIMULATION*, 76(2):60–68.
- Glover, F. W. and Kochenberger, G. A. (2006). *Handbook of metaheuristics*, volume 57. Springer Science & Business Media.
- Haddouch, K., Elmoutaoukil, K., and Ettaouil, M. (2016). Solving the weighted constraint satisfaction problems via the neural network approach. *Int. J. Interact. Multimed. Artif. Intell.*, 4(1):56–60.
- Hmer, A. and Mouhoub, M. (2016). A multi-phase hybrid metaheuristics approach for the exam timetabling. *International Journal of Computational Intelligence and Applications*, 15(04):1650023.
- Karaboga, D. (2005a). An idea based on honey bee swarm for numerical optimization. In *Technical report-tr06, Erciyes university, engineering faculty, computer engineering department*.
- Karaboga, D. (2005b). An idea based on honey bee swarm for numerical optimization. *Technical report-tr06, Erciyes university, computer engineering department*, 200:1–10.
- Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE.
- Korani, W. and Mouhoub, M. (2020). Discrete mother tree optimization for the traveling salesman problem. In Yang, H., Pasupa, K., Leung, A. C., Kwok, J. T., Chan, J. H., and King, I., editors, *Neural Information Processing - 27th International Conference, ICONIP 2020, Bangkok, Thailand, November 23-27, 2020, Proceedings, Part II*, volume 12533 of *Lecture Notes in Computer Science*, pages 25–37. Springer.
- Korani, W. and Mouhoub, M. (2021). Review on Nature-Inspired Algorithms. *SN Operations Research Forum*, 2(3):1–26.
- Kumar, V. (1992). Algorithms for constraint-satisfaction problems: A survey. *AI magazine*, 13(1):32–32.
- Larrosa, J. and Schiex, T. (2003). In the quest of the best form of local consistency for weighted csp. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence, IJCAI'03*, page 239–244, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- Lau, H. C. (2002). A new approach for weighted constraint satisfaction. *Constraints*, 7(2):151–165.
- Lawler, E. L. and Wood, D. E. (1966). Branch-and-bound methods: A survey. *Oper. Res.*, 14(4):699–719.
- Lee, J. H. and Leung, K. L. (2009). Towards efficient consistency enforcement for global constraints in weighted constraint satisfaction. In *Twenty-First International Joint Conference on Artificial Intelligence*.
- Mernik, M., Liu, S.-H., Karaboga, D., and Črepinšek, M. (2015). On clarifying misconceptions when comparing variants of the artificial bee colony algorithm by offering a new implementation. *Information Sciences*, 291:115–127.
- Mouhoub, M. (2004). Systematic versus non systematic techniques for solving temporal constraints in a dynamic environment. *AI Commun.*, 17(4):201–211.
- Nickabadi, A., Ebadzadeh, M. M., and Safabakhsh, R. (2011). A novel particle swarm optimization algorithm with adaptive inertia weight. *Applied soft computing*, 11(4):3658–3670.
- Poli, R., Kennedy, J., and Blackwell, T. (2007). Particle swarm optimization. *Swarm intelligence*, 1(1):33–57.
- Said, A. B., Mohammed, E. A., and Mouhoub, M. (2021). An implicit learning approach for solving the nurse scheduling problem. In Mantoro, T., Lee, M., Ayu, M. A., Wong, K. W., and Hidayanto, A. N., editors, *Neural Information Processing - 28th International Conference, ICONIP 2021, Sanur, Bali, Indonesia, December 8-12, 2021, Proceedings, Part II*, volume 13109 of *Lecture Notes in Computer Science*, pages 145–157. Springer.
- Schiex, T. (2000). Arc consistency for soft constraints. In *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming, CP '02*, page 411–424, Berlin, Heidelberg. Springer-Verlag.
- Talbi, E.-G. (2009). *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons.
- Tsang, E. (2014). *Foundations of constraint satisfaction: the classic text*. BoD—Books on Demand.
- Črepinšek, M., Liu, S.-H., and Mernik, M. (2013). Exploration and exploitation in evolutionary algorithms: A survey. *ACM Comput. Surv.*, 45(3).
- Wang, H., Wang, W., Zhou, X., Sun, H., Zhao, J., Yu, X., and Cui, Z. (2017). Firefly algorithm with neighborhood attraction. *Inf. Sci.*, 382(C):374–387.
- Xu, K. and Li, W. (2000). Exact phase transitions in random constraint satisfaction problems. *Journal of Artificial Intelligence Research*, 12:93–103.
- Yang, X.-S. (2008). *Nature-Inspired Metaheuristic Algorithms*. Luniver Press.
- Yang, X.-S. (2009). *Harmony Search as a Metaheuristic Algorithm*, pages 1–14. Springer Berlin Heidelberg, Berlin, Heidelberg.