Dragonfly Algorithm Application for Solving the Nurse Scheduling Problem

Arghavan Ayoughi, Chi Zhang
Stephen Okeleke, Shufei Li
Wei Xin, Yuhan Yue
Khoury College of Computer Sciences
Northeastern University
Vancouver, Canada
vancouver@northeastern.edu

Ania Aibin
School of Health Sciences
Department of Nursing
British Columbia Institute of Technology
Vancouver, Canada
aaibin@bcit.ca

Michal Aibin
School of Computing and Academic Studies
Department of Computing
British Columbia Institute of Technology
Vancouver, Canada
maibin@bcit.ca

***Abstract*—Scheduling plays a crucial role in allocating resources to tasks in various domains, including healthcare. Nurse scheduling, in particular, presents significant challenges due to the limited number of nurses. This paper illustrates how a novel meta-heuristic model known as Dragonfly Algorithm (DA) is used to efficiently solve the nurse scheduling problem faced by a major hospital in Belgium. The DA, inspired by the swarming behaviours of dragonflies, offers a new approach to solving this problem. The algorithm's effectiveness is evaluated using Shift Scheduling Benchmark Datasets, and the results show its potential for generating nurse schedules with reduced penalties.**

***Index Terms*—roster scheduling, dragonfly optimization, swarm intelligence**

I. Introduction

Scheduling involves assigning resources to tasks [1]. These resources could be machines in a workshop, processors in a computer or employees in the industry. The tasks can range from manufacturing jobs to threads or waiting tables at a restaurant. However, the availability of resources and tasks often does not match the desired level of operation. Developing a roster focuses on optimal work schedules for employees, obtaining the conditions of the organization, employees, and labour laws, to meet the organization's labour demand [2]. Certain sectors, such as social emergencies (medical care, fire department services, police), require 24-hour services and preparation of optimal timetables for work shifts and job rotations. These timetables must consider employee well-being, including satisfaction, health, stress level, and motivation, as they can significantly impact employee performance. This paper specifically addresses healthcare personnel's scheduling challenges, primarily focusing on nurses.

The global pandemic in 2020 put immense pressure on the healthcare system worldwide. The efficient allocation of healthcare resources has become crucial during such situations. Healthcare personnel scheduling, particularly for nurses, gained increasing significance [3], [4]. Scheduling the right nursing staff improves the performance and quality of the nursing units, benefiting multiple parties. It enhances operational efficiency, and resource allocation, ensures the safety and satisfactory experiences of staff and patients, and alleviates the workload of the hospital management and administration [5].

Nurse Scheduling Problems (NSP), also known as roster scheduling of nurses, pose challenges due to the relatively limited number of nurses compared to the number of patients in hospitals [6]. Effective nurse scheduling aims to distribute the workload fairly among nurses while adhering to scheduling restrictions. However, fulfilling all scheduling restrictions becomes challenging as meeting one constraint may lead to the violation of others. Efficient scheduling involves assigning nurses to shifts to maximize hospital benefits while considering hard and soft constraints such as duty hours, hospital regulations, and more. Finding combinatorial solutions to satisfy multiple constraints is a delicate task. NSP establishes a periodic schedule for a number of nurses by allocating one out of a possibly infinite number of shift schedules to each nurse. The roster must consider different contracts, nurse requirements (varying ranks), and nurse preferences. Additionally, management's desire to minimize cost or maximize profit influences the scheduling outcomes; therefore recognizing NSP as an NP-hard optimization problem [7].

Scheduling methods have evolved over the decades, transitioning from the traditional manual self-scheduling or cyclical scheduling to Dynamic Programming and Reinforcement learning. Other models such as the Liner Model and Genetic Programming have also emerged. This paper explores using the Dragonfly algorithm (DA) for solving the NSP.

II. Literature Review

Scholars across the globe have dedicated considerable attention to studying the nurse rostering problem. Early research in this field explored various aspects, including personnel scheduling, financial costs, and nurse management issues [8]. In addition to traditional methods like manual self-scheduling and cyclical scheduling, mathematical programming and optimization techniques gradually emerged [8]. However, developing a generic model for this problem occurred later through international competitions, such as INRC in 2010 [9]. This model considers nurses' assignments in relation to shifts while respecting hard constraints and minimizing violations of soft constraints. The violations of constraints lead to penalization, and thus the solution aims to optimize the assignment with minimal penalties [10]. This topic has gained even more attention in recent years with the emergence of new optimization methods and algorithms.

979-8-3503-7099-7/24/$31.00 ©2024 IEEE

This NSP problem can also be formulated as a Multiple-Choice Knapsack Problem [11]. In this formulation, the knapsack represents the requirements of shifts, and the number of staff for a valid scheduled day, while nurses can fill in their personal preferences. Hard and soft constraints are associated with rewards and penalties in the final cost function.

Genetic Algorithms (GAs) have gained popularity for solving complex optimization problems, and several researchers have used them to solve staff-scheduling problems. GAs are probabilistic search algorithms which mimic biological evolution to produce better offspring solutions gradually [12].

Linear programming is another common approach for solving scheduling problems. It is a mathematical model that maximizes or minimizes a linear function, often known as an objective function subject to constraints. Lorraine Trilling et al. (2006) proposed a solution to maximize shift fairness of anaesthesiology nurses in a French public hospital subject to several constraints [13]. They utilized an Integer Linear Programming (ILP) approach and a Constrained Programming (CP) approach. The ILP approach outperformed the CP approach in terms of speed and performance. Despite conducting an experiment over a 12-hour period with a sample of 20 nurses, they were unable to achieve the optimal value. It is important to note that meta-heuristic algorithms do not guarantee an optimal solution. However, they provide faster and more efficient solutions than Linear Programming techniques.

To the best of our knowledge, this is the first paper that introduces the DA to the nurse scheduling problem.

III. Problem Description

A. Problem Statement

NSP involves finding an optimal way to assign shifts and days off for nurses. Each nurse or physician has their own desired schedule. The task is to create weekly schedules for nurses on specific wards by assigning one of many possible shift patterns to each nurse. These schedules must satisfy contract requirements and meet the demand for a given number of nurses on each shift while being perceived as fair by the staff. Achieving fairness involves accommodating as many nurses' requests as possible and evenly distributing unsatisfied requests and unpopular shifts. This problem also has a unique day-night structure, as day and night shifts may have different requirements. For example, nurses may require longer breaks after night shifts. These characteristics make the problem challenging for local search algorithms, as finding and maintaining feasible solutions is extremely difficult.

In this paper, we work on Shift Scheduling Benchmark Data Sets provided by the KU Leuven University website, specifically instance 6. This data set contains realistic and straightforward instance data, including information about nurses, schedules, and constraints over 28 days. Our instance assumes that all nurses possess the same skill type and work on a 3-shift schedule (early/late/night shift).

B. Description of Constraints

We aim to assign shifts to nurses over 28 days, subject to several constraints. The constraints are divided into two categories: hard and soft constraints. Hard constraints must be satisfied by all feasible solutions and are related to administrative and union contract specifications. Soft constraints are desirable but not obligatory and can be violated. Violating a hard constraint renders the schedule invalid and incurs a penalty, while violations of soft constraints also result in penalties but do not invalidate the schedule. The goal is to minimize the penalty and find a valid schedule. A shift represents a block of time during work duty. Shift types in our data include early, day, and night shifts. A sequence is a series of shifts for a nurse that happen on consecutive days. A schedule consists of multiple sequences and the day-off periods between them, and it is designated for each nurse individually. The constraints are as follows.

Let $Nur = \{1, 2, \ldots, n\}$ represent the set of nurses, and let $Day = \{1, 2, \ldots, 28\}$ represent the set of days in the scheduling period. Define M as a matrix where $M_{i,j}$ represents the shift assigned to nurse i on day j, with $M_{i,j} \in \{E, D, N, O\}$ representing Early, Day, Night shifts, and Off respectively.

Hard Constraints

1) One Shift Per Day:

$$\forall i \in Nur, \forall j \in Day, \sum_{s \in \{E,D,N\}} \mathbf{1}_{\{M_{i,j}=s\}} \leq 1$$

2) No Back-to-Back Shift Sequences:

$$\forall i \in Nur, \forall j \in Day \backslash \{1\}, M_{i,j} = O \implies M_{i,j-1} = O$$

3) Maximum Consecutive Shifts ≤ 4:

$$\forall i \in Nur, \forall j \in Day, \sum_{k=j}^{\min(j+3,28)} \mathbf{1}_{\{M_{i,k} \neq O\}} \leq 4$$

4) Minimum Days Off After Shifts ≥ 2:

$$\forall i \in Nur, \forall j \in Day, M_{i,j} = O \implies \sum_{k=\max(1,j-1)}^{\min(j+2,28)} \mathbf{1}_{\{M_{i,k}=O\}} \geq 2$$

5) Maximum Working Weekends:

$$\forall i \in Nur, \sum_{\substack{j \in Day \\ j \text{ is Sat or Sun}}} \mathbf{1}_{\{M_{i,j} \neq O\}} \leq 4$$

Soft Constraints

1) Preferred Shift Types:

$$\text{Minimize} \sum_{i \in Nur, s \in \{E,D,N\}} \mathbf{1}_{\{M_{i,j} \neq s\}} \times PrefShift_{i,s}$$

2) Days Requested Not to Work:

$$\text{Minimize} \sum_{i \in Nur, j \in Day} \mathbf{1}_{\{M_{i,j} \neq O\}} \times RequestOff_{i,j}$$

C. Objective Function

In our dragonfly algorithm, the position vector represents $Nur \times Day$ matrix M. Each nurse's schedule (row $m_{i,*}$) represents a single dragonfly, and each element ($m_{i,j}$) represents the position of the dragonfly which corresponds to the shift type. We also define a cost function for our NSP to return a total weighted cost value. We mark the cost function as:

$$\text{Total Weighted Cost} = \sum_{i \in Nur, j \in Day} \left(Cov_{violation_{i,j}} + Sch_{violation_{i,j}} \right)$$

This function considers weighted shift coverage costs Cov and schedule requirement costs Sch. The coverage cost considers the coverage situation within the ward; for instance, a certain number of nurses must fulfill all shift duties. The schedule requirement costs consider the other constraints required to ensure feasible schedules, such as the requirement of day-off shifts after a night shift. As all penalties from the violation of soft and hard constraints are under consideration in the cost function, lower costs indicate less violation of constraints. Therefore, we aim to find a roster schedule with the lowest overall penalty (minimum total cost).

IV. Algorithm Description

A. Overview of DA

The Dragonfly algorithm (DA) is a new swarm intelligence optimization algorithm that Mirjalili proposed in 2015 [14]. It is a new optimization technique for solving single-objective, discrete, and multi-objective problems. The DA is inspired by two unique clusters of natural dragonflies: foraging groups (also known as static groups) and migratory groups (known as dynamic groups).

The key idea of DA is mimicking the swarming behaviours of a dragonfly. For a group of dragonflies, there are only two reasons for them to swarm - migration or hunting. In a foraging group (static group), dragonflies form a small group and move over to a small area to hunt other insects. Local motion and mutation of the moving path are the foremost features of a static group. However, in a migratory group (dynamic group), many dragonflies will be going on long-distance migration in one specific direction.

There are three principles for every dragonfly group: *Separation, Alignment and Cohesion*. Separation is to avoid collision when dragonflies are in the same neighbourhood. Alignment is a habit of a search agent that adjusts its velocity to the other agents in the same neighbourhood. Cohesion is a habit of dragonflies that flies toward the center of search agents. For any group, the final target is survival; everyone tends to be attracted to food and distracted from the enemy. Hence, 5 factors affect individuals' positions in a group.

B. Math Model

As we mentioned before, to direct artificial dragonflies to various paths, five weights were used, which are separation weight (s), alignment weight (a), cohesion weight (c), food factor (f) and enemy factor (en). This section will present how to express and calculate those factors.

Separation stands for individuals that would maintain an appropriate distance between each other to avoid a collision. It could be calculated as:

$$S_i = -\sum_{j=1}^{N} X - X_j \tag{1}$$

Where *X* indicates the position for the current dragonfly, X_j is the position for the j_th neighbouring dragonfly, *N* is the number of individual neighbours of the dragonfly swarm, and *S* indicates the separation motion for the i_th individual.

Alignment is a habit of a search agent that adjusts its velocity to the other search agents in the same neighbourhood. *Equation 3* is used for calculating the alignment:

$$A_i = \frac{\sum_{j=1}^{N} V_j}{N} \tag{2}$$

where A_i is the alignment motion for i_th individual and *V* is for the velocity of the j_th neighbouring dragonfly.

Cohesion is a habit of dragonflies that flies toward the center of search agents. It can be expressed as follows:

$$C_i = \frac{\sum_{j=1}^{N} X_j}{N} - X \tag{3}$$

where C_i is the cohesion for i_th individual, *N* is the neighbourhood size, X_j is the position of the j_th neighbouring dragonfly, and *X* is the position of current dragonfly.

Attraction is dragonflies staying as close as possible to access their food. It could be computed as follows:

$$F_i = X^+ - X \tag{4}$$

where F_i is the attraction of food for ith dragonfly,*X* is the position of the source of food, and *X* is the position of the current dragonfly individual. Here, the food is the dragonfly with the best objective function.

Distraction are outward predators which are calculated as follows:

$$En_i = X^- + X \tag{5}$$

where En_i is the enemy's distraction motion for the ith individual,X^- is the enemy's position, and *X* is the position of the current dragonfly individual.

For position updating in the search space, artificial dragonflies use step vector ΔX and position vector *X*. The step vector is an analogy to the velocity vector in the PSO algorithm. The position updating is also based mainly on the PSO algorithm framework. The step vector is defined in Equation (7) as follows:

$$\Delta X_{t+1} = (sS_i + aA_i + cC_i + fF_i + enEn_i) + w\Delta X_t \tag{6}$$

where S_i represents the separation for the ith dragonfly, A_i is the alignment for ith dragonfly, C_i represents the cohesion for ith dragonfly, F_i represents the food source for the ith individual, En_i represents the position of the enemy for i_th

dragonfly, w represents the inertia weight and *t* indicates the iteration counter.

When the step vector calculation is finished, the calculation for the position vectors starts as follows:

$$X_{t+1} = X_t + \Delta X_{t+1} \tag{7}$$

where *t* indicates the current iteration.

The above math model could successfully handle many scenarios; however, there are some restrictions for all the above formulas. This math model cannot solve the problem if no neighbours are updated around the current dragonfly; the separation, alignment and cohesion calculation would be meaningless. To provide a solution to this situation, we assumed that every dragonfly would make a random move if there were no neighbours. We will apply the *Le'vy* flight rule to simulate the randomness of a dragonfly. A *Le'vy* flight is a random walk in which the step lengths have a Lévy distribution, a heavy-tailed probability. When a walk in a space with far more than one dimension is established, the steps taken are in an isotropic random direction. *Le'vy* flight rules can be presented mathematically as:

$$Le'vy(x) = 0.01 \times \frac{t_1 \times \sigma}{|t_2|^{\frac{1}{\beta}}} \tag{8}$$

where $t1$ and $t2$ are the random numbers between 0 and 1, $\beta = 0.5$, and σ can be calculated as

$$\sigma = \left(\frac{\Gamma(1+\beta) \times \sin\left(\frac{\pi\beta}{2}\right)}{\Gamma\left(\frac{1+\beta}{2}\right) \times \beta \times 2^{\frac{\beta-1}{2}}} \right)^{\frac{1}{\beta}} \tag{9}$$

Then we can get our updated rule of dragonfly position with random moves as follows:

$$X_{t+1} = X_t + Le'vy(d) \times X_t \tag{10}$$

C. Algorithm Process

The DA starts with random initialization and then randomly returns a new dragonfly's step vector ΔX and position vector *X* between the upper and lower bounds of the solution. In order to find more neighbours, the neighbourhood radius of the dragonfly will increase with the increase of iterations so that the dragonfly eventually forms a big group and performs a local search. To guarantee that the optimization process is random and that the neighbourhood radius is large, the following formula is used to update all the weight values from Equation (7) in each iteration:

$$w = 0.9 - iter * ((0.9 - 0.4)/Max\_iter) \tag{11}$$

$$s, a, c = 2 * random * my\_c \tag{12}$$

$$f = 2 * random \tag{13}$$

$$e = my\_c \tag{14}$$

$$radius = (ub - lb)/4 + ((ub - lb) * (iter/Max\_iter) * 2) \tag{15}$$

$$my\_c = 0.1 - \text{iter} * ((0.1 - 0)/(Max\_iter/2)) \tag{16}$$

Where random is the random number between 0 and 1, and *radius* stands for the neighbourhood radius of the dragonfly.

To sum up, the algorithm process to locate the neighbours surrounding the current individual updated to use the Equations (2) to (6) to get *S, A, C, F, E* and the Euclidean distance between the individual updated and all other dragonflies. If there are neighbours, update ΔX using equation (7) and then update the position vector *X* with equation (8); if there are no neighbours, update *X* directly with equation (11). The updating of the dragonfly position continues throughout the optimization procedure until the iteration condition reaches the maximum iteration number. Algorithm 1 depicts the pseudo-code of our customized approach.

Algorithm 1 Dragonfly Algorithm

```
Init dragonflies X_i(i = 1, 2, ..., n), Init step vectors ΔX_i(i = 1, 2, ..., n), Init max iteration Max
while current number of iterations is less than Max do
    Calculate the objective values of all dragonflies
    Update the food source and enemy
    Update w, s, a, c, f and e
    Calculate S, A, C, F and E using Eqs. (2) to (6)
    Update neighbouring radius
    if a dragonfly has at least one neighbouring dragonfly then
        Update velocity vector using Eq. (7)
        Update position vector using Eq. (8)
    else
        Update position vector using Eq. (11)
    end if
    Check and correct the new positions based on the boundaries of variables
end while
```

The overall time complexity of our approach is calculated based on the most complex steps, which in the case of DA algorithm is the neighbor checking and updating step (neighboring dragonflies and updating velocity and position vectors), resulting in quadratic time complexity of $O(n^2)$.

V. Results and Discussion

We start with a random initialization of the dragonflies position and the step vector of the dragonfly between the upper and lower limits of the solution. Then we define the calculation process of each search agent's objective values (different weights) and update them. The position updating continues until the iteration condition reaches the maximum iteration number. To respect the hard constraints, we assign a cost of 999999 to each violation of the hard constraint. After the algorithm is finished running, if the cost equals or exceeds 999999, we discard that result and perform the iteration again, as the created schedule breaks the hard constraints.

In the first iteration, we included 20 agents and ran simulations for 10000 iterations, as shown in Figure 1. The DA

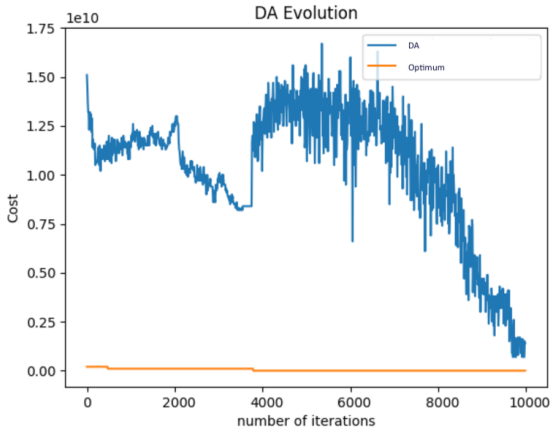

Fig. 1. First run, 20 agents

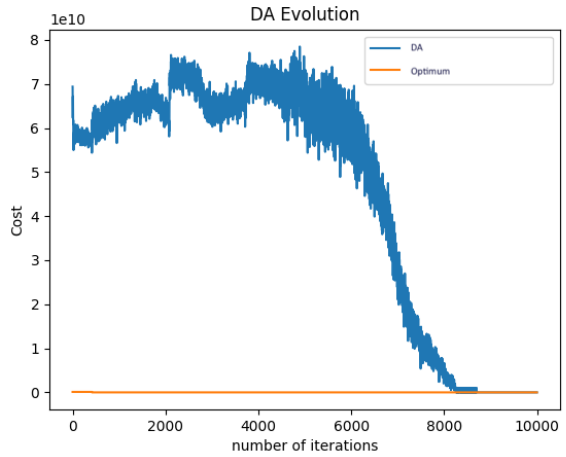

Fig. 2. Second run, 100 agents

obtained close to optimal results at the end of the simulations. In the second run, we had 100 agents and ran simulations for 10000 iterations (Figure 2). We stopped simulations at 8400 iterations, as we reached a global optimum with a cost of 0, meaning that all hard constraints were met and we did not violate any soft constraints.

We finally run different algorithms from the literature to compare results with the tested DA, as shown in Table I. We compared the DA with the Genetic Algorithm (GA), Moth Flame Optimization (MFO) and Grey Wolf Optimizer (GWO). As we can see, the DA reached the optimal result in the fastest time. Both MFO and GWO achieved the optimal results as well, while GA algorithm did not lower the cost to 0 in 10000 iterations. All simulations were performed on the AWS EC2 t3.medium instance in the cloud.

TABLE I
COMPARISON OF PERFORMANCE WITH OTHER ALGORITHMS

Algorithm	Optimal result reached?	# of iterations to reach optimal
DA	YES	8400
GA	NO	-
MFO	YES	8750
GWO	YES	9200

VI. CONCLUSION

In this paper, we investigated the nurse scheduling problem and proposed the use of the Dragonfly algorithm for optimization. The algorithm demonstrates promising capabilities in generating optimal schedules that meet the diverse constraints and requirements of nurse scheduling. By mimicking the swarming behaviours of dragonflies, the algorithm effectively balances separation, alignment, cohesion, attraction to desirable shifts, and distraction from undesirable shifts. The experimental evaluation using realistic benchmark data sets indicates that the Dragonfly algorithm can generate feasible schedules with reduced penalties compared to traditional scheduling methods. This approach contributes to enhancing the performance and quality of nursing units, benefiting both staff and patients. Future research can explore further refinements of the algorithm and its application to other scheduling problems in healthcare and beyond [15].

REFERENCES

[1] R. A. Walker and S. Chaudhuri, "Introduction to the scheduling problem," *IEEE Design & Test of Computers*, vol. 12, no. 2, pp. 60–69, 1995.

[2] A. J. Mason, D. M. Ryan, and D. M. Panton, "Integrated simulation, heuristic and optimisation approaches to staff scheduling," *Operations research*, vol. 46, no. 2, pp. 161–175, 1998.

[3] J. Schrack, R. Ortega, K. Dabu, D. Truong, M. Aibin, and A. Aibin, "Combining tabu search and genetic algorithm to determine optimal nurse schedules," in *2021 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2021, pp. 1–7.

[4] C. Huang, S. Ye, S. Shuai, M. Wei, Y. Zhou, A. Aibin, and M. Aibin, "Emergency surgical scheduling model based on moth-flame optimization algorithm," in *2023 International Conference on Computing, Networking and Communications (ICNC)*, 2023, pp. 89–94.

[5] J. L. Arthur and A. Ravindran, "A multiple objective nurse scheduling model," *AIIE transactions*, vol. 13, no. 1, pp. 55–60, 1981.

[6] T.-C. Wong, M. Xu, and K.-S. Chin, "A two-stage heuristic approach for nurse scheduling problem: A case study in an emergency department," *Computers & Operations Research*, vol. 51, pp. 99–110, 2014.

[7] Z. A. Abdalkareem, A. Amir, M. A. Al-Betar, P. Ekhan, and A. I. Hammouri, "Healthcare scheduling in optimization context: a review," *Health and Technology*, vol. 11, pp. 445–469, 2021.

[8] E. K. Burke, P. De Causmaecker, G. V. Berghe, and H. Van Landeghem, "The state of the art of nurse rostering," *Journal of scheduling*, vol. 7, pp. 441–499, 2004.

[9] S. Haspeslagh, P. De Causmaecker, A. Schaerf, and M. Stølevik, "The first international nurse rostering competition 2010," *Annals of Operations Research*, vol. 218, pp. 221–236, 2014.

[10] M. A. Awadallah, A. T. Khader, M. A. Al-Betar, and A. L. Bolaji, "Global best harmony search with a new pitch adjustment designed for nurse rostering," *Journal of King Saud University-Computer and Information Sciences*, vol. 25, no. 2, pp. 145–162, 2013.

[11] S. Petrovic and G. Vanden Berghe, "A comparison of two approaches to nurse rostering problems," *Annals of Operations Research*, vol. 194, pp. 365–384, 2012.

[12] A. Ławrynowicz, "Genetic algorithms for solving scheduling problems in manufacturing systems," *Foundations of Management*, vol. 3, no. 2, pp. 7–26, 2011.

[13] L. Trilling, A. Guinet, and D. Le Magny, "Nurse scheduling using integer linear programming and constraint programming," *IFAC Proceedings Volumes*, vol. 39, no. 3, pp. 671–676, 2006.

[14] S. Mirjalili, "Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems," *Neural computing and applications*, vol. 27, pp. 1053–1073, 2016.

[15] M. Aibin and K. Walkowiak, "Simulated annealing algorithm for optimization of elastic optical networks with unicast and anycast traffic," in *2014 16th International Conference on Transparent Optical Networks (ICTON)*, 2014, pp. 1–4.