A METAHEURISTICS-BASED ALGORITHM TO OPTIMIZE THE FATIGUE SPECTRA OF MECHANICAL PARTS

Laurent Ferhi, Airbus Helicopters, Marignane, France

Abstract

Design usage spectra for fatigue evaluation contain occurrences per hour of each maneuver but no sequential order is imposed. However, when applying fatigue cycle counting methodologies (such as rainflow) associated to Maneuver-to-Maneuver (MTM) spectrum, the order of maneuvers has a preponderant influence. To define a sequential order, mission scenarios based on operator's usage may not cover all possible flight operations and may not be in line with design spectrums found in certification guidance materials. On the other hand, sequences maximizing load amplitude (and then, fatigue damage) may be physically unrealistic and too penalizing. This paper proposes the use of Ant Colony Optimization (a metaheuristics-based algorithm) on a MTM spectrum to generate sequences respecting simultaneously the constraints of occurrence (given by the spectrum) and physical realism (given by maneuver pairwise chaining rules).

1. NOTATION

Symbols

D	distance matrix
P	pheromone matrix
n_{ants}	number of ants
n_{best}	number ants updating pheromones
n_{iter}	Number of iterations
α	distance emphasizing factor
β	pheromones emphasizing factor
δ	decay rate

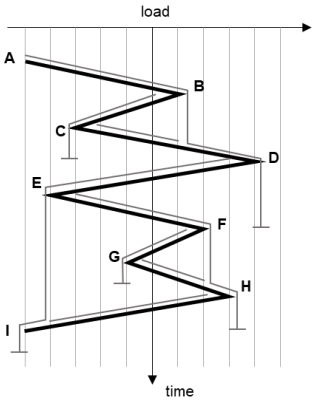
Acronyms:

ACO	Ant Colony Optimization
DUS	Design Usage Spectrum
LCF	Low Cycle Fatigue
GAG	Ground-Air-Ground
MG	Material Guidance
MTM	Maneuver-To-Maneuver
TSP	Traveling Salesman Problem

2. INTRODUCTION

Low cycle fatigue is one of the most contributing factors in helicopter mechanical part sizing. It consists in addressing the fatigue damage linked to high load amplitudes sustained during flight and due to the sequence of flight maneuvers. These fatigue cycles are traditionally reduced to GAG cycles as they generally cover the MTM cycles. However, advances in the field of mechanical parts fatigue evaluation, optimization of designs and emergence of Condition Based Maintenance concepts make the fatigue evaluation of MTM cycles increasingly important (Ref. 1.).

MTM fatigue computation is generally done thanks to a design usage spectrum (DUS) such as the one proposed in Ref. 2. It provides a set of maneuvers and associated occurrences (number per hour) but does not impose any fatigue evaluation method. It is up to the rotorcraft manufacturer to propose an exploitation method and submit it to the regulation authorities for approval. For each maneuver, a load evaluation is performed either through simulation or test. To determine a LCF fatigue damage from this load spectrum, one well-known way is to perform a fatigue cycle counting with the rainflow methodology such as described in Ref. 3. and Ref. 4. Rainflow cycle counting consists in extracting each closed loading reversals from a load history. LCF load histories can be seen as sequences of peaks and valleys (non-reversal points are removed prior to the rainflow cycle counting). The total fatigue damage can be obtained using Miner's rule for each rainflow cycle.



Rainflow cycles	
B	C
F	G
E	H
A	D

Figure 1: Rainflow fatigue cycle counting methodology illustration

However, the rainflow cycle counting methodology is history dependent, meaning that the total damage associated to a set of cycles depends on the sequential order of the maneuvers.

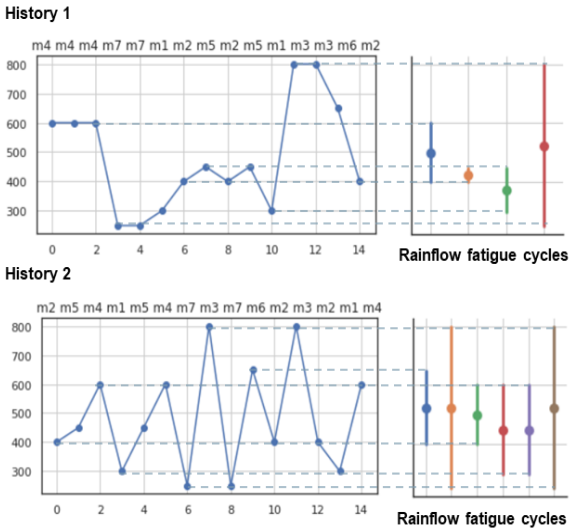


Figure 2: Two different histories from same maneuvers and occurrence spectrum

The fatigue spectrum from Ref. 2. does not provide any guidance in the sequencing of the recommended spectrum, therefore, a method for ordering the DUS must be defined when using the rainflow methodology.

3. SPECTRUM OPTIMIZATION

3.1. Traditional solutions

Several methods are appropriate to build load sequences for this purpose. One way is to define flight scenarios inspired from real operator usage. The main issue with this method is its unadaptability to determine scenarios covering all possible usages made by the operators. It is possible to build one scenario per mission type (oil and gas, emergency medical services, search and rescue etc.) but it is very difficult to ensure that operators remain in their associated scenario without a periodic and thorough verification of their usage. The case of multi-purpose rotorcrafts also addresses the problem of combining different scenarios in appropriate proportions.

Another way is to ignore the maneuvers and only consider loads. It is then possible to calculate the load sequence that maximizes load amplitudes, thus, maximizing the damage calculated by the rainflow method. This can be done by combining the maximum loads in decreasing order with the minimum loads in increasing order and distribute the occurrences to match the spectrum. This method is simple but also very conservative as it produces the most penalizing sequence in terms of loads. The obtained sequence, when translated in terms of maneuvers has no physical realism as it chains maneuvers that could not been always physically chained.

The challenge here is is to get a middle ground between a severe but unrealistic sequencing and the real flight histories performed by the operator. Rotorcraft manufacturers must also be able to ensure that the fatigue safe life limits cover the damage that any operator might have during its operations.

This paper proposes a methodology of MTM sequencing aiming at chaining the maneuvers in a physically realistic way, while fitting to the DUS occurrences.

3.2. Toward sequence optimization

The DUS can be seen as a “bag” of maneuvers that has to occur in a flight history representing thousands of flight hours. By extending the occurrences to sufficient flight hours, each occurrence can be transformed to an integer. Basically, a maneuver with an occurrence of 0.01 per hour needs an history of at least 100 hours to happen. The number of hours necessary to get each maneuver at least once depends on the lowest occurrence of the spectrum.

The problem is then to determine physically realistic histories that contain each and every maneuver from the spectrum at the specified occurrence. To do so, one must first define rules of sequencing for each

couple of maneuvers. This can be done with a pairwise chaining rules matrix giving the information: “what maneuvers can follow what maneuver”.

Building maneuver chaining rules is rather peremptory as long as it is kept simple and obvious. One would easily agree to the fact that a take-off can be followed by a hover or a climb but not by a taxiing (a landing must occur first). By keeping these rules at this level, the number of possible sequences remains huge whereas all the flights performed by actual operator are necessarily contained in the space of possible. A too detailed matrix would also make its validation more difficult.

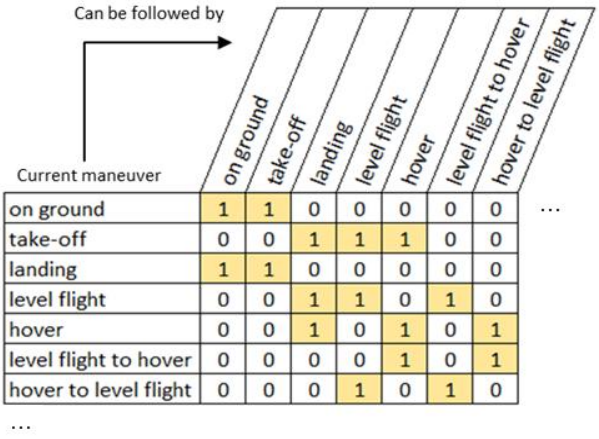


Figure 3: Extract of maneuver pairwise chaining rules matrix (chaining: 1 = possible, 0 = impossible)

A simple example of a DUS is given in Table 1 for the sake of illustration. In this example, each maneuver has an occurrence allowing it to appear in a flight history of 10 flight hours. A simple pairwise chaining rules matrix is joined on the spectrum to define the next possible maneuvers associated to each line.

Maneuver	Occurrence for 10 h	Next possible maneuvers
G	5	G, D
D	2	H, F, T, A
H	3	H, F, T, A
F	3	H, F, T
T	2	F, T
A	2	G, D

Table 1: Simple example of a design usage spectrum with pairwise chaining rules

Having this construct in mind, it is possible to design the problem as a navigation through node to node into a directed cyclic graph with 2 constraints: 1) all the nodes shall be visited the exact number of times specified by the occurrence, 2) the navigation shall respect the directions provided by the arrows.

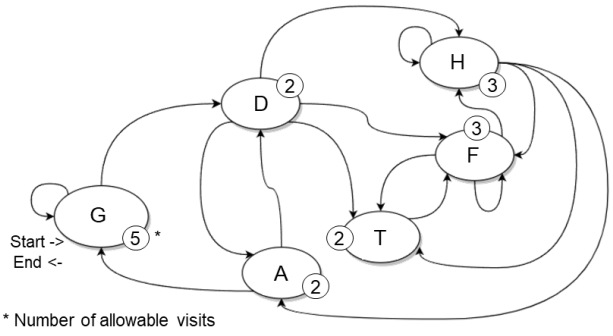


Figure 4: Example of a directed cyclic graph from Table 1 data

Depending on the number of nodes and the pairwise chaining rules, it is not always possible to find solutions that respects both constraints. Choice has been made to preserve the DUS occurrences at the expense of pairwise chaining rules in case no solution exists.

Solving this problem can be done with dynamic programming using a deterministic algorithm as long as the number of nodes remains small. However, the simple example from Figure 4. has already 2400 unique solutions. One can easily demonstrate that this problem is a variant of the so called “travelling salesman problem” (TSP) and as so, is NP-hard (it cannot be solved exactly in polynomial time). Given a collection of cities and the pairwise distance between them, the TSP aims at finding the shortest way of visiting all of the cities and returning to the starting point (Ref. 5.). The complexity to solve this problem by brute force is *O(n!)* as there are *(n-1)!* alternative routes. There is no algorithm to this day that is capable of exactly solving it in a polynomial time for large values of n.

3.3. Ant colony optimization

Heuristic methods exist to find efficient solutions to the TSP such as greedy method, genetic algorithm or swarm intelligence (Ref. 6. and Ref. 7.). Swarm intelligence combines optimization techniques based on the collective behavior of decentralized, self-organized systems using multiple agents able to interact with each other and with their environment. One of its numerous derivatives is called Ant Colony Optimiza-

tion (ACO) (Ref. 8. and Ref. 9.). ACO is a metaheuristics algorithm in which a colony of artificial ants cooperate in finding good solutions to difficult discrete optimization problems. ACO is inspired by the foraging behavior of ant colonies that travel from their nest to a food source and deposit pheromones on their way to guide other ants. In ACO, simple agents (artificial ants) interact with the environment and update it to influence the way other agents will behave. Good solutions result by this indirect communication between agents mediated by the environment.

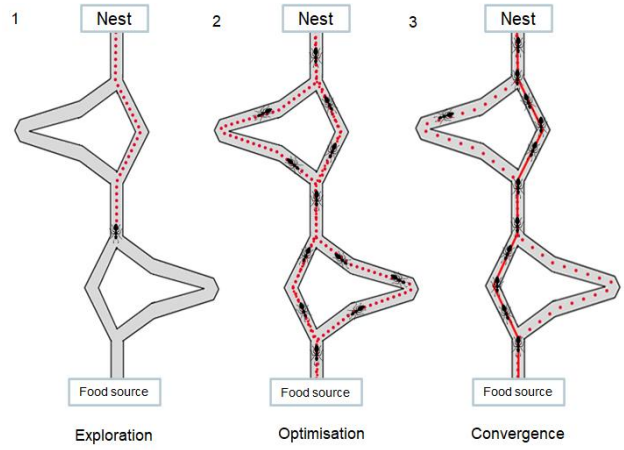


Figure 5: Ant colony optimization illustration

Many variants of this algorithm exist. The simple implementation of ACO described in the following lines represents a good trade-off between solution efficiency and required computing power.

The input of ACO is a distance matrix representing, for each couple of nodes, the distance between them. In the present problem, the distance matrix D is a *M x M* asymmetrical matrix with *M*, the number of maneuvers to distribute. Each node corresponds to a maneuver and the distance to every other node depends on the pairwise chaining rules. Unrealistic sequencing is characterized by "long" distances and realistic sequencing is characterized by "short" distances. D is then asymmetrical as some pairwise sequencing is not possible in both directions (taxiing can be followed directly by a take-off but a take-off cannot be followed directly by a taxiing). The short distances between nodes with realistic pairwise chaining can be set in different manners, for instance, inversely proportional to the load amplitude associated to the chaining of two maneuvers (maximizing the fatigue damage).

A pheromone matrix P is instantiated, containing initially equal values inversely proportional to the distance between nodes.

(1) $$pheromone_{edge_i} = \frac{1}{distance_i}$$

Starting from a specified node, n_{ants} artificial ants (class-based entity) travel through all the nodes until all the graph is visited. Each ant can store each visited node into a memory (stack type) to avoid visiting a node more times than allowed and remember the path. After all the nodes have been visited by each ant, the n_{best} ants (meaning those who discovered the best paths) deposit pheromones on the path they have travelled. P is then updated by adding new pheromones to the existing ones, still according to equation (1).

On their way forward, each ant has a stochastic node selection policy to make decisions on which node to go next. This policy is based on the distance to the next node and the quantity of pheromones deposited on the edge between current node and next node.

(2) $$Score(node_i) = pheromone_i^{\left[\alpha\left(\frac{1}{distance_i}\right)^{\beta}\right]}$$

The role α of is to control the concentration of pheromones deposited by the ants at each iteration. The role of β is to emphasize the attractiveness of next nodes. If the concentration of pheromones is very low (α close to 0), only closest nodes are likely to be selected which makes the algorithm equivalent to a classic stochastic greedy algorithm. On the other hand, too high attractiveness of nodes due to pheromones will increase the probability of a stagnation situation.

A weight is calculated as the ratio between the score of next possible nodes and the sum of all available scores.

(3) $$W(node_i) = Score\ (node_i)/\sum_i Score(node_i)$$

The effective choosing of the next node to visit is calculated using a random proportional rule weighted by the score calculated previously. The higher is the weight, the higher is the probability for the node to be chosen. This random rule ensures that nodes with a low score could still be chosen but with a low probability. Increasing the number of ants of one generation improves the exploration rate of the graph.

After P has been updated, a decay factor δ is applied to all edges. Letting pheromones decay avoids old pheromones to confuse next generations of ants. Non optimal paths on which pheromones have been applied by ants of first generations will eventually vanish as the decay rate is applied and no new ants travel through that path.

Ants of the current generation are then discarded and another generation of ants is initialized. This routine runs during n_{iter} generations or when no progress has been made since a specified number of iterations (early stopping). The iterative updating of P leads to the emergence of the best discovered paths.

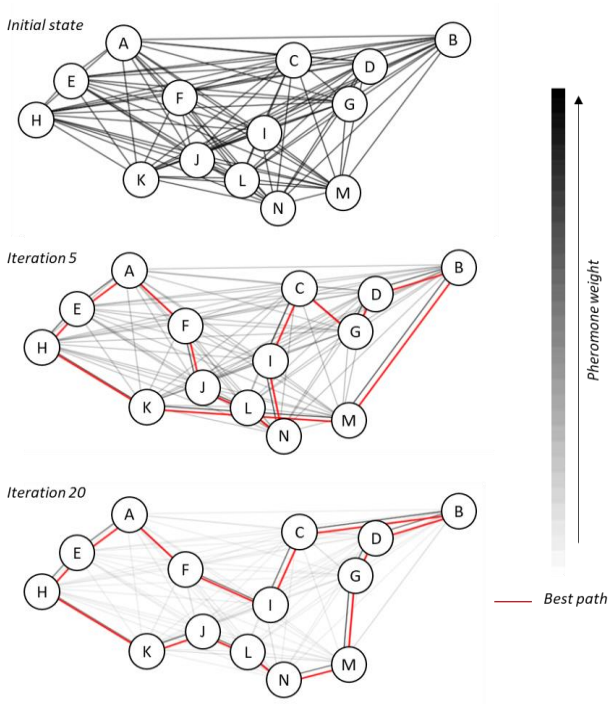


Figure 6: Pheromone matrix evolution on a simple case

The power of ACO algorithm lies in the local interactions between each independent agent and its environment. Although each agent is a simple entity with only basic characteristics (memory of the path, node selection policy and updating of environment), the algorithm is capable a complex optimization, only due to the multitude of agents involved.

A possible procedure in pseudo-code is given in Table 2. While several implementations exist, the one presented in the present paper has the advantage of being simple and efficient.

Table 2: Pseudo code for Ant Colony Optimization algorithm

	Algorithm: Ant Colony Optimization
1:	**Procedure** ACO(D,n_ants,n_best,n_iter,α,β,δ)
2:	Initialize pheromone matrix P
3:	**While** iteration < n_iter **do**
4:	Create new ants generation of n_ants
5:	**For each** ant **do**
6:	Initialize visited node set
7:	Position ant at start node
8:	**While** not all nodes are visited **do**
9:	Move ant to chosen to node based on P, α, β (equations 2 and 3)
10:	Add node to visited node set
11:	**End while**
12:	Save path and total distance
13:	**End for**
14:	**For each** n_best saved path **do**
15:	Update P based on D (equation 1)
16:	**End for**
17:	Decay pheromons based on δ
18:	**End While**
19:	**Return** path list
20:	**End-procedure**

In the present approach, the ACO implementation differs from a classical TSP on several points. First, the distance matrix in input is asymmetrical (the distance from A to B is not equal to the distance from B to A). Secondly, the nodes have to be visited a specific number of times (depending on the occurrence of the maneuver). Due to the high flexibility of the ACO algorithm, these adjustments can be implemented without degrading the algorithm performances.

3.4. Hyper-parameters tunning

In order to get the best performances from the algorithm some hyper-parameters can be tuned. The number of ants n_{ants} should be maximized in order to have the highest exploration rate but at a cost of increasing computational power required and time. The key parameters to tune are α, β and δ. A grid search has been conducted on several representative but reduced distance matrix (randomly generated) with a set of hyper-parameter values. The grid search score is a min-max scaling of the best distance path found by the algorithm (best = 1, worst = 0).

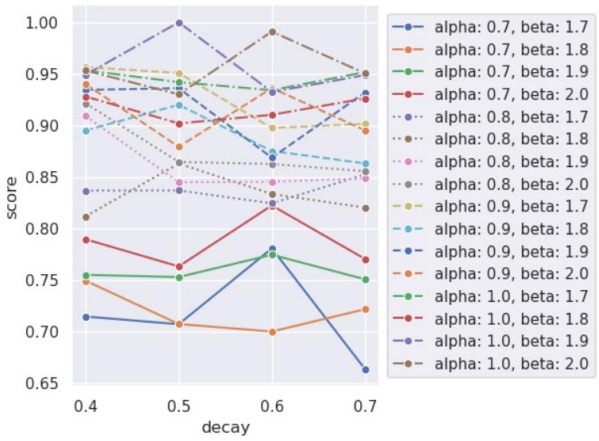


Figure 7: Extract of grid search best results for ACO hyper-parameters tuning

The grid search results for best optimization in the present definition of the problem are α = 1, β = 1.9 and δ = 0.5. This setting is close to the optimal one presented in Ref. 8.

4. TEST CASES

4.1. Distance matrix build

There are several ways to define the initial distance matrix which will lead to different kind of solutions. The simplest way is to define only two different distances, a small one for realistic connections and a high one for unrealistic connections. The gap between those two distances has to be significative to avoid misdirection of ants during the node selection steps. The ACO will then produce a large amount of flight sequences (depending on the number of ants) both physically realistic and respecting the DUS occurrences. However, when using a permissive pairwise chaining rule matrix on a high number of occurrences, the dispersion will be tremendous and the number of iterations required to cover the field of possibilities becomes huge.

Another way is to calculate distances according to the fatigue damage each pairwise chaining would produce. One way is to define the distances proportionally to the load amplitude.

$$d_{AB} = \frac{1}{\max(load_A, load_B) - \min(load_A, load_B)} \quad (4)$$

The lower the distance, the higher is the load amplitude generated by chaining node A to node B. The ACO can then find sequences among the most damaging ones, both physically realistic and respecting the DUS occurrences. As ACO is a meta-heuristic algorithm, there is no guarantee that it will find exactly the most damaging sequence (the probability decreases drastically with the number of possible solutions). However, it is almost certain that the solutions found by ACO (when correctly tuned) are among the best possible. Physically realistic sequences generated with ACO (with amplitude maximizing) generally produce less fatigue damage than sequences maximizing amplitude without account of sequencing rules. On the other hand, the ACO generated flight sequence damage per hour remains higher than the one seen by real operators (provided that the DUS is severe compared to real operator usage).

4.2. ACO sequences interpretation

When all sequences have been generated (depending on number of ants and number of iterations), it is possible to translate each sequence in terms of load history (having a correspondence between each maneuver and its corresponding load). After running a rainflow counting cycle routine, a fatigue damage can be calculated for each sequence. It is then possible to compute the distribution of damages per hour associated to the DUS and respecting a physical realism.

In order to asses the benefits of this methodology, one must demonstrate that the damages per hour obtained thanks to ACO are lower than the one obtained with the load maximizing approach (most penalizing but unrealistic sequence). It also necessary to show that this method generates sequences covering the actual usage of the operators.

4.3. Application cases

An application case on an actual H175 helicopter drive system part has been performed. The DUS that has been used derives from the twin turbine usage spectrum given in Ref. 2. The loads for each maneuver are those measured on an actual part during flight tests.

In the end, three fatigue damages per hour were calculated. 1) fatigue damage from sequences generated by ACO, 2) fatigue damage of the most penalizing sequence without taking into account of physical realism, 3) real fatigue damage estimated from in-service flight data. For the latter, a flight regime recognition algorithm was run on a large sample of flight recordings for H175 fleet. The loads have been mapped on the recognized maneuvers. Fatigue damage has

been calculated using rainflow cycle counting rational with a representative fatigue curve.

Figure 8 shows the layout of the fatigue damage per hours distribution evaluated with the three previously describe methodologies.

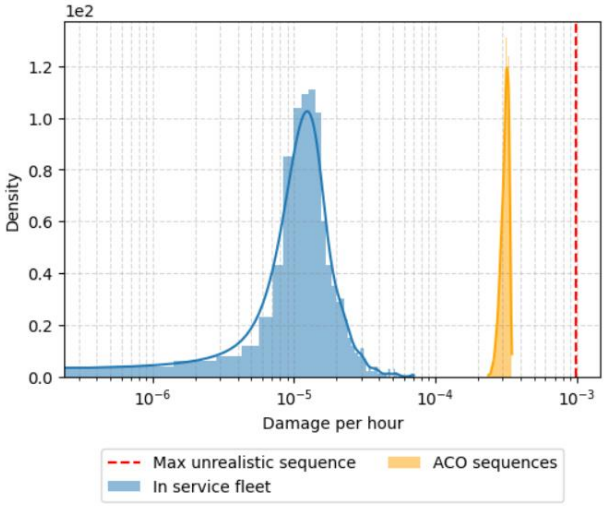


Figure 8: Distribution of damage per hour comparing actual fleet, ACO generated sequences and most penalizing unrealistic sequence

On this test case, the ACO sequences generate an average fatigue damage per hour 68% lower than the most penalizing unrealistic sequence. It is, on the other hand, 20 times higher than average estimated fleet damage. The less damaging ACO sequence generates a damage per hour 3 times higher than the most penalizing in-service flight recorded.

5. CONCLUSIONS

1. ACO provides a simple and quick optimization method to generate flight sequences both respecting a given occurrence spectrum and a physical realism.

2. This technique may help generating MTM sequences having a good trade-off between fixed scenarios (always challengeable and not covering all possible usage from the operators) and amplitude maximizing histories (physically unrealistic and too penalizing).

3. Test case shows that the ACO generated sequences remain conservative compared to real operator usage while providing substantial gains.

4. This kind of multi-agent metaheuristic technique is also adaptable to solve other optimization problems.

Author contact:

Laurent Ferhi laurent.ferhi@airbus.com

6. REFERENCES

1. Robert E. Benton, Jr., Robert J. Dudley, Jung-Hua Chang, *Maneuver-to-Maneuver Load Cycle Case Study*, US Army Aviation Engineering Directorate, Redstone Arsenal, AL

2. AC 27 MG-11, Miscellaneous Guidance, Normal Category Rotorcraft: Fatigue Evaluation of rotorcraft Structure, included in Chapter 3 of AC 27-1B, Chg. 1*, Advisory Circular: Certification of Normal Category Rotorcraft*, Federal Aviation Administration, Washington, DC, 12 Feb 03.

3. ASTM E1049-85 *standard practices for cycle counting in fatigue analysis*

4. Yung-Li Lee, Tana Tjhung,Chapter 3 - *Rainflow Cycle Counting Techniques, Metal Fatigue Analysis Handbook,* Butterworth-Heinemann, 2012, Pages 89-114

5. Sangwan, Shabnam. (2018). *Literature Review on Travelling Salesman Problem*. International Journal of Research. 5. 1152.

6. Osaba, Eneko & Yang, Xin-She & Del Ser, Javier. (2020*). Traveling salesman problem: a perspective review of recent research and new results with bio-inspired metaheuristics*. 10.1016/B978-0-12-819714-1.00020-8.

7. Deepti Chopra, Praveen Arora, *Swarm Intelligence in Data Science: Challenges, Opportunities and Applications*, Procedia Computer Science, Volume 215, 2022, Pages 104-111

8. Dorigo, M. and Stützle, T. (2004) *Ant Colony Optimization*. The MIT Press, Cambridge, 305 p.

9. Pintea, C-M & Pop, Petrica & Dumitrescu, Dan. (2007). *An ant-based technique for the dynamic generalized traveling salesman problem*.