Project Literature Review

1 Introduction

A significant amount of research has been put into the efficient solving of combinatorial optimisation problems. However, given the fact that many of these problems are NP-hard, it is typically not feasible to employ optimal algorithms. Instead, it is common to employ approximate metaheuristic algorithms to achieve good solutions efficiently [3]. Many such algorithms have been suggested, with a common approach in recent years being population-based nature-inspired metaheuristic algorithms (PSO, ACO, GA etc.) two such algorithms are Cuckoo Search proposed by Yang et al. in 2009 [15] and Cuckoo Optimisation proposed by Rajabioun in 2011 [14], both inspired by the breeding patterns of cuckoo birds. Cuckoos exhibit an aggressive breeding strategy known as brood parasitism whereby a mother cuckoo lays her eggs in the nests of other host birds. Cuckoos are incentivised to mimic the appearance of the host species' egg as if the cuckoo's egg is spotted by the host bird it will be thrown out of the nest [13].

One such NP-hard combinatorial optimisation problem is the graph colouring problem: Given an undirected graph $G=(V,E)$, a colouring is a function $C:V\to\mathbb{N}$. A colouring is valid if for any two adjacent vertices u and v, $C(u)\neq C(v)$. The goal with the general graph colouring problem is to find a valid colouring for a graph with the fewest possible colours. One common variante is the k-colouring problem, where some positive integer k is fixed before hand and the goal of the problem is simply to find a valid colouring with at most k colours (such a colouring is called a k-colouring of G).

Both Cuckoo Search and Cuckoo Optimisation are optimisation algorithms originally configured for the continuous domain (where a solution is a real-numbered vector that minimises some objective function f), this review will discuss these algorithms in detail and investigate the research surrounding discretising them for the graph colouring problem.

2 Cuckoo Search

2.1 Continuous Algorithm and Lévy Flights

In the continuous optimisation algorithm Cuckoo Search, first proposed by Yang et al. [15], an initial population of host nests (where each nest represents a solution) is constructed randomly. These solutions are then iteratively replaced and improved in a process that mimics the brood parasitism described above.

1. Each cuckoo lays one egg at a time, and dumps its egg in a randomly chosen nest.
2. The best nests with the highest quality of eggs will carry over to the next generation.
3. The number of available host nests is fixed, and the egg laid by a cuckoo is discovered by the host bird with probability $p_a\in[0,1]$.

For simplicity, the last rule is typically approximated by a fraction p_a of the worst nests being replaced by new random solutions.

When cuckoos replace a nest with a new solution, a Lévy flight is used. A Lévy flight is a random walk where the step-lengths have a Lévy distribution. Lévy distributions are heavy-tailed meaning that it is fairly common for a large step to be made. Research into Lévy flights shows that many animals explore their landscape with Lévy-flight-like random walks and that they are an efficient method for stochastically exploring a space.

When generating a new solution $x^{(t+1)}$ for a nest $x^{(t)}$, a single step of a Lévy flight is performed:

$$x_i^{(t+1)}=x_i^{(t)}+\alpha L\acute{e}vy(\beta) \tag{1}$$

Where $\alpha > 0$ is the step size which should be related to the scale of the problem of interest. *Lévy*(β) denotes the result of generating Lévy noise. Tethis et al. [10] conducted a comparison of three Lévy noise generation algorithms and found Mantegna's [12] to be the most efficient and accurate:

$$\textit{Lévy}(\beta) = \frac{p}{|q|^{\frac{1}{\beta}}} \tag{2}$$

Where $0.3 \leq \beta < 2$ is the scale parameter, p and q are normally distributed random variables

$$p \sim N(0, \sigma_p^2), \qquad q \sim N(0, \sigma_q^2), \qquad \sigma_p = \left(\frac{\Gamma(1+\beta) sin(\frac{\pi\beta}{2})}{\Gamma(\frac{1+\beta}{2})\beta 2^{(\beta-1)/2}} \right)^{\frac{1}{\beta}}, \qquad \sigma_q = 1 \tag{3}$$

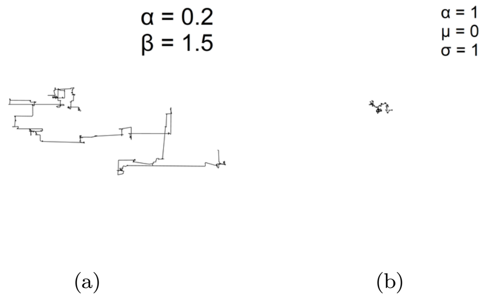

Figure 1: 1000 iterations of a Lévy flight (a) vs. a normally distributed random walk (b)

At each step of the algorithm, a new solution is generated for each nest via the method described in eq. (1). If the new solution's fitness is better than the current one's the current solution is replaced. Next, p_a of the worst nests are replaced with completely new solutions. This process repeats for a set number of iterations or until some stop criterion is met at which point the best solution in the current generation is returned. The full pseudo code for the continuous algorithm is shown in algorithm 1.

Algorithm 1 Continuous Cuckoo Search

```
Generate the initial population of cuckoos
while ($t < numGenerations$) and not (stop criterion) do
    for each nest $i$ do
        Generate a new solution $j$ via a Lévy flight
        if $f(j) < f(i)$ then
            Replace $i$ with $j$
        end if
    end for
    Replace $p_a$ of the worst nests
end while
```

2.2 Binary Cuckoo Search for Graph Colouring

In 2014, Djelloul et al. [6] proposed a discretisation formulating the k-colouring problem as a binary optimisation problem and constraining solution domains using the sigmoid function. The main challenges when discretising cuckoo search are the creation of new cuckoos and discrete Lévy flights. To solve this, Djelloul et al. adapted their binary approach from Gherboudj et al. [8].

Each cuckoo is a (possibly invalid) k-colouring of the graph and the fitness function to minimise is the number of conflicting edges.

$$f(x) = \sum_{uv \in E} \delta_{uv}, \qquad \delta_{uv} = \begin{cases} 1 & C(u) = C(v) \\ 0 & otherwise \end{cases} \tag{4}$$

In order to represent a k-colouring in the binary domain, a kxn binary matrix is used where a presence of 1 in position (i,j) indicates that the node j is coloured with the colour i. Thus, in each column there is a single 1.

To get a new solution $x^{(t+1)}$ from a solution $x^{(t)}$, a continuous Lévy flight is performed on a matrix to generate a real valued kxn matrix A. A matrix $S(A)$ of "flipping chances" is constructed by applying the sigmoid function to each element in A.

$$S(A)_{ij} = \frac{1}{1+e^{-A_{ij}}} \tag{5}$$

Next, a matrix $r(k,n)$ is constructed where each element is chosen uniformly at random from the interval [0,1]. Finally, the following algorithm is performed to construct $x^{(t+1)}$:

Algorithm 2 Binary Solution Representation (BSR) Algorithm

```
for i = 1 to k do
  for j = 1 to n do
    if $S(A)_{ij} > r_{ij}$ and any neighbour of $j$ is coloured $i$ in $x^{(t)}$ then
      $x_{ij}^{(t+1)} = 1$
    else
      $x^{(t+1)} = 0$
    end if
  end for
end for
Colour any uncoloured nodes uniformly at random
```

For efficiency, there is no need to generate the matrix $r(k,n)$, a new random variable can simply be generated each time a flipping check is made. To generate the initial population and to replace any cuckoos discovered by host birds (the fraction p_a of worst cuckoos), the Recursive Largest First (RLF) greedy algorithm is used. A random node v from the set Y of uncoloured nodes (initially, $Y = V$), is coloured Ncl (initially, $Ncl = 1$). After this, v and $N(v)$ (the set of nodes adjacent to v) are removed from Y. This is repeated until $Y = \emptyset$, at which point Ncl is incremented. The entire process repeats until $Ncl > k$, at which point any remaining uncoloured nodes are coloured randomly.

Algorithm 3 Incomplete greedy algorithm for generating new cuckoos

```
$Ncl \leftarrow 1$
while $Ncl \leq k$ do
  $Y \leftarrow \{v | v \in V$, v is uncoloured$\}$
  while $Y \neq \emptyset$ do
    Choose a random vertex $v \in Y$ and colour it $Ncl$
    $Y \leftarrow Y \setminus (\{v\} \cup N(v))$
  end while
  $Ncl \leftarrow Ncl + 1$
end while
Colour remaining nodes randomly
```

Once the Lévy flight and cuckoo generation operators are discretised, the overall algorithm functions the same as the continuous version (Algorithm 1). Although the algorithm does provide an effective discretisation for cuckoo search when applied to the k-colouring problem and the algorithm achieved moderate experimental success when compared to other simple metaheuristic algorithms, the binary formulation of k-colouring seems unnecessarily complex and the algorithm's use of the sigmoid function to discretise Lévy flights has been criticised. For solving the general graph colouring problem, they suggest beginning with $k = |V|$ and when a feasible k-colouring is found, k is reduced by one. This process repeats until no valid k-colouring is found.

2.3 An Alternative Approach to Graph Colouring with Cuckoo Search

In 2017, Aranha et al. [2] proposed a novel method for applying cuckoo search to the k-colouring problem. They criticised Djelloul et al.'s use of the sigmoid function [6], stating that it didn't take the concept of solution distance (which they felt was important to the functioning of Lévy flights).

In their paper, Aranha et al. used a slightly modified version of the base algorithm with more Lévy flights and a modified parasitism operator. In each iteration, up to two replacement candidates are generated for each individual x, a solution u_1 is created via Lévy flights and with probability p_a, a second individual u_2 is generated (they propose three methods of computing u_2 and found Lévy flights to be the most effective) and the best among x, u_1, and u_2 (if u_2 was generated) is added to the next generation.

2.3.1 The Propopsed Method

A candidate solution x is a vector where $x \in \{1,...,k\}^{|V|}$. Each entry of x encodes the colour of a vertex (i.e. the colour of node i is the value of x_i). The fitness function to be minimised is the number of conflicting edges, already defined in equation (4).

To generate a new solution $x^{(t+1)}$ via a Lévy flight from $x^{(t)}$, a random number $M = \lfloor \alpha L\acute{e}vy(\beta) \rfloor + 1$ (where $\alpha > 0$ is the step-size) is generated and M random elements from $x^{(t)}$ are replaced uniformly at random. The random number M encodes the distance between solutions $x^{(t)}$ and $x^{(t+1)}$.

Algorithm 4 Aranha et al.'s proposed cuckoo search algorithm for k-colouring

```
while current evaluations < max evaluations do
  for each candidate solution $x_i^{(t)} \in S^{(t)}$ do
    $M \leftarrow \lfloor \alpha L\acute{e}vy(\beta) \rfloor + 1$
    Generate $u_1$ by randomly changing the colour of $M$ vertices in $x_i^{(t)}$
    if $f(u_1) \leq f(x_i^{(t)}$ then
      Replace $x_i^{(t)}$ with $u_1$
    end if
    if random uniform number $k \in (0,1) \leq p_a$ then
      Select $M$ using one of{uniform distribution, discrete Lévy flight, fixed value}
      Generate $u_2$ by randomly changing the colour of M vertices in $x_i^{(t)}$
      if (not parasitism_comparison) or ($f(u_2) \leq f(x_i^{(t)})$) then
        Replace $x_i^{(t)}$ with $u_2$
      end if
    end if
  end for
end while
```

They note that due to the number of local optima in the graph colouring search space, the parasitism comparison $f(u_2) \leq f(x_i^{(t)})$ rarely succeeds. To fix this, the parasitism comparison may be removed (hence the 'parasitism_comparison' boolean).

When evaluated, their algorithm achieved good results but it was only tested against a basic genetic algorithm so these results don't necessarily hold much weight.

3 Cuckoo Optimisation

3.1 Continuous Algorithm

The continuous Cuckoo Optimisation Algorithm (COA), proposed in 2011 by Rajabioun [14] functions quite differently from cuckoo search.

In the algorithm, each cuckoo resides at a habitat from some region R, where R is defined as $[var_{lo}, var_{hi}]^n$, for some $n \geq 1$, where var_{lo} (resp. var_{hi}) is the minimial (resp. *maximal*) value for some component. To begin the algorithm, a candidate habitat matrix of size $N_{pop} \times N_{var}$ is generated at random, each row of this habitat matrix is a habitat at which a cuckoo resides.

To begin each iteration, each cuckoo lays between 5 to 20 eggs (this number is chosen uniformly at random for each cuckoo). Each is a random point within the cuckoo's Egg Laying Radius (ELR). The ELR for cuckoo i is calculated like so:

$$ELR_i = \alpha \times \frac{\text{Number of eggs laid by i}}{\text{Number of eggs laid by all cuckoos}} \times (var_{hi} - var_{lo}) \tag{6}$$

Where α is an integer used to handle the maximum ELR. After each cuckoo lays their eggs, a fraction $p \in (0,1)$ of the worst eggs (eggs with less fitness) are killed. The remaining eggs mature into fully grown cuckoos and are added to the population. In order to control the maximum number of cuckoos a variable N_{max} is created and the worst cuckoos are killed until there are less than N_{max} cuckoos in the population.

The final stage in cuckoo breeding is migration where the now mature cuckoos move towards better habitats before laying their own eggs and starting the cycle again. To replicate this, the cuckoos are clustered using k-means $(3 \leq k \leq 5)$ and a mean fitness value for each cluster is calculated and the "goal point" for cuckoo migration is set to the mean of the fittest cluster.

However, cuckoos don't migrate all the way to the goal point, they fly to some intermediate habitat. To calculate this new point, two methods are typically used. The first (proposed by Rajabioun [14]) is to generate two random numbers λ and ϕ.

$$\lambda \sim U(0,1), \qquad \phi \sim U(-\omega,\omega) \tag{7}$$

Where $x \sim U(a,b)$ states that x is a uniformly distributed random variable from the interval (a,b) and ω is a constant (typically $\omega = \pi/6$). In this formulation, cuckoos travel distance $d \times \lambda$ from their original habitat (where d is the distance from their habitat and the goal point) with a deviation of ϕ radians. It is unclear how this method generalises for dimensions greater than 2, so the following formulation is typically used [5] [11]:

$$H_{i,j}^{new} = H_{i,j}^{old} + F \times r \times (H_{best,j} - H_{i,j}^{old}) \tag{8}$$

Where $r \sim U(0,1)$ is a random number, F is a constant, $H_{i,j}$ refers to the j^{th} entry of the i^{th} habitat, H_{best} refers to the best habitat (the goal point).

The algorithm has achieved impressive experimental results in the continuous domain, finding optimal solutions to benchmark functions in less than 10 iterations (whereas it takes algorithms like PSO or GA upwards of 50).

3.2 Adaptive Cuckoo Optimisation

In 2018, Bovieiri et al. [5] proposed ACOA, an adaptive variation of the continuous cuckoo optimisation algorithm proposed in [14].

In their algorithm, they used the second approach to cuckoo migration described in equation (8). They also added a new parameter ε, after the egg laying phase eggs are compared and if any two are within distance ε of each other one is killed. Although the original paper [14] mentions that in nature only one cuckoo survives per nest, no functionality of this nature is described.

Bovieri et al.'s [5] algorithm functions by adapting the values of the egg-laying coefficient ($alpha$), the ε parameter, and the migration coefficient (F) depending on the current state of the algorithm.

They note that a higher $alpha$ allows global exploration of the search space, whereas a lower value allows for intensification through local search. Thus, they suggest decreasing $alpha$ non-linearly over time, via the equation:

$$\alpha_t = \alpha_{max} - \frac{\alpha_{max} - \alpha_{min}}{t_{max} - t + 1} \tag{9}$$

Where α_{max} (resp. α_{min})) is the maximum (resp. minimum) possible value for α, t_{max} is the total number of iterations (before the algorithm terminates). For simplicity, α_{min} may be assumed to be 0.

With a similar justification for the adaptation of α, ε is decreased non-linearly like so:

$$\varepsilon_t = \varepsilon_{max} - t \times \frac{\varepsilon_{max} - \varepsilon_{min}}{t_{max}} \tag{10}$$

In their migration function, Bovieri et al. use a different F value for each cuckoo. Farther cuckoos from the goal point will be given a higher F and closer ones a lower F value:

$$F_i = F_{min} + \frac{f_i - f_{min}}{f_{max} - f_{min}} \times (F_{max} - F_{min}) \tag{11}$$

Where F_i is the cuckoo i's F value, F_{max} (resp. F_{min}) is the maximum (resp. minimum) allowable value for F, f_i is cuckoo i's fitness, f_{max} (resp. f_{min}) is the highest (resp. lowest) fitness value from among the current generation of cuckoos. The intuition behind the adaption of F in this way is on the one hand to allow farther cuckoos that may be stuck in local optima to perform large jumps in order to escape, and on the other hand to allow cuckoos closer to the goal point to perform smaller jumps to aid local search and improve accuracy, as well as to avoid premature convergence.

On average, the new adaptive algorithm performed 36% better on unimodal functions and 13% better on multimodal functions [5].

3.3 Modified Cuckoo Search for Graph Colouring

In 2015, Mahmoudi et al. [11] proposed a modified cuckoo optimisation algorithm (MCOA) to solve the graph colouring problem. Unlike the methods previously described for cuckoo search, MCOA was developed to solve the general graph colouring probem (as opposed to the k-colouring problem) where there is no fixed k value and the goal is not only to find a valid colouring but also to find a valid colouring with the fewest colours possible. Although the algorithm seems sufficient and well realised, there are some issues that will be discussed later.

To generate the initial population, the Colouring Vertex function is used:

Algorithm 5 Colouring Vertex
Input: A partial colouring C, and a vertex v

```
adjacentNodeSet $\leftarrow N(v)$
adjacentNodesColours $\leftarrow \bigcup_{u \in adjacentNodeSet} \{C(u)\}$
$N \leftarrow |V|$
for $j$ = 1 to $N$ do
  if $j \notin adjacentNodesColours$ then
    $C(v) \leftarrow j$
    return
  end if
end for
```

This function takes a partial colouring C and colours a vertex v with the smallest valid colour. To generate a new cuckoo: First, a node is randomly selected and coloured by the Colouring Vertex function. Then, all other nodes are sorted in non-increasing order by degree and coloured iteratively using the Colouring Vertex function. Repeating this N_{pop} times will create a population of valid colourings.

MCOA's egg laying algorithm is heavily inspired by the neighbourhood search algorithm is used in BEECOL proposed by Faraji et al. [7]. This algorithm relies on several new properties being defined for vertices:

Definition 3.1.

$$Hdeg(v) = \sum_{u \in N(v)} deg(u) \tag{12}$$

$Hdeg(v)$ is the sum of degrees of adjacent nodes of v.

Definition 3.2. Let $\{v_1, ..., v_{D_T}\}$ be nodes from a colouring S that all share the same colour T, then:

$$Q_{S_T} = \sum_{i=1}^{D_T} Hdeg(v_i) \tag{13}$$

Q_{S_T} is the sum of $Hdeg(v)$ for all vertices v which share the colour T in colouring S.

Definition 3.3. If N is the number of nodes in a graph and $k(S)$ is the number of colours used by S, then:

$$F_S = \frac{N/k(S)}{3} \tag{14}$$

Definition 3.4. If S is a colouring, then:

$$X_{S_i} = \begin{cases} Q_{S_i} - Q_{S_{i+1}} & 0 < i < k(S) \\ Q_{S_i} - Q_{S_1} & i = k(S) \end{cases} \tag{15}$$

Finally, the egglaying algorithm can be described with the pseudocode:

Algorithm 6 MCOA Egg Laying Algorithm
Input: mother cuckoo S

```
$k \leftarrow k(S)$
$tmpColouring \leftarrow \{|V|\}^{|V|}$ (an array of length $|V|$ where all entries are $|V|$)
Create an array $nodescolour$ s.t. $nodesColour[i]$ is an array of all nodes coloured $i$ in $S$
Create an array $nodesPerColour$ s.t. $nodesPerColour[i]$ is the number of nodes coloured $i$ in $S$
Create an array $X$ of all colours and sort it non-decreasingly by $X_S$
Calculate and store $F_S$
$b_S \leftarrow \emptyset$
while any of $tmpColouring = |V|$ do
  Let $num[1..k]$ be a random permutation of $(1,...,k)$
  $i \leftarrow X[num[1]]$
  $Step(i-1)$
  $Step(i+1)$
  $Step(i)$
end while
```

Where $Step(x)$ is an inline function defined like so:

Algorithm 7 Step(x)

```
if $i > 1$ and $i < k$ and $nodesPerColour[i] > F_S$ and $a \notin b_S$ then
  for node in $nodesPerColour[x]$ do
    Apply Vertex Colouring function to node in tmpColouring
  end for
  $b_S \leftarrow b_S \cup \{x\}$
end if
```

MCOA's fitness function to be minimised is defined as follows: if there is no edge between two nodes i and j and the proposed colours for these two nodes are not the same, $Penalty_S$ is increased by one.

$$f(S) = P_1 \times Penalty_1 + P_2 \times k(S) \tag{16}$$

Where P_1 and P_2 are user-defined weights.

MCOA uses the a discrete version of the migration function used in ACOA (equation (8)). However, this equation must be discretised to be applied to the graph colouring problem. To do this, the search space must be turned into a metric space. A search space is a tuple (S,O) where S is the set of all possible solutions and O is the set of operations that can be applied to solutions such that S is closed under O. A metric space is a tupl (M,d) where M is a set and $d: M \times M \to \mathbb{R}$ is distance function s.t., forall $x,y,z \in M$:

1. $d(x,y) = 0 \iff x = y$
2. $d(x,y) = d(y,x)$
3. $d(x,z) \leq d(x,y) + d(y,z)$

The most obvious way of converting a search space (S,O) into a metric space (M,d) is to set $M = S$ and $d(x,y)$ to the fewest number of operations $o \in O$ to convert x into y. However, this is easier said than done when dealing with an NP-hard problem such as graph colouring.

Mahmoudi et al. suggest defining a single operation that changes a single vertex's colour. The distance $d(x,y)$ between x and y can thus be easily computed as the number of positions in which x and y differ and x can be transformed into y just as easily by iteratively replacing $x's$ colours in these positions with $y's$. However, the set S is not closed under these operations which poses a significant problem later on:

When cuckoos migrate, they don't migrate all the way to the goal point instead they migrate to some intermediate point. To replicate this, Mahmoudi et al. generate a random number r and, given a list $M_{x \to y}$ of operations required to transform a solution x into a solution y, simply perform the first $\rceil F \times r \times |M| \rceil$ of these operations on

x. But, as stated earlier, S is not closed under these operations so the solution generated by this partial migration may not (and in fact probably won't) be a valid colouring meaning the entire algorithm

Beyond the migration function's production of invalid colourings, several other issues can be found with this paper and algorithm: Their egg laying algorithm is unnecessarily complex and involved, to the point that it obscures the justification for the function. Furthermore, although they claim the egg laying algorithm performs a neighbourhood search, they never fully defined their neighbourhood structure or justified this claim (by showing that the egg laying algorithm produces a solution within a certain distance of another).

3.4 A Potential Complete Discretisation of Cuckoo Optimisation for Graph Colouring

A lot of research has been done into the exploration of the graph colouring search space. Given a graph G, we can construct a "reconfiguration" graph $C_k(G)$ where each vertex represents a proper k-colouring of G and two vertices are connected if one colouring can be transformed into the other by changing the colour of exactly one vertex.

In 2009, Bonsma et al. [4] proved that, given a graph G and two proper k-colourings α and β the problem of determining whether there is a path between α and β in $C_k(G)$ is PSPACE-complete for $k \geq 4$. In particular, they showed that for every $k \geq 4$, a class of graphs $\{G_{N,k} | N \in \mathbb{N}\}$ exists such that each $G_{N,k}$ is of size $O(N^2)$ and there exists two k-colourings such that the distance between them in $C_k(G_{N,k})$ is $\Omega(2^N)$.

In fact, Bonsma et al. also showed that a graph G's reconfiguration graph $C_k(G)$ is not always connected meaning that with a naive approach of exploring the reconfiguration graph (such as starting in a random position and performing a breadth-first search) may be doomed to never find an optimal colouring.

These discoveries make the problem of exploring the graph colouring space much more difficult than Mahmoudi et al. [11] anticipated. For my project, one of my more advanced goals is to construct a full discretisation of cuckoo optimisation for the general graph colouring problem (where k is not fixed). This will require more research and experimentation surrounding graph colouring reconfiguration graphs.

References

[1] Mohamed Abdel-Basset, Abdel-Naser Hessin, and Lila Abdel-Fatah. "A comprehensive study of cuckoo-inspired algorithms". In: *Neural Computing and Applications* 29.2 (2018), pp. 345–361.

[2] Claus Aranha, Keita Toda, and Hitoshi Kanoh. "Solving the Graph Coloring Problem using Cuckoo Search". In: *International Conference on Swarm Intelligence*. Springer. 2017, pp. 552–560.

[3] Christian Blum and Andrea Roli. "Metaheuristics in combinatorial optimization: Overview and conceptual comparison". In: *ACM computing surveys (CSUR)* 35.3 (2003), pp. 268–308.

[4] Paul Bonsma and Luis Cereceda. "Finding paths between graph colourings: PSPACE-completeness and superpolynomial distances". In: *Theoretical Computer Science* 410.50 (2009), pp. 5215–5226.

[5] Hamid Reza Boveiri and Mohamed Elhoseny. "A-COA: an adaptive cuckoo optimization algorithm for continuous and combinatorial optimization". In: *Neural Computing and Applications* 32.3 (2020), pp. 681–705.

[6] Halima Djelloul, Abdesslem Layeb, and Salim Chikhi. "A binary cuckoo search algorithm for graph coloring problem". In: *International Journal of Applied Evolutionary Computation (IJAEC)* 5.3 (2014), pp. 42–56.

[7] Majid Faraji and H Haj Seyyed Javadi. "Proposing a new algorithm based on bees behavior for solving graph coloring". In: *International Journal of Contemporary Mathematical Sciences* 6.1 (2011), pp. 41–49.

[8] Amira Gherboudj, Abdesslem Layeb, and Salim Chikhi. "Solving 0-1 knapsack problems by a discrete binary version of cuckoo search algorithm". In: *International Journal of Bio-Inspired Computation* 4.4 (2012), pp. 229–236.

[9] Alain Hertz and Dominique de Werra. "Using tabu search techniques for graph coloring". In: *Computing* 39.4 (1987), pp. 345–351.

[10] Matteo Leccardi. "Comparison of three algorithms for Levy noise generation". In: *Proceedings of fifth EUROMECH nonlinear dynamics conference*. 2005.

[11] Shadi Mahmoudi and Shahriar Lotfi. "Modified cuckoo optimization algorithm (MCOA) to solve graph coloring problem". In: *Applied soft computing* 33 (2015), pp. 48–64.

[12] Rosario Nunzio Mantegna. "Fast, accurate algorithm for numerical simulation of Levy stable stochastic processes". In: *Physical Review E* 49.5

(1994), p. 4677.

[13] Robert B Payne and Michael D Sorensen. *The cuckoos*. Vol. 15. Oxford University Press, 2005.

[14] Ramin Rajabioun. “Cuckoo optimization algorithm”. In: *Applied soft computing* 11.8 (2011), pp. 5508–5518.

[15] Xin-She Yang and Suash Deb. “Cuckoo search via Lévy flights”. In: *2009 World congress on nature & biologically inspired computing (NaBIC)*. IEEE. 2009, pp. 210–214.

[16] Yongquan Zhou et al. “An improved cuckoo search algorithm for solving planar graph coloring problem”. In: *Applied Mathematics & Information Sciences* 7.2 (2013), p. 785.