J Intell Manuf (2016) 27:417–429
DOI 10.1007/s10845-014-0873-z

A hybrid cuckoo search-genetic algorithm for hole-making sequence optimization

W. C. E. Lim · G. Kanagaraj · S. G. Ponnambalam

Received: 20 August 2013 / Accepted: 13 January 2014 / Published online: 23 January 2014

Abstract Biologically-inspired algorithms are stochastic search methods that emulate the behavior of natural biological evolution to produce better solutions and have been widely used to solve engineering optimization problems. In this paper, a new hybrid algorithm is proposed based on the breeding behavior of cuckoos and evolutionary strategies of genetic algorithm by combining the advantages of genetic algorithm into the cuckoo search algorithm. The proposed hybrid cuckoo search-genetic algorithm (CSGA) is used for the optimization of hole-making operations in which a hole may require various tools to machine its final size. The main objective considered here is to minimize the total non-cutting time of the machining process, including the tool positioning time and the tool switching time. The performance of CSGA is verified through solving a set of benchmark problems taken from the literature. The amount of improvement obtained for different problem sizes are reported and compared with those by ant colony optimization, particle swarm optimization, immune based algorithm and cuckoo search algorithm. The results of the tests show that CSGA is superior to the compared algorithms.

Keywords Cuckoo search · Genetic algorithm · Hybrid algorithm · Hole-making sequence optimization

W. C. E. Lim · S. G. Ponnambalam (✉)
Advanced Engineering Platform and School of Engineering, Monash University Malaysia, Jalan Lagoon Selatan, 46150 Bandar Sunway, Selangor, Malaysia
e-mail: sgponnambalam@monash.edu

W. C. E. Lim
e-mail: esmondelwc@gmail.com

G. Kanagaraj
Department of Mechanical Engineering, Thiagarajar College of Engineering, Madurai 625015, India
e-mail: gkmech@tce.edu

Introduction

Machining round holes with varies sizes in metal stock is one of the most common operations in the manufacturing industry. Hole-making operations such as drilling, reaming, tapping, and punching compose a large portion of machining processes for many industrial parts such as dies, plastic injection mold manufacturing and digital T/R modules for radar, where the machining process of a hole consists of several individual operations with different machining tools. In printed circuit board drilling, only one tool and one operation is needed to drill each hole. However, with many other industrial parts like a plastic injection mould, it may have many holes of various diameters, surface finishes, tolerances, and possibly different depths; in such cases, several tools with different diameters may be needed to finish one hole. A hole may or may not be completed depending on the hole diameter, tool geometry, and surface quality specifications. A relatively large hole may not be able to be finished with a single tool. A pilot hole may have to be drilled first using a tool of smaller diameter and enlarge subsequently using larger tools to its final size. Where necessary, additional cutting tools may be needed for reaming or tapping.

In drilling different holes specifications, the common practice in the industry is to complete all operations on the holes that require the current tool before switching to the next tool. This practice is common because the tool switching time is often longer than the tool positioning time. The path of each individual tool is commonly solved as separate travelling salesman problem (TSP) and the tours are executed one after another with tool switching between each tour. However, the repeated movement of the worktable to drill all the holes that requires the current tool is not optimal. On the other hand, the movement of the worktable can be minimized by completing a hole using several tools of different diameters before

moving to the next hole. This in turn would lead to excessive tool switches.

According to the research of Merchant (1985), the non-cutting time take on average 70 % of the total time in the manufacturing process. Therefore, efforts in planning the holes machining sequence in order to shorten the tool positioning time and the tool switching time in a machining process is of key issues. This can lead to significant reduction in machining time and increase the efficiency of hole-making process, which directly improves productivity of manufacturing system. As a result, the problem of finding the best sequence for the drilling operation has drawn attention of numerous researchers.

Many works have been reported for minimizing the productive time by optimizing the cutting parameters (Kolahan and Liang 2000; Dereli et al. 2001), type of tool paths to be involved (Yao and Gupta 2004), tool selection and tool sequencing (Zhang and Ge 2009), however, very limited number of researchers have considered non-productive time. Literatures on drilling route optimization problem can be traced back to the year 1996 where Kolahan and Liang (1996) presented a case study with variable holes sizes. Kolahan and Liang (1996) also applied tabu search approach to minimize the cost in hole making processes by optimizing tool travel scheduling, tool switch scheduling, tool selection, and machining speed specification. The algorithm was tested for the job size of 50, 100, 150, and 200 nodes with stopping limit of 600*s* and showed good results especially for the large size jobs.

Ghaiebi and Solimanpur (2007) dealt with the optimization of hole-making operations in conditions where a hole may need several tools to get completed. The objective was to minimize the summation of tool airtime and tool switch time. The objective was affected by the sequence through which each operation of each hole was done. The problem was formulated as a 0–1 non-linear mathematical model. An ant algorithm was developed to solve the proposed mathematical model. An illustrative example showed the application of the algorithm to optimize the sequence of hole-making operations in a typical industrial part. The author's assumption was that a hole is made in multiple passes each of which need a particular tool and the machining process can be started from any point. A minimum path search algorithm using genetic algorithm (GA) was developed by Oysu and Bingul (2007) for the tool-path optimization. The algorithm successfully optimized the number of retraction points together with total airtime.

Most of the researcher's formulated hole-making problem is similar to that of travelling salesman problem (TSP) in which each city (hole) in a tour (sequence of operations) must be visited only once. Liu et al. (2013) formulated the mathematical model for process planning problem by considering the selection of machining resources, operations sequence optimization and the manufacturing constraints, mapping them to a weighted graph and converted it to a constraint-based travelling salesman problem. The operation sets for each manufacturing features are mapped to city groups, the costs for machining processes (including machine cost and tool cost) are converted to the weights of the cities; the costs for preparing processes (including machine changing, tool changing and set-up changing) are converted to the 'distance' between cities. Solving the TSP means finding the route with minimal total cost, which is known to be an NP-hard problem (Nguyen et al. 2007). The time required for the optimal analysis to derive the shortest path of a TSP using an analytic method increases exponentially as the problem complexity increases. However, this problem is more complicated than TSP as, unlike the classical TSP, multiple visits of a hole may be needed for holes to be machined with several different tools. In addition to that, a hole needs to be drilled with different tools in a specific order and the switching time for a tool to another tool depends on the current tool and the tool of interest. For this reason, heuristics and metaheuristics are often used in more recent studies.

Biologically-inspired algorithms are one of the main categories of metaheuristic algorithms. The power and beauty of these algorithms comes from the ability to emulate the best features in nature, namely by selecting the fittest in biological systems through natural selection over millions of years of evolution via two important characteristics: the selection of the fittest, and the adaptation to the environment. Statistically, these characteristics can be classified as intensification and diversification (Blum and Roli 2003; Gazi and Passino 2004; Yang 2010a). Intensification intends to improve the existing solutions in a local region by exploiting the current good solutions, while diversification makes sure that the algorithm can explore the search space more efficiently, often via randomization (Yang and Deb 2010). A good trade-off between diversification and intensification will often lead to a more efficient algorithm (Yang 2010a).

Various bio-inspired optimization algorithms have been presented in literature. Among the most popular algorithms are genetic algorithms (Goldberg 1989), tabu search (Glover 1986), particle swarm optimization (Eberhart and Kennedy 1995), ant colony optimization (Dorigo and Di Caro 1999), differential evolution (Storn and Price 1995), cuckoo search (Yang and Deb 2009), hunting search (Oftadeh et al. 2010), and bat algorithm (Yang 2010b). The cuckoo search (CS) developed by Yang and Deb (2009) is based on the obligate brood-parasitic behavior of some cuckoo species in combination with the Lévy flight behavior of some birds and fruit flies. The preliminary studies on the CS appears to be very promising and could outperform existing algorithms in solving optimization problems (Gandomi et al. 2012). CS algorithm was successfully applied for solving manufacturing optimization problems (Yildiz 2013) and for optimization

of sequence in printed circuit board holes drilling problems (Lim et al. 2012).

In recent years, bio-inspired algorithms and its variants have been applied widely to solve many combinatorial optimization problems (Yang et al. 2013). For example, Gandomi et al. (2013) presented a chaos-enhanced accelerated particle swarm optimization algorithm to solve three engineering problems. Sayadi et al. (2012) presented discrete firefly algorithm (DFA) to solve the manufacturing cell formation optimization problem. Li et al. (2003) applied a tabu-enhanced genetic algorithm approach for assembly process planning problem. Ho and Ji (2004) presented a hybrid genetic algorithm to optimize the sequence of component placements on a printed circuit board. Prakash et al. (2008) proposed an adaptive hierarchical ant colony optimization to solve the traditional machine loading problem in flexible manufacturing systems. Arnaout (2013) introduced a three-stage ant colony optimization algorithm for location allocation problem with an unknown number of facilities. Li et al. (2012) investigated two-tool parallel drilling process plan optimization problem using two phase GA. In the first phase, the GA is used to determine the optimal process parameters satisfying all constraints such as drill feed, spindle speed, thrust force, torque, power and tool life, and the minimum machining time is the optimization criteria. In the second phase, the GA is used to determine the operation completion time and machining sequence. The minimum operation completion time is the optimization criteria in this phase. Yang and Deb (2013) presented multi-objective cuckoo search algorithm for design optimization problems.

For many continuous optimization problems, cuckoo search can find the desired solutions very efficiently. However, sometimes, some difficulty may arise, which is true for all nature-inspired algorithms when the appropriate solutions could not be found for some other optimization problems. This is consistent with the so-called No Free Lunch theorem (Wolpert and Macready 1997). To circumvent this theorem, hybridization has been applied to optimization algorithms for solving a given set of problems. These hybridized algorithms are the combination of components from two or three algorithms which performs more efficiently than their parent algorithms.

In line with this, cuckoo search has been hybridized with other optimization algorithms, machine learning techniques, heuristics, etc. Hybridization can take place in almost every component of the cuckoo search (Fister et al. 2014). For example, initialization procedure, evaluation function, moving function and others have all been tried by many researches. A hybrid CS/GA algorithm was developed to solve global optimization problems (Ghodrati and Lotfi 2012) and reliability-redundancy optimization problems (Kanagaraj et al. 2013), hybrid CS/PSO was developed for solving global optimization problems (Ghodrati and Lotfi 2012), a hybrid cuckoo search via Lévy flights for the permutation flow shop scheduling problems (Li and Yin 2013) and fuzzy assisted hybrid cuckoo search algorithm for multi-objective scheduling problems (Chandrasekaran and Simon 2012) have been applied and showed promising results. It is also found that genetic algorithms can be hybridized with many algorithms such as particle swarm optimization; more specifically, may involve the use of generic operators to modify some components of another algorithm (Yang and Koziel 2011). In this paper a hybrid of cuckoo search and genetic algorithm (CSGA) is proposed to find the optimal sequence of hole-making process. The performance of the hybrid CSGA is demonstrated with small and large size problem instances which can significantly reduce the non-cutting time of hole-making process.

The remainder of this paper is organized as follows. A brief description of hole making problem is presented in "Hole-making sequence optimization problem" section. The proposed hybrid CSGA developed in this study is described in "Proposed cuckoo search-genetic algorithm" section. The verification of the algorithm detail is given in "Algorithm verification" section. The details of computational experiments used to test the performance of CSGA are discussed in "Computational experience" section. Finally, the results from the study are summarized in the last section.

Hole-making sequence optimization problem

When drilling a group of holes on a workpiece using a computer numberical control (CNC) drilling machine, the machine table is driven back and forth in the $x - y$ directions so that each hole is to be drilled in its designed position. Many times, several cutting tool is needed to machine a hole to the specific size and tolerance; this means that there is also a need for the turret lathe to be rotated back and forth. The optimum drilling sequence can minimize the total table and turret lathe movements, thus shorten the no-cutting time and lengthen working life of the table's driving system as well as the turret lathe.

The drilling operation can be divided into three parts: the drilling of the holes, the positioning of the worktable, and the switching of the cutting tool. The total operation time for the drilling process is simply the sum of the machining time, the positioning time, and the tool switching time. Since machining time is decided by the CNC program, hence minimizing the positioning time and the tool switching time reduces the total operation time for the drilling process. This can only be achieved by optimizing the sequence of drilling.

The distance between two holes are often considered to be rectilinear or Euclidean. The rectilinear distance between two points, say i and j of coordinates (x_i, y_i) and (x_j, y_j), respectively, can be calculated with the following equation:

$$d_{ij}^{R} = |x_i - x_j| + |y_i - y_j| \tag{1}$$

The movement of the worktable in the x and y directions are realized using stepper motors (Onwubolu and Clerc 2004). By considering this machine characteristics, the problem to be solved here is to find a sequence in which the holes are to be drilled such that the worktable positioning time is minimized. The time needed for the worktable to position itself depends strongly on the machine characteristics. In practice, usually, this travelling time cannot be computed exactly. Travelling consists of three phases: accelerating the worktable, running at full speed, slowing down to a complete stop. For small distances, full speed may not be reached and we may have anomalies in the sense that a farther position can be reached faster than a nearer position. Even if a timing function is available it may be not accurate and it will be so complicated that its evaluation takes too long for large problem instances (where we cannot store a distance matrix). Therefore one has to be satisfied with making reasonable approximations on the true movement time. In this paper, the gears are taken to rotate at the constant speed at all times. The formula to calculate the positioning time for the worktable to move from point i to point j is as follows:

$$t_{(i,j)} = \sum \frac{|x_i - x_j|}{v_x} + \sum \frac{|y_i - y_j|}{v_y} \tag{2}$$

where v_x and v_y are the linear velocities of the worktable in the x and y directions, respectively. To further simplify the problem, both v_x and v_y are taken to be the same, $v_x = v_y = v$

$$t_{(i,j)} = \frac{1}{v}\left(\sum |x_i - x_j| + \sum |y_i - y_j|\right) \tag{3}$$

Hence, minimizing of the positioning time is simply minimizing the total distance travelled by the worktable, which is represented by the Eq. 1.

However, in optimizing the sequence of drilling, not only the distance travelled by the worktable should be considered; the tool switching time should also be considered. The sequence of drilling should be chosen such that a balance between the positioning time and the tool switching time can be achieved, whereby the total operation time is minimized.

Mathematical formulation

The mathematical model of this hole-making sequence optimization problem is being modeled by Ghaiebi and Solimanpur (2007). Suppose the operation j of hole i is done in order k. If so, the variable x_{ij}^{k} will be equal to 1. Now, if processing of operation j' of hole i' takes place in order $k+1$, the variable $x_{i'j'}^{k+1}$ will be equal to 1. By these assumptions, the tool has to move from hole i to hole i'. Therefore, the corresponding positioning time can be mathematically modeled as $a_{ii'}x_{ij}^{k}x_{i'j'}^{k+1}$. The total positioning time taking all the movements into consideration can be determined by

$$\sum_{i=1}^{I}\sum_{j=1}^{n_i}\sum_{\substack{i'=1\\ i'\neq i}}^{I}\sum_{j'=1}^{n_{i'}}\sum_{k=1}^{N-1} a_{ii'}x_{ij}^{k}x_{i'j'}^{k+1} \tag{4}$$

where I is the total number of holes, N is the total number of operations, and n_i is the number of operations for holes i.

Let us then assume that the tool required for doing operation j of hole i is different from the one needed for doing operation j' of hole i'. If so, the parameter $\delta(T_{ij}, T_{i'j'})$ will be equal to 1, where T_{ij} is the tool required for operation j of hole i. If the time required to switch from the tool used for making operation j of hole i into the tool required for the operation j' of hole i' is denoted by $s_{ij,i'j'}$, this time can be mathematically modeled as $s_{ij,i'j'}\delta(T_{ij}, T_{i'j'})x_{ij}^{k}x_{i'j'}^{k+1}$. The total tool switching time can then be calculated by

$$\sum_{i=1}^{I}\sum_{j=1}^{n_i}\sum_{i'=1}^{I}\sum_{j'=1}^{n_{i'}}\sum_{k=1}^{N-1} s_{ij,i'j'}\delta(T_{ij}, T_{i'j'})x_{ij}^{k}x_{i'j'}^{k+1} \tag{5}$$

Consequently, the objective function of the proposed zero-one mathematical programming model is expressed as follows:

Minimize:

$$\sum_{i=1}^{I}\sum_{j=1}^{n_i}\sum_{\substack{i'=1\\ i'\neq i}}^{I}\sum_{j'=1}^{n_{i'}}\sum_{k=1}^{N-1} a_{ii'}x_{ij}^{k}x_{i'j'}^{k+1} + \sum_{i=1}^{I}\sum_{j=1}^{n_i}\sum_{i'=1}^{I}\sum_{j'=1}^{n_{i'}}\sum_{k=1}^{N-1} s_{ij,i'j'}\delta(T_{ij}, T_{i'j'})x_{ij}^{k}x_{i'j'}^{k+1} \tag{6}$$

Subject to:

$$\sum_{k=1}^{N} x_{ij}^{k} = 1, \quad i = 1, 2, \ldots, I, \quad j = 1, 2, \ldots, n_i \tag{7}$$

$$\sum_{i=1}^{I}\sum_{j=1}^{n_i} x_{ij}^{k} = 1, \quad k = 1, 2, \ldots, N \tag{8}$$

$$x_{ijk} \leq \sum_{k'=k+1}^{N} x_{i,j+1,}^{k'} = 1, \quad i = 1, 2, \ldots, I, \quad i = 1, 2, \ldots, n_i - 1 \tag{9}$$

$$x_{ij}^{k} \in \{0, 1\} \quad \forall i, j, k \tag{10}$$

Constraints 7 ensure that each operation of each hole is assigned to only one position in the sequence. Similarly, Constraints 8 guarantee that only one operation is assigned to each position of the sequence. Constraints 9 represent the

precedence of operations of each hole. Lastly, Constraints 10 confine the decision variables into zero-one values.

Proposed cuckoo search-genetic algorithm

Yang and Deb (2009) developed the CS optimization algorithm which was inspired by the obligate brood parasitism of some cuckoo species by laying their eggs in the nests of other host birds (often other species). Sometimes the host birds engage in direct conflict with the intruding cuckoos. If a host bird discovers that the eggs do not belong to it, it will either kick these alien eggs out from the nest or simply abandon its nest and build a new nest elsewhere.

The algorithm of the original CS is then governed by three rules (Yang and Deb 2009):

1. Each cuckoo lays one egg at a time and selects a nest randomly.
2. The best nest with the highest quality egg can pass onto the new generations.
3. The number of host nests is fixed, and the egg laid by a cuckoo can be discovered by the host bird with a probability p_a.

However, several behaviors of cuckoo birds were not taken into account in the original CS implementation. Firstly, the original CS failed to consider the mating of cuckoo birds. Secondly, the evolution of cuckoo birds to improve adaptation was also not considered. To reduce eggs discrimination by the host birds, cuckoo birds evolve to lay eggs that mimics the eggs of certain species of host birds (Davies et al. 1989). In some cases, the young cuckoo can even mimic the begging call of the young hosts. Studies have also shown that when there is a lack of suitable nests, parasitized nests can be parasitized the second time by another cuckoo. Since only one cuckoo is ever reared per nest, it pays for the second cuckoo to remove the first cuckoo's egg because the earlier laid egg is likely to hatch first (Davies and Brooke 1988). All these behaviors serve as an inspiration in the hybridizing of CS with GA. In the original CS (for continuous space), Lévy flights and random walks are used to generate new solutions. In solving combinatorial problems (discrete integers solution), the GA operators take the place of both Lévy flights and the random walks. More specifically, the crossover operator takes the place of the Lévy flights, and the mutation operator takes the place of the random walk.

The proposed hybridized CSGA as shown in Fig. 1 can be summarized as follows:

1. Cuckoos mate with one another and chromosomal crossover occurs between them.

```
Objective function f(x), x = (x_1, ..., x_d)^T
Generate initial population x_i, (i = 1, 2, ..., N)
while t < maxGeneration or stopping criterion
    Cuckoos crossover to reproduce new cuckoo eggs, y_i
    Evaluate fitness, F(y_i)
    if F(y_i) > F(x_i)
        Replace x_i with y_i
    end if
    Cuckoo eggs mutate and lay in other hosts' nests
    Low quality eggs are rejected by host birds
    Rotation of represented solutions
    Find the current best solution
end while
```

Fig. 1 Pseudocode of cuckoo search-genetic algorithm

2. Cuckoos lay eggs in other host birds' nests and compete with each other. Only the best eggs will survive.
3. Cuckoos evolve by mutation to lay eggs that mimic the host's.
4. Low quality eggs are rejected by host birds.

Solution representation

In this hybrid CSGA, each cuckoo bird represents a solution in the current generation, and cuckoo eggs represent the newly generated solutions in that generation (either by crossover or mutation). The solution is represented through a vector of integers in which each integer in the solution vector corresponds to a hole to be machined. Each point i to be machined is numbered with a unique number beginning from 1 with an increment of 1 until all the points are numbered. i is repeated for n_i times in the solution and therefore the length of the solution depends on the total number of operations needed to complete the hole-making process. For k number of holes that requires n_i number of operation for each hole i, the solution is:

$$x = (x_1, \ldots, x_N) \tag{11}$$

where each element in x represents the hole to be machined and

$$N = \sum_{i=1}^{k} n_i \tag{12}$$

Let us consider a simple 3 holes problem, numbered as 1, 2 and 3, as shown in Fig. 2. Hole 1 requires one operation, hole 2 requires two operations, and hole 3 requires three operations to complete the hole-making process. Using Eqs. 11 and 12, the length of the solution is 6. Suppose in the simple 3 holes problem illustrated, the 3 holes are needed to be finished with the following cutting tool in the following order:

- Hole 1 - *Twist drill bit*
- Hole 2 - *Center drill bit* → *Twist drill bit*
- Hole 3 - *Center drill bit* → *Twist drill bit* → *Reamer*

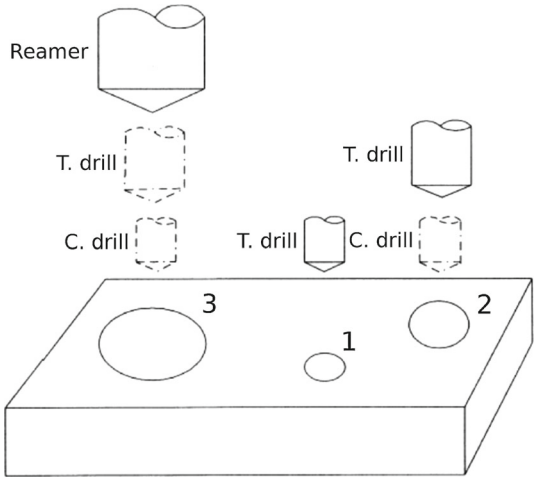


Fig. 2 A simple three holes problem

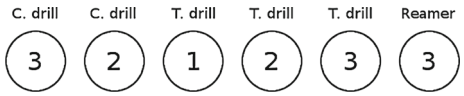


Fig. 3 Solution interpretation

A solution sequence or vector of 3-2-1-2-3-3 would means that the first operation for hole 3, hole 2, and hole 1 is done one after another using the corresponding cutting tool. The second operation for hole 2 is then followed after, and then last two operations of hole 3 with the corresponding cutting tool. The meaning of the solution summarized in Fig. 3.

Due to the constrains of the order of cutting tools, for the solution sequence or vector 3-2-1-2-3-3, it is understood that the "3" that appears first means that the hole will be drilled with a *center drill bit*. It follows that the second appearing "3" refers to the drilling of hole 3 using a *twist drill bit* and the third with a *reamer*. This rule also applies to all other holes. Thus, in the representation of solution, there is no need to specify the cutting tool used in the operation.

Initial population

The initial population is generated randomly. One possible way is to first generate a valid solution vector by listing the hole number from the first hole to the last hole. Where a hole needs more than one operations, the hole number repeatedly listed according to the number of operations required. As for the simple example shown in Fig. 2, the solution vector would be 1-2-2-3-3-3. After the valid solution vector is generated, the initial population can be easily generated by randomizing the order of the elements in the vector.

Crossover operator

Before carrying out the crossover operator, the order of the solutions are randomized to allow cuckoos to mate with a more diversify population. Then the parent cuckoo birds are split into pairs. Two cuckoo eggs are reproduced from the two parent cuckoos.

The two-point crossover technique was adopted. Two points, i and j, are randomly selected for each pair of parents. Everything between the two points is swapped between the two parents, generating two eggs. Using the 3 holes problem as an example again, when $i = 4,\ j = 6$:

Parents:

3	2	1	2	3	3

2	3	3	3	2	1

Children:

3	2	1	3	2	3

2	3	3	2	3	1

However, most of the time the crossover requires some repair work to be done. In the reparation work, the excess genes are first identified. From the valid solution vector generated in "Initial population" section the number of occurrences for each hole number can be determined. The children's genes are compared with the valid solution vector. All the hole numbers in the children solution vector that has a number of occurrence that exceeds that of the same hole number in valid solution vector is identified. These excess genes are randomly removed from the chromosome until the number of occurrences of all hole numbers in the children solution vector do not exceed the number of occurrence of the same hole number. These excess genes are deleted from the chromosome that are apart of the swapped genes.

The children chromosomes are once again compared with the valid solution vector. This time, all the hole numbers in the children solution vector that has a number of occurrence that is less than that of the same hole number in valid solution vector is identified. These missing genes are added to the chromosomes in a random order right after the swapped genes until the number of occurrences of all hole numbers are matched. The illustration of the reparation work is as follows:

Parents:

3	2	1	2	3	3

3	3	3	2	2	1

Children:

3	2	1	2	2	3

3	3	3	2	3	1

Reparation:

Step 1: Randomly remove excess genes

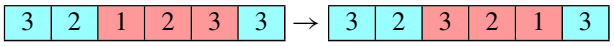
3	1	2	2	3

3	3	2	3	1

Step 2: Insert missing genes

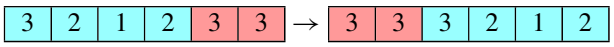
Mutation operator

In the mutation operation, each cuckoo egg mutates on its own. To allow small variations in the solution, a simple 2-opt moves proposed by Croes (1958) is used:



Rotation strategy

After each generation, the genes of all the cuckoo birds are rotated randomly. This rotation strategy serves as a simple way to diversify the search without introducing significant additional computational cost.



Elitism

Each time after the crossover, mutation, and rotation strategy, a form of elitism is performed to ensure that the best solutions are selected and maintained in the population. This procedure is summarized with the following equation:

$$x_{i,t+1} = \begin{cases} y_i & \text{if } F(y_i) > F(x_{i,t}) \\ x_{i,t} & \text{otherwise} \end{cases} \quad i = 1, 2, \ldots, N \quad (13)$$

where y is the newly obtained solution, x is the current solution, and t is the generation number.

Algorithm verification

To verify the performance of CSGA, the algorithm will be used to solve the example part in Fig. 4 proposed by Ghaiebi and Solimanpur (2007). The position and size of various holes are indicated in Fig. 5. The x and y coordinates for the holes are taken from Ghaiebi and Solimanpur (2007). In this problem, there are a total of 5 drill bits with different sizes and 1 reamer used. The tool and its respective diameter is shown in Table 1. Depending on the what the current tool is, the time needed to switch to a certain tool varies and they are asymmetrical. The switching time for the tool is given in Table 2.

Since each hole is of different size, each hole has to be drilled several times with different set of cutting tools. To solve this problem, the tool required and the sequence of the passes for each holes are shown in Table 3. In this problem, it is taken that the speed of the worktable to be $v = 1m\,min^{-1}$ and that tool 1 is currently in use in the spindle. The initial position of the cutting tool is at the coordinate (0, 0) at the bottom-left corner of the workpiece.

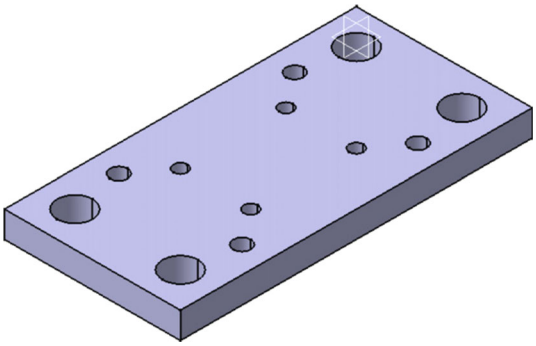


Fig. 4 Solid model of example part

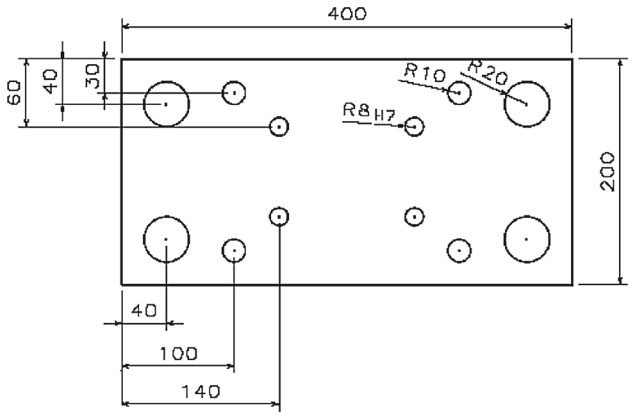


Fig. 5 Position of the holes on the example part

Table 1 Cutting tool diameter

Cutting tool	Drill					Reamer
	1	2	3	4	5	6
Diameter	10	15.8	20	30	40	16

Table 2 Tool switching times in minutes

Predecessor tool	Successor tool					
	1	2	3	4	5	6
1	0.0	0.6	0.2	0.4	0.4	0.9
2	0.6	0.0	0.8	1.2	0.4	0.8
3	0.2	0.8	0.0	0.6	1.4	1.2
4	0.4	1.2	0.6	0.0	0.4	0.7
5	0.4	0.4	1.3	0.5	0.0	0.8
6	0.5	0.5	1.2	0.2	0.8	0.0

Table 3 Sequence of cutting tools needed to machine the holes

Hole	R8	R10	R20
Required tools	1-2-6	1-3	1-3-4-5

Table 4 Statistical results of CSGA and CS for the example part

Algorithm	Best	Mean	Worst	SD	Time (s)
CS	6.26	6.564	7.42	0.5216	84.55
CSGA	6.10	6.216	6.46	0.0964	49.44

In solving this problem, the number of cuckoos is set to 100 and the generation number is set to 1000. The CS by Lim et al. (2012) was also applied to compare the performance of CSGA and CS. The algorithms were made to run 25 times and the statistical results for CSGA and CS are summarized in Table 4.

The numerical results and the solutions obtained by ACO (Ghaiebi and Solimanpur 2007), PSO (Hsieh et al. 2011a), IA Hsieh et al. 2011b, CS and CSGA is shown in Table 5, where the column '**Algo**' indicates the algorithm used to solve the problem, the column '**Obj**' denotes the objective function value associated with the solution, which is the total operation time, the column '**PT**' and '**ST**' shows the positioning time and the switching time, respectively, and the '**Sequence of operation**' shows the solution found by the respective algorithms in the format of (hole, tool). The 5 solutions obtained by CSGA and CS in Table 5 are the best 5 solutions in the 25 runs.

As pointed out by Hsieh et al. (2011b), the objective function value of the solution provided by Ghaiebi and Soliman-

Table 5 Sequence of cutting tools needed to machine the holes

Algo	Obj	PT	ST	Sequence of operations
ACO	8.80	4.58	4.30	(5,1)(9,1)(12,1)(8,1)(7,1)(11,1)(10,1)(6,1)(2,1)(1,1)(5,3)(8,3)(11,2)(12,2)(9,2)(10,2)(6,3)(7,3)(3,1)(4,1)(4,3)(3,3)(2,3)(1,3)(1,4)(2,4)(3,4)(4,4)(12,6)(9,6)(10,6)(11,6)(3,5)(4,5)(1,5)(2,5)
PSO	7.42	4.02	3.40	(9,1)(2,1)(6,1)(10,1)(3,1)(8,1)(4,1)(12,1)(11,1)(7,1)(7,3)(6,3)(5,1)(1,1)(2,3)(1,3)(5,3)(8,3)(4,3)(3,3)(3,4)(4,4)(4,5)(3,5)(11,2)(12,2)(9,2)(10,2)(10,6)(9,6)(12,6)(11,6)(2,4)(1,4)(1,5)(2,5)
	7.44	4.64	2.80	(1,1)(5,1)(2,1)(6,1)(2,3)(1,3)(10,1)(9,1)(4,1)(8,1)(12,1)(11,1)(7,1)(3,1)(7,3)(6,3)(5,3)(8,3)(4,3)(3,3)(3,4)(4,4)(1,4)(2,4)(2,5)(3,5)(4,5)(1,5)(9,2)(10,2)(11,2)(12,2)(11,6)(12,6)(9,6)(10,6)
	7.52	4.52	3.00	(5,1)(1,1)(9,1)(10,1)(2,1)(6,1)(11,1)(7,1)(8,1)(4,1)(3,1)(12,1)(8,3)(4,3)(3,3)(7,3)(6,3)(5,3)(1,3)(2,3)(2,4)(1,4)(4,4)(4,5)(1,5)(10,2)(11,2)(12,2)(9,2)(10,6)(9,6)(12,6)(11,6)(3,4)(3,5)(2,5)
	7.54	4.54	3.00	(9,1)(1,1)(2,1)(10,1)(6,1)(5,1)(12,1)(7,1)(11,1)(3,1)(8,1)(4,1)(4,3)(5,3)(1,3)(2,3)(6,3)(7,3)(3,3)(8,3)(4,4)(3,4)(3,5)(11,2)(12,2)(9,2)(10,2)(10,6)(11,6)(12,6)(9,6)(1,4)(2,4)(2,5)(4,5)(1,5)
	7.56	4.76	2.80	(5,1)(2,1)(11,1)(3,1)(12,1)(8,1)(4,1)(7,1)(7,3)(3,3)(8,3)(4,3)(5,3)(6,1)(10,1)(1,1)(1,3)(2,3)(6,3)(9,1)(1,4)(2,4)(3,4)(4,4)(4,5)(3,5)(2,5)(1,5)(9,2)(12,2)(11,2)(10,2)(10,6)(11,6)(12,6)(9,6)
IA	6.26	3.86	2.40	(1,1)(5,1)(8,1)(4,1)(3,1)(7,1)(12,1)(9,1)(10,1)(2,1)(6,1)(6,3)(2,3)(1,3)(5,3)(8,3)(4,3)(3,3)(7,3)(11,1)(11,2)(12,2)(9,2)(10,2)(10,6)(11,6)(12,6)(9,6)(1,4)(2,4)(3,4)(4,4)(4,5)(3,5)(2,5)(1,5)
	6.32	3.92	2.40	(1,1)(5,1)(9,1)(12,1)(8,1)(4,1)(3,1)(7,1)(11,1)(10,1)(6,1)(2,1)(2,3)(1,3)(5,3)(8,3)(4,3)(3,3)(7,3)(6,3)(10,2)(9,2)(12,2)(11,2)(11,6)(12,6)(9,6)(10,6)(2,4)(1,4)(4,4)(3,4)(3,5)(4,5)(1,5)(2,5)
	6.34	3.94	2.40	(5,1)(1,1)(2,1)(6,1)(7,1)(3,1)(11,1)(12,1)(8,1)(4,1)(4,3)(8,3)(3,3)(7,3)(6,3)(2,3)(1,3)(5,3)(9,1)(10,1)(10,2)(9,2)(12,2)(11,2)(11,6)(12,6)(9,6)(10,6)(2,4)(1,4)(4,4)(3,4)(3,5)(4,5)(1,5)(2,5)
	6.34	3.94	2.40	(5,1)(1,1)(2,1)(6,1)(10,1)(9,1)(12,1)(11,1)(7,1)(3,1)(4,1)(8,1)(8,3)(4,3)(3,3)(7,3)(6,3)(2,3)(1,3)(5,3)(9,2)(10,2)(11,2)(12,2)(12,6)(11,6)(9,6)(10,6)(2,4)(1,4)(4,4)(4,3)(3,5)(4,5)(1,5)(2,5)
	6.36	3.96	2.40	(1,1)(5,1)(6,1)(2,1)(10,1)(12,1)(8,1)(4,1)(3,1)(11,1)(7,1)(7,3)(3,3)(4,3)(8,3)(5,3)(1,3)(2,3)(6,3)(9,1)(9,2)(12,2)(11,2)(10,2)(10,6)(11,6)(12,6)(9,6)(1,4)(2,4)(3,4)(4,4)(4,5)(3,5)(2,5)(1,5)
CS	6.26	3.86	2.40	(1,1)(5,1)(8,1)(4,1)(3,1)(7,1)(12,1)(9,1)(10,1)(2,1)(6,1)(6,3)(2,3)(1,3)(5,3)(8,3)(4,3)(3,3)(7,3)(11,1)(11,2)(12,2)(9,2)(10,2)(10,6)(11,6)(12,6)(9,6)(1,4)(2,4)(3,4)(4,4)(4,5)(3,5)(2,5)(1,5)
	6.26	3.86	2.40	(1,1)(5,1)(8,1)(4,1)(3,1)(7,1)(12,1)(9,1)(10,1)(2,1)(6,1)(6,3)(2,3)(1,3)(5,3)(8,3)(4,3)(3,3)(7,3)(11,1)(11,2)(12,2)(9,2)(10,2)(10,6)(11,6)(12,6)(9,6)(1,4)(2,4)(3,4)(4,4)(4,5)(3,5)(2,5)(1,5)
	6.34	3.94	2.40	(5,1)(1,1)(2,1)(6,1)(7,1)(3,1)(11,1)(12,1)(8,1)(4,1)(4,3)(8,3)(3,3)(7,3)(6,3)(2,3)(1,3)(5,3)(9,1)(10,1)(10,2)(9,2)(12,2)(11,2)(11,6)(12,6)(9,6)(10,6)(2,4)(1,4)(4,4)(3,4)(3,5)(4,5)(1,5)(2,5)
	6.34	3.94	2.40	(5,1)(1,1)(2,1)(6,1)(7,1)(3,1)(11,1)(12,1)(8,1)(4,1)(4,3)(8,3)(3,3)(7,3)(6,3)(2,3)(1,3)(5,3)(9,1)(10,1)(10,2)(9,2)(12,2)(11,2)(11,6)(12,6)(9,6)(10,6)(2,4)(1,4)(4,4)(3,4)(3,5)(4,5)(1,5)(2,5)
	6.34	3.94	2.40	(5,1)(1,1)(2,1)(6,1)(10,1)(9,1)(12,1)(11,1)(7,1)(3,1)(4,1)(8,1)(8,3)(4,3)(3,3)(7,3)(6,3)(2,3)(1,3)(5,3)(9,2)(10,2)(11,2)(12,2)(12,6)(11,6)(9,6)(10,6)(2,4)(1,4)(4,4)(4,3)(3,5)(4,5)(1,5)(2,5)

Table 5 continued

Algo	Obj	PT	ST	Sequence of operations
CSGA	6.10	3.70	2.40	(5,1)(1,1)(2,1)(6,1)(10,1)(11,1)(7,1)(3,1)(4,1)(12,1)(8,1)(8,3)(4,3)(3,3)(7,3)(6,3)(2,3)(1,3) (5,3)(9,1)(9,2)(10,2)(11,2)(12,2)(12,6)(11,6)(10,6)(9,6)(1,4)(2,4)(3,4)(4,4)(4,5)(3,5)(2,5)(1,5)
	6.10	3.70	2.40	(5,1)(1,1)(2,1)(6,1)(10,1)(11,1)(7,1)(3,1)(4,1)(12,1)(8,1)(8,3)(4,3)(3,3)(7,3)(6,3)(2,3)(1,3) (5,3)(9,1)(9,2)(10,2)(11,2)(12,2)(12,6)(11,6)(10,6)(9,6)(1,4)(2,4)(3,4)(4,4)(4,5)(3,5)(2,5)(1,5)
	6.14	3.74	2.40	(5,1)(1,1)(2,1)(6,1)(10,1)(9,1)(12,1)(11,1)(7,1)(3,1)(4,1)(8,1)(8,3)(4,3)(3,3)(7,3)(6,3)(2,3) (1,3)(5,3)(9,2)(10,2)(11,2)(12,2)(12,6)(11,6)(10,6)(9,6)(1,4)(2,4)(3,4)(4,4)(4,5)(3,5)(2,5)(1,5)
	6.14	3.74	2.40	(5,1)(1,1)(2,1)(6,1)(10,1)(11,1)(12,1)(8,1)(4,1)(3,1)(7,1)(7,3)(3,3)(4,3)(8,3)(5,3)(1,3)(2,3) (6,3)(9,1)(9,2)(10,2)(11,2)(12,2)(12,6)(11,6)(10,6)(9,6)(1,4)(2,4)(3,4)(4,4)(4,5)(3,5)(2,5)(1,5)
	6.14	3.74	2.40	(5,1)(1,1)(2,1)(6,1)(11,1)(12,1)(8,1)(4,1)(3,1)(7,1)(7,3)(3,3)(4,3)(8,3)(5,3)(1,3)(2,3)(6,3) (10,1)(9,1)(9,2)(10,2)(11,2)(12,2)(12,6)(11,6)(10,6)(9,6)(1,4)(2,4)(3,4)(4,4)(4,5)(3,5)(2,5)(1,5)

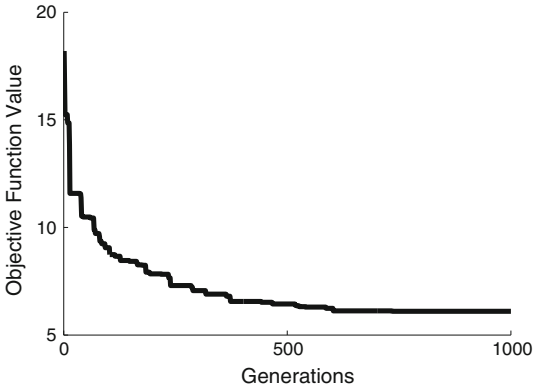


Fig. 6 The best solution of CSGA over 1000 generations

pur (2007) is 8.88 min and not 7.58 min as claimed in their paper. CSGA arrived at its best solution of 6.10 min twice and the calculation of the objective function value is as follows:

- Positioning Time = [(13) + 7 + 12 + 7 + 7 + 12 + 7 + 7 + 12 + 12 + 7 + 0 + 7 + 12 + 7 + 20 + 7 + 12 + 7 + 7 + 0 + 8 + 12 + 8 + 0 + 8 + 12 + 8 + 12 + 12 + 32 + 12 + 0 + 12 + 32 + 12 + (8)]/100 = 3.70
- Switching Time = 0.2+0.2+0.6+0.8+0.2+0.4 = 2.40
- Total Operation Time = 3.70 + 2.40 = 6.10

where the time in brackets denotes the time to move from the initial position to the first hole, and the time to move from the last hole back to the initial position. Figure 6 shows the convergence plot of the solution.

It is clear from Table 5 the proposed CSGA performs astonishingly well when compared to ACO, PSO, IA, and CS. The 5 best solutions obtained by CSGA are better than all the solutions obtained by the other algorithms. The 25 independent runs of CSGA return solutions with an average objective function value of 6.216, which is, by itself, better than the best solutions obtained by ACO, PSO, IA, and CS. It can thus be concluded that CSGA is a more efficient algorithm.

Computational experience

The performance of the proposed CSGA was further evaluated through comparing it with ACO proposed by Ghaiebi and Solimanpur (2007), the IA by Hsieh et al. (2011b) and CS by Lim et al. (2012) for the travelling salesman problem. The TSP problem was made to be easily reproducible. The arrangement of holes for the problem is considered as follows:

- The number of rows in the workpiece is $\lfloor\sqrt{I}\rfloor$ (floor of $\sqrt{I}$), where I is the total number of holes.
- The center to center distance of adjacent holes in each direction is assumed to be 2 cm.

For example, the position of holes in the workpiece when N = 10 is shown in Fig. 7. As seen in this figure, the number of rows is $\lfloor\sqrt{10}\rfloor = 3$.

In solving this TSP problems, the number of holes considered were 5, 10, 15, 20, 25, 50, 100, 200, 500 and 1000.

We have implemented CSGA using MATLAB® R2012a under 64-bit Linux Mint 14: Nadia operating system. Experiments are conducted on a desktop with Intel® Core™ i7-2600 CPU @ 3.40 GHz x 4, and 12 GB of RAM. In each case study, 25 independent runs of the algorithms are carried out with a population number of 2.

The experimental results of the CSGA algorithm are compared with ACO, IA, and CS in Table 6. The results for ACO and IA are obtained from Ghaiebi and Solimanpur (2007) and Hsieh et al. (2011b), respectively, while the results for CS and CSGA are obtained through experiments. For all the test problems compared, CSGA and IA obtained equally good

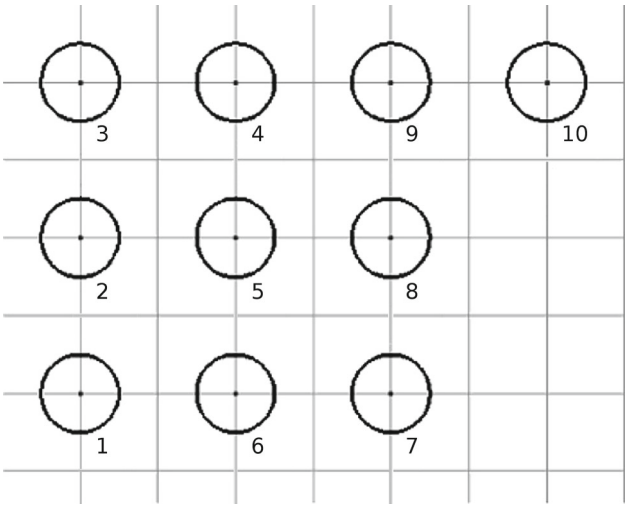


Fig. 7 Position and numbering of the 10 holes in the workpiece

Table 6 Comparison of the experimental results of CSGA with ACO, IA, and CS

N	ACO			IA	CS			CSGA		
	Best	Average	Time	Best	Best	Average	Time	Best	Average	Time
5	**12**	**12**	**0.00**	**12**	**12**	**12**	**0.00**	**12**	**12**	**0.00**
10	**24**	**24**	**0.00**	**24**	**24**	**24**	**0.00**	**24**	**24**	**0.00**
15	**32**	**32**	0.03	**32**	**32**	**32**	0.05	**32**	**32**	**0.02**
20	**40**	**40**	0.14	**40**	**40**	**40**	0.24	**40**	**40**	**0.08**
25	**52**	52.96	3.83	**52**	**52**	**52**	0.76	**52**	**52**	**0.06**
50	136	137.04	20.14	**104**	**104**	111.8	1.37	**104**	**104**	**0.75**

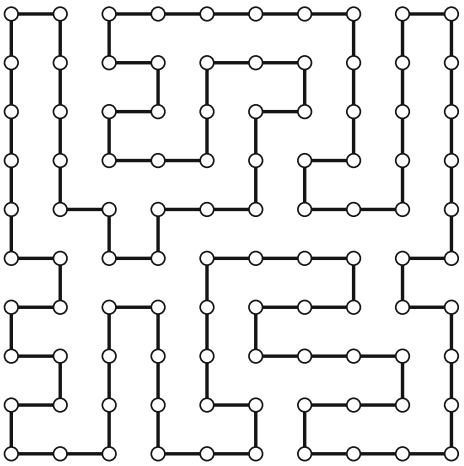


Fig. 8 The best sequence obtained for $N = 100$ by CSGA

solutions. In terms of timings, it can be seen from Table 6 that ACO, CS, and CSGA performs equally well when $N = 5$ and $N = 10$. For $N = 15$ and $N = 20$, both ACO, CS, and CSGA obtained the optimal result and all three algorithms

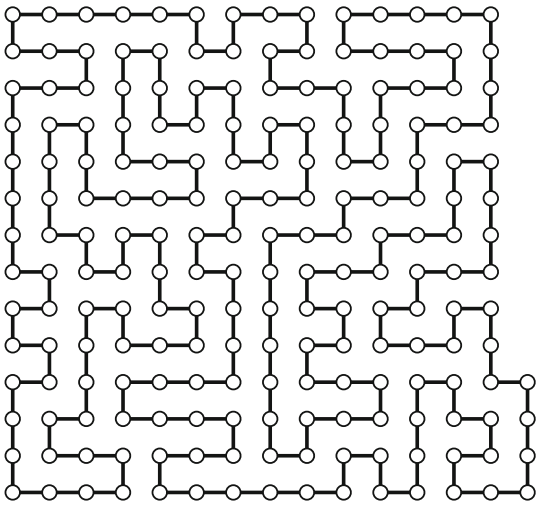


Fig. 9 The best sequence obtained for $N = 200$ by CSGA

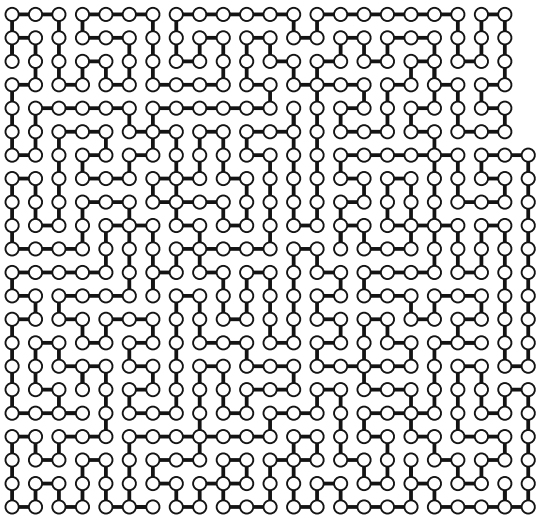


Fig. 10 The best sequence obtained for $N = 500$ by CSGA

have very close timings. However, for $N = 25$ and $N = 50$, CSGA significantly outperform ACO and CS both in terms of both optimality and timings.

The graphical solutions for the best sequences for the larger size problems obtained by CSGA are shown in Figs. 8, 9, 10 and 11, while the complete experimental results are summarized in Table 7, where the first column '**N**' shows the number of holes, the column '**Best**' shows the length of the best solution found by the algorithm, the column '**Mean**' gives the average solution length of the 25 independent runs of the algorithm, while the column '**Worst**' denotes the worst solution length obtained of 25 independent runs, the column '**SD**' denotes the standard deviation of the solutions over 25 runs, and the column '**Time**' shows the average time in seconds for 25 runs. In the Tables 6 and 7, the best results are presented in boldface. As the problem size grows the

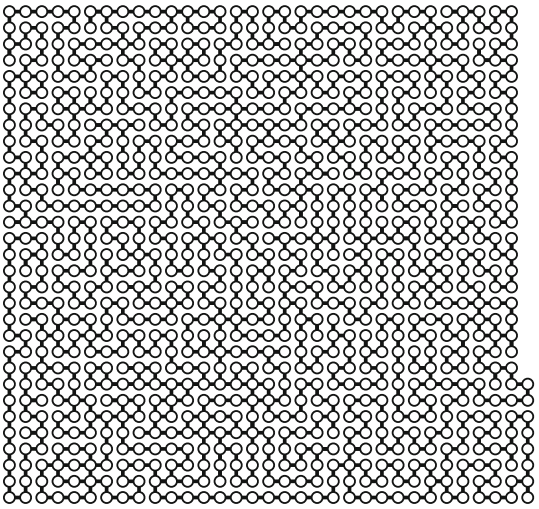


Fig. 11 The best sequence obtained for $N = 1000$ by CSGA

superiority of CSGA over CS becomes more apparent and overwhelming. For $N = 1000$ the best results obtained by CSGA is almost 3 times smaller than CS in addition to a near tenfold reduction in computational time.

Figure 12 shows the convergence plot of the TSP problem with $N = 50$ and $N = 200$ obtained by CS and CSGA. It can be observed that CSGA converges quicker than CS and the prolonged generations of CSGA thereafter are just to slightly converge the solution further. It is evident that CSGA not only obtain better results, it also has a faster convergence speed when compared to CS.

Discussion and conclusion

In this paper, the hybrid CSGA was proposed by incorporating the evolutionary strategies of GA with the breeding behavior of cuckoos. By embedding the GA's operators such as crossover and mutation into the standard CS imitates the real life behavior of cuckoo birds. The success or failure of population based algorithms depends on its ability to establish proper trade-off between exploration and exploitation. Since both GA and CS are population based algorithms, while developing the hybrid CSGA, more attention is given to the two major components of the algorithm: intensification and diversification, or exploitation and exploration (Blum and Roli 2003). The superior performance of CSGA can be explained by the balance between the exploration and exploitation ability of the algorithm. It is also partly due to the fact that the CSGA maintains the Lamarckian property of the solution (Gen and Cheng 2000). Having Lamarckian property in the solutions will ensure meaningful crossover between parent cuckoo birds.

The efficacy of the algorithm should also be attributed to the exploratory search in the 2-point crossover. In complex problems such as the hole-making problem addressed in this paper, many algorithms easily fall into a local minima. The vastly diversifying search of the 2-point crossover allows the CSGA to extensively search the solution space that would otherwise not be searched by the standard search strategies employed for the standard TSP. While exploring the solution space, the crossover of the cuckoo birds still retain partial identity of the parent cuckoo bird rather than blindly searching for better solutions. The mutation of the cuckoo eggs further improves the performance of the algorithm in contributing to the local search of the solution space by making small but logical changes in the genes. In addition, the rotation strategy applied at the end of every generation aids to the search for the global minimum with minimal computational cost, and further reducing the possibility of a premature convergence at a local minima. Some form of elitism strategy is used in CSGA to ensure that best solutions are always carried forward to the next generation.

Besides that, the proposed CSGA is designed in such a way that there is virtually no parameters to be fine tuned; it

Table 7 Comparison of statistical results of CS and CSGA for TSP

N	CS					CSGA				
	Best	Mean	Worst	SD	Time	Best	Mean	Worst	SD	Time
5	**12**	**12**	**12**	**0.00**	**0.00**	**12**	**12**	**12**	**0.00**	**0.00**
10	**24**	**24**	**24**	**0.00**	**0.00**	**24**	**24**	**24**	**0.00**	**0.00**
15	**32**	**32**	**32**	**0.00**	0.05	**32**	**32**	**32**	**0.00**	**0.02**
20	**40**	**40**	**40**	**0.00**	0.24	**40**	**40**	**40**	**0.00**	**0.08**
25	**52**	**52**	**52**	**0.00**	0.76	**52**	**52**	**52**	**0.00**	**0.06**
50	**104**	111.80	124	4.01	1.37	**104**	**104**	**104**	**0.00**	**0.75**
100	236	248.96	268	8.59	17.62	**200**	**200**	**200**	**0.00**	**14.61**
200	660	730.40	808	32.08	32.36	**400**	**408.16**	**416**	**3.91**	**31.61**
500	2,496	2,599.60	2,724	80.49	360.99	**1,052**	**1,071.04**	**1,100**	**10.09**	**101.56**
1,000	6,424	6,638.34	6,940	184.60	2,812.81	**2,240**	**2,291.68**	**2,320**	**20.26**	**297.25**

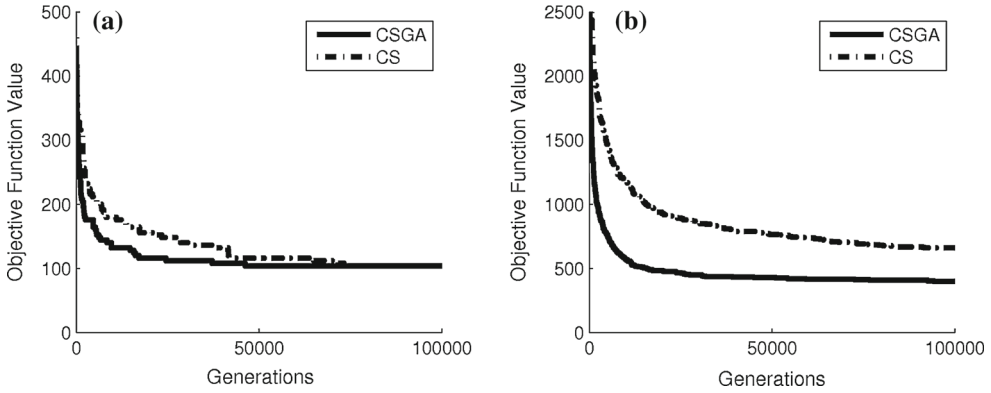


Fig. 12 The best solution by CSGA and CS

has only two basic parameters: the population size and the generation number. The balance between the exploration and exploitation ability of the algorithm can simply be adjusted by the population size and the generation number. A larger number of cuckoos will increase the diversifying ability, while increasing the generation number improves the intensification ability of the algorithm. With this in mind, the population size can simply be chosen based on the size and complexity of the problem. The stopping criterion of the optimization algorithm can then be chosen for a balance between time and solution quality desired by the user. The user can either set the maximum number of generations, or stop the algorithm if no change in solution was made after a certain number of generations.These unique characteristics of CSGA make the algorithm easy to implement and apply and will potentially be the attractive optimization technique for practical engineering optimization problems.

In hole-making operations, to search the best path to complete all the holes is a complex problem, in which random paths may lead to much waste in non-productive time. A 12 holes problem with different hole sizes is considered in this paper. A set of problem instances was also chosen to cover a wide spectrum of typical problem sizes, namely 5–1000 holes evenly distributed. In all the problems considered, CSGA prevailed at a level well above all algorithms compared. As a comparison between CSGA and its parent algorithm, the original CS uses Lévy flights and random walk to generate new solutions for the next iterations. In continuous space problems, these techniques are effective. However, dealing with discrete solutions with these techniques unnecessarily increases the computational cost. CSGA, which uses GA operators, are quick and there is no need to encode and decode the solutions. Thus, in solving combinatorial problems, CSGA surpasses CS in terms of efficiency.

Future works can emphasize on extending the application of this optimization technique to other complex combinatorial engineering optimization problems such as the redundancy allocation problem, the container loading problem, and the job shop scheduling problem.

References

Arnaout, J. P. (2013). Ant colony optimization algorithm for the euclidean location-allocation problem with unknown number of facilities. *Journal of Intelligent Manufacturing*, *24*, 45–54.

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, *35*, 268–308.

Chandrasekaran, K., & Simon, S. P. (2012). Multi-objective scheduling problem: Hybrid approach using fuzzy assisted cuckoo search algorithm. *Swarm and Evolutionary Computation*, *5*, 1–16.

Croes, G. A. (1958). A method for solving traveling-salesman problems. *Operations Research*, *6*, 791–812.

Davies, N. B., Bourke, A. F. G., & de L Brooke, M. (1989). Cuckoos and parasitic ants: Interspecific brood parasitism as an evolutionary arms race. *Trends in Ecology & Evolution*, *4*, 274–278.

Davies, N. B., & Brooke, Md L. (1988). Cuckoos versus reed warblers: Adaptations and counteradaptations. *Animal Behaviour*, *36*, 262–284.

Dereli, T., Filiz, I., & Baykasoglu, A. (2001). Optimizing cutting parameters in process planning of prismatic parts by using genetic algorithms. *International Journal of Production Research*, *39*, 3303–3328.

Dorigo, M., & Di Caro, G. (1999). Ant colony optimization: A new meta-heuristic. In *IEEE proceedings of the 1999 congress on evolutionary computation, 1999 (CEC 99)*.

Eberhart, R., & Kennedy, J. (1995). A new optimizer using particle swarm theory. In *IEEE proceedings of the sixth international symposium on micro machine and human science, 1995 (MHS'95)* (pp. 39–43).

Fister, I., Jr., Yang, X. S., Fister, D., & Fister, I. (2014). Cuckoo search: A brief literature review. In *Cuckoo search and firefly algorithm* (pp. 49–62). Berlin: Springer.

Gandomi, A. H., Talatahari, S., Yang, X. S., & Deb, S. (2012). Design optimization of truss structures using cuckoo search algorithm. *The Structural Design of Tall and Special Buildings*. doi:10.1002/tal.1033.

Gandomi, A. H., Yun, G. J., Yang, X. S., & Talatahari, S. (2013). Chaos-enhanced accelerated particle swarm optimization. *Communications in Nonlinear Science and Numerical Simulation*, *18*, 327–240.

Gazi, V., & Passino, K. M. (2004). A class of attractions/repulsion functions for stable swarm aggregations. *International Journal of Control*, *77*, 1567–1579.

Gen, M., & Cheng, R. (2000). *Genetic algorithms and engineering optimizationm, vol. 7*. New York: Wiley.

Ghaiebi, H., & Solimanpur, M. (2007). An ant algorithm for optimization of hole-making operations. *Computers & Industrial Engineering*, *52*, 308–319.

Ghodrati, A., & Lotfi, S. (2012). A hybrid cs/ga algorithm for global optimization. In *Proceedings of the international conference on soft computing for problem solving (SocProS 2011)* (pp. 397–404). Berlin: Springer.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, *13*, 533–549.

Goldberg, D. E. (1989). Genetic algorithm. *Search, Optimization and Machine Learning*, 343–349.

Ho, W., & Ji, P. (2004). A hybrid genetic algorithm for component sequencing and feeder arrangement. *Journal of Intelligent Manufacturing*, *15*, 307–315.

Hsieh, Y. C., Lee, Y., You, P. S., & Chen, T. C. (2011a). Optimal operation sequence of hole-making with multiple tools in manufacturing: A pso evolutionary based approach. *Key Engineering Materials*, *460*, 398–403.

Hsieh, Y. C., Lee, Y. C., & You, P. S. (2011b). Using an effective immune based evolutionary approach for the optimal operation sequence of hole-making with multiple tools. *Journal of Computational Information Systems*, *7*, 411–418.

Kanagaraj, G., Ponnambalam, S., & Jawahar, N. (2013). A hybrid cuckoo search and genetic algorithm for reliability-redundancy allocation problems. *Computers & Industrial Engineering*, *66*, 1115–1124.

Kolahan, F., & Liang, M. (1996). A tabu search approach to optimization of drilling operations. *Computers & Industrial Engineering*, *31*, 371–374.

Kolahan, F., & Liang, M. (2000). Optimization of hole-making operations: A tabu-search approach. *International Journal of Machine Tools and Manufacture*, *40*, 1735–1753.

Li, J., Khoo, L., & Tor, S. (2003). A tabu-enhanced genetic algorithm approach for assembly process planning. *Journal of Intelligent Manufacturing*, *14*, 197–208.

Li, S., Liu, Y., Li, Y., Landers, R.G., & Tang, L. (2012). Process planning optimization for parallel drilling of blind holes using a two phase genetic algorithm. *Journal of Intelligent Manufacturing* (pp. 1–14).

Li, X., & Yin, M. (2013). A hybrid cuckoo search via lévy flights for the permutation flow shop scheduling problem. *International Journal of Production Research*, *51*, 4732–4754.

Lim, W. C. E., Kanagaraj, G., & Ponnambalam, S. (2012). Cuckoo search algorithm for optimization of sequence in pcb holes drilling process. In *Emerging trends in science, engineering and technology* (pp. 207–216). Brelin: Springer.

Liu, Xj, Yi, H., & Ni, Zh. (2013). Application of ant colony optimization algorithm in process planning optimization. *Journal of Intelligent Manufacturing*, *24*, 1–13.

Merchant, M. E. (1985). World trends and prospects in manufacturing technology. *International Journal of Vehicle Design*, *6*, 121–38.

Nguyen, H. D., Yoshihara, I., Yamamori, K., & Yasunaga, M. (2007). Implementation of an effective hybrid ga for large-scale traveling salesman problems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, *37*, 92–99.

Oftadeh, R., Mahjoob, M., & Shariatpanahi, M. (2010). A novel meta-heuristic optimization algorithm inspired by group hunting of animals: Hunting search. *Computers & Mathematics with Applications*, *60*, 2087–2098.

Onwubolu, G. C., & Clerc, M. (2004). Optimal path for automated drilling operations by a new heuristic approach using particle swarm optimization. *International Journal of Production Research*, *42*, 473–491.

Oysu, C., & Bingul, Z. (2007). Tool path optimization using genetic algorithms. In *GEM* (pp. 120–126).

Prakash, A., Tiwari, M., & Shankar, R. (2008). Optimal job sequence determination and operation machine allocation in flexible manufacturing systems: An approach using adaptive hierarchical ant colony algorithm. *Journal of Intelligent Manufacturing*, *19*, 161–173.

Sayadi, M. K., Hafezalkotob, A., & Naini, S. G. J. (2012). Firefly-inspired algorithm for discrete optimization problems: An application to manufacturing cell formation. *Journal of Manufacturing Systems*, *32*, 78–84.

Storn, R., & Price, K. (1995). *Differential evolutiona simple and efficient adaptive scheme for global optimization over continuous spaces*. International Computer Science Institute, Berkeley. Technical Report. CA, 1995, Tech. Rep. TR-95-012.

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, *1*, 67–82.

Yang, X. S. (2010a). *Nature-inspired metaheuristic algorithms*. Luniver Press.

Yang, X. S. (2010b). A new metaheuristic bat-inspired algorithm. In *Nature inspired cooperative strategies for optimization (NICSO 2010)* (pp. 65–74). Berlin: Springer.

Yang, X. S., Cui, Z., Xiao, R., Gandomi, A. H., & Karamanoglu, M. (2013). *Swarm intelligence and bio-inspired computation: Theory and applications*. Amsterdam: Elsevier.

Yang, X. S., & Deb, S. (2009). Cuckoo search via lévy flights. In *IEEE world congress on nature & biologically inspired computing 2009 (NaBIC 2009)* (pp. 210–214).

Yang, X. S., & Deb, S. (2010). Engineering optimisation by cuckoo search. *International Journal of Mathematical Modelling and Numerical Optimisation*, *1*, 330–343.

Yang, X. S., & Deb, S. (2013). Multiobjective cuckoo search for design optimization. *Computers & Operations Research*, *40*, 1616–1624.

Yang, X. S., & Koziel, S. (2011). *Computational optimization and applications in engineering and industry, vol. 359*. Berlin: Springer.

Yao, Z., & Gupta, S. K. (2004). Cutter path generation for 2.5 d milling by combining multiple different cutter path patterns. *International Journal of Production Research*, *42*, 2141–2161.

Yildiz, A. R. (2013). Cuckoo search algorithm for the selection of optimal machining parameters in milling operations. *The International Journal of Advanced Manufacturing Technology*, *64*, 55–61.

Zhang, Y., & Ge, L. (2009). Selecting optimal set of tool sequences for machining of multiple pockets. *The International Journal of Advanced Manufacturing Technology*, *42*, 233–241.