International Journal of Applied Metaheuristic Computing
Volume 13 • Issue 1

Experimental Programming of Genetic Algorithms for the "Casse-Tête" Problem

Sarab Almuhaideb, King Saud University, Saudi Arabia*
Shahad Hassan Alohaydib, King Saud University, Saudi Arabia
Dana Aldossari, King Saud University, Saudi Arabia
Ghaida Alfarraj, King Saud University, Saudi Arabia
Shahad bin Musaibeeh, King Saud University, Saudi Arabia
Atheer Saad Alrosayes, King Saud University, Saudi Arabia
Afnan Abouelwafa, King Saud University, Saudi Arabia

ABSTRACT

The Casse-tête board puzzle consists of an n×n grid covered with n^2 tokens. n^2 tokens are deleted from the grid so that each row and column of the grid contains an even number of remaining tokens. The size of the search space is exponential. This study used a genetic algorithm (GA) to design and implement solutions for the board puzzle. The chromosome representation is a matrix of binary permutations. Variants for two crossover operators and two mutation operators were presented. The study experimented with and compared four possible operator combinations. Additionally, it compared GA and simulated annealing (SA)-based solutions, finding a 100% success rate (SR) for both. However, the GA-based model was more effective in solving larger instances of the puzzle than the SA-based model. The GA-based model was found to be considerably more efficient than the SA-based model when measured by the number of fitness function evaluations (FEs). The Wilcoxon signed-rank test confirms a significant difference among FEs in the two models (p=0.038).

KEYWORDS

Board Puzzle, Casse Tête, Genetic Algorithm, Simulated Annealing

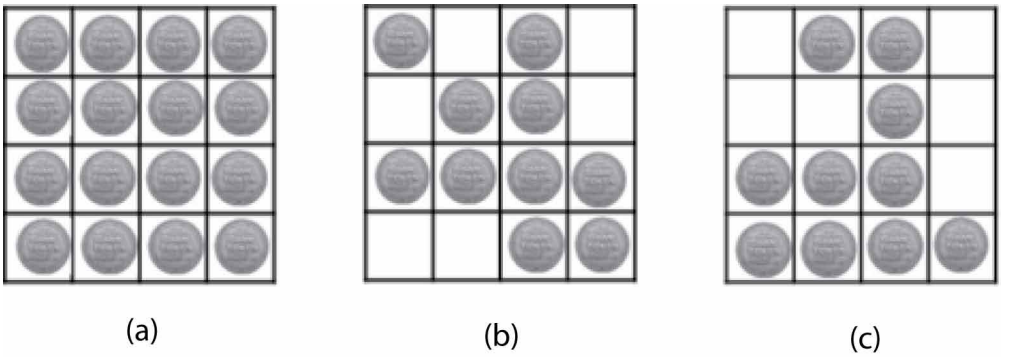
INTRODUCTION

Puzzles are among the most popular ways in which to measure computers' abilities to tackle problems that require intelligence. They have well-defined rules and a performance that can be objectively quantified via numeric scores or win-lose outcomes.

The present research is aimed at solving the Casse-tête problem (Talbi, 2009), which is a one-player board puzzle that consists of a two-dimensional grid with a size of covered with $n^2 n \times n$ tokens (see Figure 1a). The idea of this puzzle is to delete m tokens from the grid, with the number

DOI: 10.4018/IJAMC.292513 *Corresponding Author

Figure 1. (a) Grid with $n = 4$ covered by 16 tokens (b) Example of the correct solution of the problem (c) Example of an incorrect solution of the problem, where n=4 and m=6



of remaining tokens being even in each row and column of the grid (see Figure 1b). The configuration in Figure 1(c) is considered an incorrect solution because there is an odd number of tokens in each of row 2, row 3, column 2, and column 4. The search space size of this problem is given by:

$$C_m^{n^2} = \frac{n^2!}{m!\left(n^2 - m\right)!}$$

which is exponential in size. As far as the authors' are concerned, this problem has not been approached in the previous literature.

A genetic algorithm (GA) (Holland, 1992) is a search and optimization metaheuristic inspired by the Darwinian principle of evolution through genetic selection and variation. A GA uses a decidedly abstract version of development operations to include solutions for a given problem (McCall, 2005). GAs have shown good performance in various NP-hard problems (Katoch et al., 2020), including puzzles that are similar in nature to Casse-tête, such as the n -queens and n -puzzle.

This study involved designing a GA to solve the Casse-tête problem. Despite the abundance of available crossover and mutation operators, no operators were found that were suitable for this problem. Thus, the work involved proposing two new crossover methods to fit the problem: binary partially mapped crossover (BPMX) and binary modified order crossover (BMOC). Moreover, two mutation methods (binary swap [BSwap] and binary inversion [BInversion]) were used to commensurate with the chromosome representation. The fitness evaluation acted as a cost function, penalizing the individual with each row/column with an odd number of tokens. Additionally, the study involved evaluating the proposed model against a simulated annealing (SA)-based solution to the problem.

Figure 2. Example of matrix encoding for the board configuration in Figure 1(b)

1	0	1	0
0	1	1	0
1	1	1	1
0	0	1	1

This article is organized as follows. First, it introduces the background of the problem. Second, a review of similar board puzzles, such as the n-queens problem and the n-puzzle problem, is presented. Third, the methodology for solving the Casse-tête problem is defined based on the requirements and algorithm design. Fourth, the dataset and experimental methodology are explained. Fifth, the experimental results are reported and discussed before conclusions are drawn.

BACKGROUND

Puzzles and board games have dominated artificial intelligence research since its inception in the 1950s (Yannakakis & Togelius, 2018). An associated domain has emerged that comprises conferences, such as the IEEE Conference on Computational Intelligence and Games (CoG), as well as journals, such as the *IEEE Transactions on Games* (ToG). Puzzles are used as tools for teaching in artificial intelligence courses (Sturm, 2009; Uke & Thool, 2011). It is essential to see a diversity of games (beyond chess, Mastermind, or the n-queens) played using artificial intelligence tools. The puzzle is described in *The Moscow Puzzles* (Kordemsky, 1972), a translation of Kordemsky's iconic 1954 book, *Matematicheskaya Smekalka* (Mathematical Quick-Wits). The puzzle, which was originally called Keep it Evan, was restricted to 16 objects arranged in four rows (each with four objects). After six objects are removed, the remaining number of objects in each row and column is even.

In 2009, Talbi (2009) proposed the use of GAs to solve this problem in its original form, without describing the methodology required. This stimulated the current work. However, the definition of the problem is generalized in this work so that the grid is of a variable size, n, and a variable number of objects, m, is removed.

The problem was studied to obtain a deeper understanding of its properties and constraints. A brute-force algorithm was implemented to solve the Casse-tête problem for various small instance sizes. This helped with identifying the properties of the boards that had no solutions. A formal definition of the Casse-tête problem $CT_n\left(m\right)$ of size n with parameter $m \in N$ is presented in Constraints (1) through (5) as follows. Constraint 1 describes that matrix M of size $n \times n$. M_{rc} is a binary variable denoting the cell value of the r^{th} row and c^{th} column of $M, c \in \left\{1,\ldots,n\right\}, r \in \left\{1,\ldots,n\right\}$. M_{rc} equals 1 if a token exists in row r column c, and 0 otherwise. The total number of tokens in the board is $n^2 - m$ in Constraint 2. Constraints 3 and 4 indicate that the number of tokens must be even in every column and row, respectively. Constraint 5 indicates that if n is even, then m must be, too, and vice versa:

$$M \in \left\{0,1\right\}^{n \times n} \tag{1}$$

$$\sum_{r=1}^{n}\sum_{c=1}^{n} M_{rc} = n^2 - m \tag{2}$$

$$(\sum_{r=1}^{n} M_{rc}) mod 2 = 0 \;\; \forall c \in \left\{1,\ldots,n\right\} \tag{3}$$

$$(\sum_{c=1}^{n} M_{rc}) \ mod 2 = 0 \ \ \forall r \in \{1, \ldots, \ n\} \tag{4}$$

$$m \equiv n \left(mod 2\right) \tag{5}$$

This problem has a feasible solution if and only if $\left(n^2 - m\right) mod 2 = 0$ and $n^2 \geq m$. Otherwise, the problem is infeasible. Furthermore, the solution is never unique because the problem is highly symmetric. Swapping rows or columns or rotating the board does not invalidate a feasible solution.

In this research study, $CT_n\left(m\right)$ is approached as a decision problem. Given a grid with size $n \times n$ covered with $n^2 - m$ tokens, is the number of tokens in each row and column of the grid even? It is evident that the validity of an instance of $CT_n\left(m\right)$ can be evaluated in polynomial time. Checking whether the number of tokens is even in each of n rows and n columns of size n can be done in a quantity of time that is quadratic in n.

The $CT_n\left(m\right)$ is expressed as a conjunctive normal form (CNF) satisfiability problem (SAT), which has been proven to be NP-complete (Cook, 1971). The later problem is to determine if a satisfying truth assignment exists for a Boolean formula in CNF. Let $Even\left(\cdot\right)$ be true if its argument (either a row or a column of size *n*) contains an even number of tokens, and false otherwise. The $CT_n\left(m\right)$ problem is about determining whether the conjunction in (6) is satisfiable:

$$Even\left(row_1\right) \wedge Even\left(row_2\right) \wedge \cdots \wedge Even\left(row_n\right) \wedge Even\left(col_1\right) \wedge Even\left(col_2\right) \wedge \cdots \wedge Even\left(col_n\right) \tag{6}$$

Without the loss of generality, for a row of size to contain an even number of tokens, note that the number of tokens must be either 0, 2, …, or, *k* where *k* is the largest even integer not exceeding *n*. The same argument holds for the columns. Let $R_{r,i}$ denote that row *r* contains *i* tokens, and let $C_{c,i}$ denote that column *c* contains *i* tokens. Each conjunct $Even\left(row_r\right)$ can be expressed as the clause of $(R_{r,0} \vee R_{r,2} \vee \cdots \vee R_{rk,})$. Similarly, each conjunct $Even\left(col_c\right)$ can be expressed as the clause of $(C_{c,0} \vee C_{c,2} \vee \cdots \vee C_{c,k})$. The $CT_n\left(m\right)$ problem is then about determining whether the CNF in (7) is satisfiable. As an example, for a problem of size $n = 4$, our problem is about determining whether the CNF in (8) is satisfiable:

$$(R_{1,0} \vee R_{1,2} \vee \cdots \vee R_{1,k}) \wedge (R_{2,0} \vee R_{2,2} \vee \cdots \vee R_{2,k}) \wedge \cdots \wedge (R_{n,0} \vee R_{n,2} \vee \cdots \vee R_{n,k}) \wedge$$

$$(C_{1,0} \vee C_{1,2} \vee \cdots \vee C_{1,k}) \wedge (C_{2,0} \vee C_{2,2} \vee \cdots \vee C_{2,k}) \wedge \cdots \wedge (C_{n,0} \vee C_{n,2} \vee \cdots \vee C_{n,k}) \tag{7}$$

$$(R_{1,0} \vee R_{1,2} \vee R_{1,4}) \wedge (R_{2,0} \vee R_{2,2} \vee R_{2,4}) \wedge (R_{3,0} \vee R_{3,2} \vee R_{3,4}) \wedge (R_{4,0} \vee R_{4,2} \vee R_{4,4}) \wedge$$

$$(C_{1,0} \vee C_{1,2} \vee C_{1,4})(C_{2,0} \vee C_{2,2} \vee C_{2,4})(C_{3,0} \vee C_{3,2} \vee C_{3,4})(C_{4,0} \vee C_{4,2} \vee C_{4,4}) \tag{8}$$

LITERATURE REVIEW

Yannakakis and Togelius (Yannakakis & Togelius, 2018) described a number of game characteristics from the artificial intelligence perspective. These include the number of players, stochasticity, observability, branching factor, and time granularity. The Casse-tête puzzle was not attempted in the literature; therefore, this study reviews metaheuristic-based solutions to board games that are similar to the Casse-tête puzzle in this section, specifically the n -queens, Mastermind, and n -puzzle. Like the Casse-tête, the n -queens and n -puzzle are single player games. In addition, they are completely deterministic. Mastermind is a two-player deterministic game. Considering observability, perfect information exists in the Casse-tête, n -queens, and n -puzzle. In Mastermind, the information is partially hidden. The branching factor is in the order of n^2 in Casse-tête and n -queens. In Mastermind, the branching factor is also quadratic in the number of poles (Ville, 2013). The n -puzzle has a much smaller branching factor, between two (corner tiles) and four (center tiles). The amount of real time that passes is not of high importance in these games. However, time limits are enforced in some variants.

Crawford (1992) represented a GA-based solution to the n -queens problem. The steps of this solution are as follows. First, a representation is formed where each chromosome is represented as a one-dimension array. The size of the array is n . Each cell represents a column and contains the location of the queen (row number). Second, two parents are randomly selected from the solution pool (possible solution). Third, a specialized crossover is performed; the parents are then used for the k -point crossover with duplicate numbers. A partially mapped crossover fixes this duplicate.

Metaheuristics were compared for solving the n -queens problem (Masehian et al., 2013). These included two single solution-based metaheuristics, tuned simulated annealing (Kirkpatrick et al., 1983) and local search (Korst & Aarts, 1989), and two population-based metaheuristics (two versions of scatter search, Glover, 1977). The effective swap, which is a new variant of the neighborhood generation method, was also proposed. The result is that local search outperforms the other algorithms. Simulated annealing proved to be slow and ineffective by comparison. A biogeography-based optimization (BBO) algorithm was used for the n -queens problem (Habiboghli & Jalali, 2017). BBO demonstrated better efficiency than that of particle swarm optimization–based algorithms but less efficiency than that of a GA-based algorithm for instances of sizes up to 100 queens. The GA was hybridized with a bat algorithm and showed superior performance in comparison with a pure GA and bat algorithms (Al-Gburi et al., 2018). Several other metaheuristics were investigated for the n -queens problem including the intelligent water drops algorithm (Shah-Hosseini, 2008), the modified GA (Heris & Oskoei, 2014), and the cuckoo search algorithm (Sharma & Keswani, 2013).

Berghman et. al. (2009) and Maestro-Montojo et. al. (2013) compared evolutionary algorithms for the Mastermind puzzle, and Bhasin and Singla (2012) solved the n -puzzle problem using a GA. The number of possible initial configurations is $\left(N+1\right)!$. Half of these configurations are solvable, whereas the other half are not solvable. Moreover, the researchers checked whether the configuration was solvable before beginning the search. In fact, they proposed a hybrid GA–iterative deepening algorithm. The Manhattan distance was used to calculate the fitness evaluation and the roulette wheel selection method to select parents. An algorithm called DSolving was proposed for the n -puzzle and showed good performance for instances of up to size 20 (Wan & Li, 2017). Browne and Maire (2010) used evolutionary search to synthesize and evaluate the quality of new combinatorial games, and Brown and Valtchanov (2017) used genetic programming (Koza, 1992) to randomly generate environments and opponents in the Diablo game.

DESCRIPTION OF THE PROPOSED GA

This section explains the algorithm design, including the fitness function, chromosome representation, population initialization, parent selection, crossover, mutation, survivor selection, and the termination condition.

A GA is a type of metaheuristic aimed at finding the optimal solution to a search problem on the basis of a theory of natural selection and evolutionary biology (Sivanandam & Deepa, 2007). Initially, Holland (1992) described his early research as a means of studying adaptive behavior. However, GAs have been considered to be optimization methods (Eiben & Smith, 2003).

Traditionally, GAs have had a structure as shown in Algorithm 1. $numG$ is the generation number given a population of μ individuals that is initialized with random candidate solutions in $numG$ -th generation G_{numG}. Then, fitness is calculated for individuals in G_{numG}. Next, the algorithm goes into an iterative cycle, which begins with the selection of parents making up the intermediary population of G_{numG} individuals. The crossover between individuals and the mutation are performed. The fitness of new offspring is then computed and the next generation is formed accordingly. These steps are iteratively repeated until the termination condition has been achieved. This occurs when the maximum number of iterations is reached or when the results converge (Dorronsoro et al., 2014). A GA is defined using experiment components: chromosome representation, population initialization, fitness function computation, the parent-selection method, crossover and mutation operators, the survivor-selection method, and the termination condition.

Algorithm 1: Simple pseudocode for a GA

1: $numG \leftarrow 0$
2: $G_{numG} \leftarrow initialPopulation(\ )$
3: $Compute\ fitness(G_{numG})$
4: **repeat**
5: $\quad G'_{numG} \leftarrow Selection(G_{numG})$
6: $\quad G''_{numG} \leftarrow Crossover\left(G'_{numG}\right)$
7: $\quad Mutation\left(G''_{numG}\right)$
8: $\quad G''_{numG+1} \leftarrow Computefitness\left(G''_{numG}\right)$
9: **until** population has converged or the maximum number of generations is achieved

Fitness Function

The study was aimed at minimizing the number of rows and columns with odd numbers of tiles as given in (9):

Minimize

$$\sum_{r=1}^{n}((\sum_{c=1}^{n}M_{rc})mod2)+\sum_{c=1}^{n}((\sum_{r=1}^{n}M_{rc})mod\ 2) \tag{9}$$

where:

- n is the dimension of matrix M.
- M is a matrix of size $n \times n$.
- M_{rc} is the cell value of the r^{th} row and c^{th} column.

Chromosome Representation

The representation is a binary matrix of size $n \times n$, where a value of 0 represents a deleted tile and 1 represents a present tile. For example, Figure 2 depicts the encoding of the board in Figure 1(b).

Population Initialization and Fitness Function

The initial population was randomly generated in the study. Specifically, the study started by creating an $n \times n$ grid covered with n^2 tokens. It involved randomly selecting and deleting *m* tokens, where $m \leq n^2$. In addition, (9) was used to evaluate the candidate solution. Thus, a solution to the puzzle has a fitness value equal to zero.

Parent and Survivor Selection

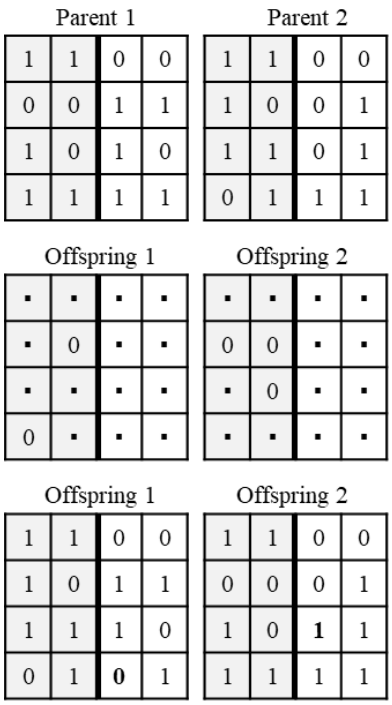
Tournament selection was applied for parent selection in the study. GENITOR was used as the survivor selection method (Eiben & Smith, 2003).

Crossover

Various crossover operators are designed for chromosomes that encode permutation, such as modified order crossover (MOC) and partially mapped crossover (PMX, Davis, 1985; Goldberg & Lingle Jr., 1985). However, none of these operators fits the particularities of the Casse-tête problem because the problem deals with a matrix of binary permutations. Thus, the study involved devising binary modified order crossover (BMOC) and binary partially mapped crossover (BPMX), which have been adapted to this representation.

BMOC (see Figure 3) makes one random vertical split in each parent, copying k zeros from the left segment of the first parent (where k is the number of zeros in the left segment of the first parent). Then, it copies the remaining $m-k$ zeros from the right segment of the second parent (where m is the total number of deleted tokens). If the number of zeros in the offspring is less than m, the missed zeros are randomly added to the offspring. The same procedure is applied, with parent numbers being switched, to produce the second offspring. Pseudocode is shown in Algorithm 2. P_1 and P_2

Figure 3. BMOC



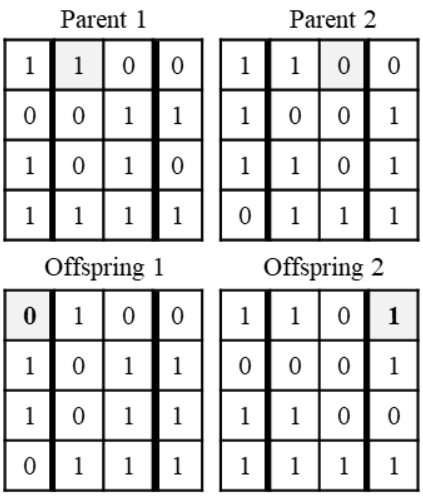
refer to parent 1 and parent 2, respectively. Similarly, O_1 and O_2 refer to offspring 1 and offspring 2, respectively.

Algorithm 2: BMOC algorithm

```
Input: n×n matrices P1 and P2
Output: n×n matrices O1 and O2
Initialize all cells of O1 and O2 with 1.
Choose one random crossover point a in P1 and P2, where 2 ≤ a ≤ n.
Copy zeros from left segment of P1 to O2 using the same order
    in P1 and similarly from P2 to O1.
Calculate number of zeros nZero in O2 and similarly in O1
Copy remaining m-nZero from right segment of P2 to O2 using
    the same order of P2 and similarly from P1 to O1
Calculate number of zeros nZero in O2 and similarly in O1
if nZero < m
Add m − nZero randomly to O2 and similarly to O1.
endif
```

BPMX (see Algorithm 3 and Figure 4) should have two crossover points at each parent to maintain the in-between segment. It copies in-between crossover points to the offspring, compares the number of zeros of offspring 1 and offspring 2 if they are not equal, and adds a missed zero to the one that has the least number of zeros in a particular position.

Figure 4. BPMX



Algorithm 3: BPMX algorithm

```
1:  Input: n×n matrices P1 and P2
2:  Output: n×n matrices O1 and O2
3:  Initialize all cells of O1 and O2 with 1.
4:  Choose two random crossover points and in and,
    where 2 ≤ a < b ≤ n - 1
5:  Copy values inside of portion (between a and b) from P1 to O1
    and from P2 to O2
6:  Calculate number of zeros nZero1 and nZero2 in O1 and O2
7:  Copy values outside of portion (between a and b) from P1 to
    O2 and from P2 to O1
8:  if nZero1 != nZero2
9:     Replace extra zeros in P1 with ones in O1, where the
       positions of replaced ones in O1 are the positions of zeros
       in P2 and similarly in O2
10: endif
```

Mutation

Mutation alternatives for permutation representation, such as swap and inversion (Eiben & Smith, 2003), are all designed for linear structures. Thus, they cannot be applied directly. In this research study, a modification of the mutation operators for linear permutation representation is needed; it enables the mutation of individuals represented as matrices. In the BInversion method, a subset is chosen at random and then reversed in the chromosome. This method is illustrated in Figure 5, when $n = 4$ and $m = 8$. In the BSwap method, two positions are randomly selected and swapped. Figure 6 shows an example of a BSwap.

Termination Condition

The termination condition is satisfied when a workable solution that maintains that the number of tiles is even in each row and column on the board is found or when the execution reaches the maximum number of generations.

EXPERIMENTAL METHODOLOGY

For the purpose of achieving the research goal, a GA-based solution in which the dataset is artificially generated was proposed in this study. The solution consisted of μ individuals, with

Figure 5. Example of BInversion mutation

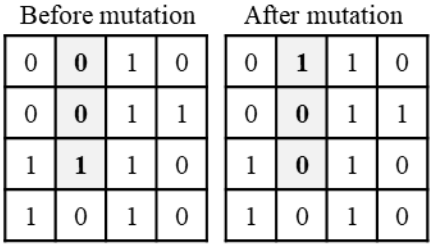
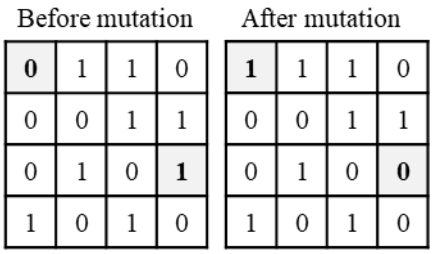


Figure 6. Example of BSwap mutation



each being generated randomly. The values of n are in (10, 20, 50, 100) based on Crawford (1992). For each value of n, a set of three values of m representing a small, medium, and large number of tiles is randomly removed. Based on an ad hoc design, 25% of n is considered to be a small value of m, 50% of n is medium, and 75% of n is large. In addition, the study involved testing the models using only the values of n and m that generated a workable board. Stochastic algorithms were used; therefore, each run was repeated 10 times for each pair $\left(n,m\right)$, and the average of these runs was reported. Table 1 shows the parameter values, namely the population size $\left(\mu\right)$, maximum number of generations $\left(numG\right)$, tournament size $\left(q\right)$, crossover probability $\left(p_c\right)$, and mutation probability $\left(p_m\right)$.

For the purpose of identifying the best combination of crossover-mutation operators, experiments were performed with four possible combinations separately:

Study 1: Uses BPMX crossover with BSwap mutation.
Study 2: Uses BPMX crossover with BInversion mutation.
Study 3: Uses BMOC crossover with BSwap mutation.
Study 4: Uses BMOC crossover with BInversion mutation.

In the study, the best-performing model among the four studies of GA-based solutions was identified and compared with the SA-based solution to the $CT_n\left(m\right)$ problem (Study 5). Note that using SA to solve the Casse-tête problem had never before been conducted. This study also involved designing and implementing the SA-based solution to the $CT_n\left(m\right)$ problem as follows. The same representation defined for GA was used for the state. The initial state was generated randomly. The study followed random swap as a move operator as well as the geometric cooling function, with α equaling 0.95 and n being the initial temperature (Masehian et al., 2013). For determining the equilibrium state, the static strategy was followed (Talbi, 2009), with y equaling 0.25 and the final temperature equaling 0.001. The termination condition was the same as that used for the GA model.

Next, for the purpose of measuring performance, the success rate (*SR*) (the percentage of successful runs over all runs) and the number of fitness function evaluations (FEs), which indicates

Table 1. Values of parameters for GA experiments

Parameters	μ	$numG$	q	p_c	p_m
Value	1,000 (for =10, 20, 50) 10,000 (for =100)	10,000	10	0.8	0.01

the required amount of computing, were used in the study (Ville, 2013). In addition, the average elapsed time $\left(Time\right)$ was computed over all runs (tic toc MATLAB functions). Furthermore, a MacBook Pro 2.9 GHz processor and an 8-GB RAM MacOS 10.12 Sierra were used. Matlab R2016b update 4 was additionally used for programming, and IBM SPSS Statistics Base Package was for statistical analysis.

RESULTS

The study involved implementing four GA models that were obtained from the four crossover and mutation combinations defined in the experimental design section. A comparison of mutation operators is presented in combination with the BPMX crossover operator. Next, a comparison of mutation operators is presented in combination with the BMOC crossover. The comparison of crossover operators is then shown in combination with the BSwap mutation operator. The best GA model is evaluated against the SA model. Finally, a discussion of the results is presented. The results of the four GA studies are tabulated in Appendix A. Meanwhile, Appendix B tabulates the results for Study 5.

Comparison of Mutation Operators in Combination With BPMX Crossover Operator

This subsection uses Study 1 and Study 2 to compare the performance of the two proposed mutation operators: BSwap and BInversion. BPMX was the crossover operator in both studies. All parameters were identical in these studies (as defined in Table 1). Study 1 used the BSwap mutation operator, whereas Study 2 used the BInversion mutation operator. Both studies involved experimenting with board sizes n in (10, 20, 50). For each value of n, three m values were experimented with as defined earlier. Figure 7 shows the FEs obtained over all runs for each combination of $\left(n,m\right)$. Figure 8 shows the SRs for all sizes. As can be seen in Figure 8, no solution was obtained in Study 2 for the $\left(n,m\right)$ combinations of (20, 100) and (20, 200). Also, the GA in Study 2 did not generate a solution for any of the boards of dimension $n=50$.

Figure 7. Number of FEs obtained in Study 1 and Study 2 over the course of all runs for each board size n in (10, 20, 50) and in combination with (small, medium, large) percentage of removed tokens m

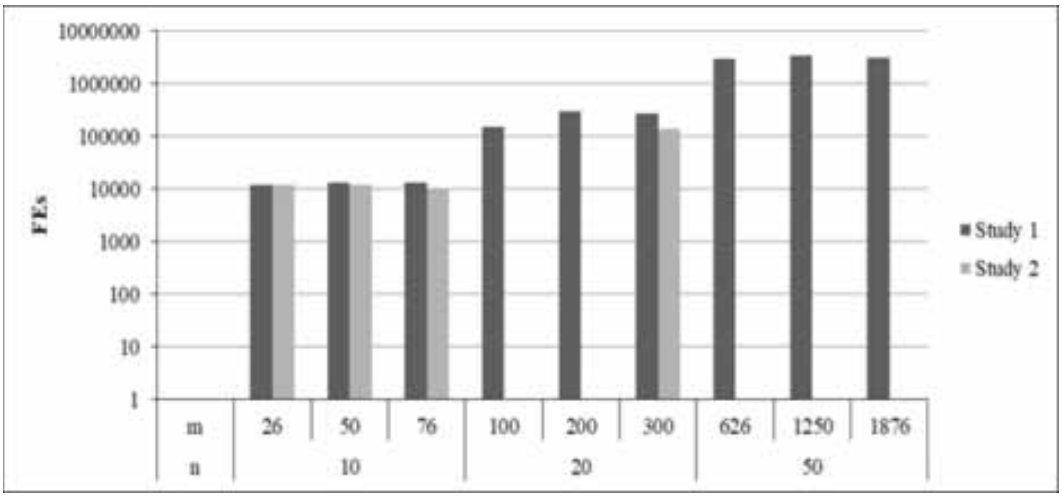
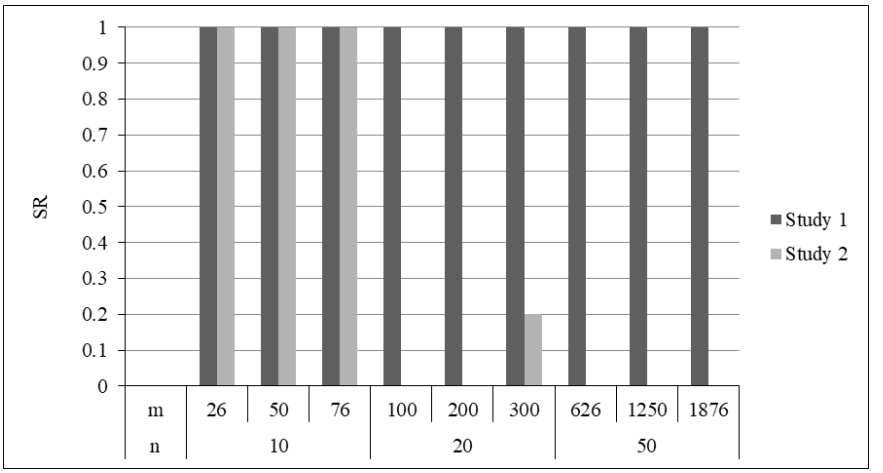


Figure 8. SRs obtained in Study 1 and Study 2 over the course of all runs for each size n in (10, 20, 50) and in combination with (small, medium, large) percentage of removed tokens m



Performance Analysis

Study 1:

- **Effectiveness:** The SR was 100% in every case.
- **Efficiency:** For FEs, when $n = 10, n = 20$, and $n = 50$, the most efficient result was that with the smallest value of m .

Study 2:

- **Effectiveness:** The SR was 100% only when $n = 10$. It was 20% at $n = 20$ and $m = 20$. Otherwise, it was 0%.
- **Efficiency:** For FEs, when $n = 10$ and $n = 20$, the most efficient result was that with the largest value of m . When $n = 50$, the values of FEs were 0 for all values of m .

The outcomes of Studies 1 and 2 showed that the number of FEs was better in Study 2 (Figure 7). However, Study 1 had a better SR. Study 1 was more effective than Study 2 was (Figure 8); thus, we concluded that BSwap mutation performs better than BInversion does in combination with the BPMX crossover operator.

Comparison of Mutation Operators in Combination With BMOC Crossover Operator

This subsection uses Study 3 and Study 4 to compare the performance of the two proposed mutation operators: BSwap and BInversion. However, BMOC was used as the crossover operator in both studies. Table 1 shows all parameters and operators used in these studies. The sizes for board n are in (10, 20, 50). Also, for each value n , the studies used three different values of m . Figure 9 shows the FEs obtained over all runs for each combination of $\left(n, m\right)$. Additionally, Figure 10 shows the SRs for all sizes.

Figure 9 shows that the models in Study 4 did not generate a solution at $n = 50$. As can be seen in Figure 10, when $n = 20$, the SR decreased when m increased in Study 4.

Performance Analysis

Study 3:

- **Effectiveness:** The SR was 100% in every case.
- **Efficiency:** Based on FEs, when $n = 10$, the most efficient result was that with the smallest value of m , where $n = 20$ and $n = 50$ were medium values of m.

Figure 9. Number of FEs obtained in Study 3 and Study 4 over the course of all runs for each board size n with different combinations of m

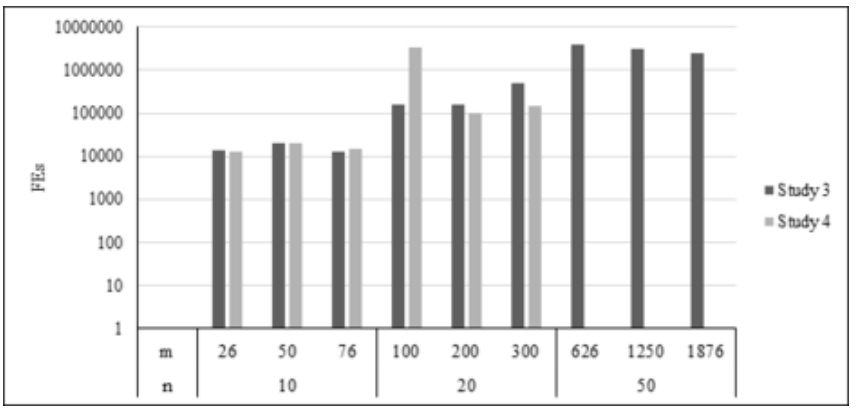
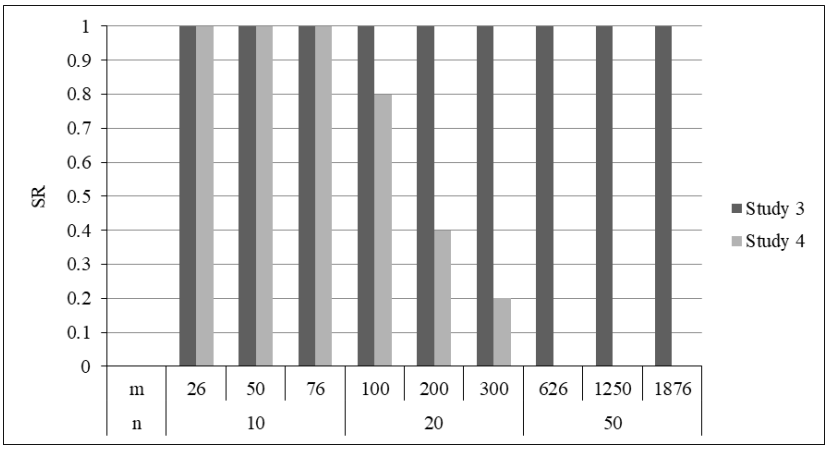


Figure 10. SRs obtained in Study 1 and Study 2 over the course of all runs for each size n in (10, 20, 50) and in combination with (small, medium, large) percentage of removed tokens *m*



Study 4:

- **Effectiveness:** The SR was 100% only when $n = 10$. For $n = 20$, it was 80% for the smallest value of m, 40% for the medium value of m, and 20% for the largest value of m. When $n = 50$, the SR was 0% because no solutions were found for all indicated values of m.
- **Efficiency:** For FEs, when $n = 10$, the most efficient result was that with the smallest value of m. The most efficient result where $n = 20$ was that obtained with medium values of m.

The number of FEs was better in Study 3 (Figure 9). Additionally, Study 3 had a better SR (Figure 10). Study 4 was more effective than Study 3 was; thus, we conclude that BSwap mutation performs more powerfully than BInversion does in combination with the BMOC crossover operator.

Comparison of Crossover Operators in Combination With BSwap Mutation Operator

Based on earlier comparisons, the results of Study 1 outperformed Study 2. Similarly, the results for Study 3 were better than those of Study 4. In this step, we compare the best performing study from

both sets: Study 1 and Study 3. Board sizes n were in (10, 20, 50, 100). For each board size, three corresponding percentages m of the number of tokens were removed. In these studies, we reported the elapsed time in seconds for board sizes n in (10, 20, 50). For $n=100$, the time was not reported. Figure 11 shows the FEs for n in (10, 20, 50, 100) over all runs.

The SRs obtained in both Study 1 and Study 3 were 100%. Both studies considered FEs, which was better for comparison with Study 5. The two studies were similar, so a Wilcoxon signed-rank test was conducted to choose the best. No substantive differences in FE values were found ($p=0.95$); therefore, the average over all instances was used to obtain the following scores: Study 1: 1,140,980 and Study 3: 1,144,602 (Appendix A).

The elapsed time was used as an additional performance measure. Figure 12 shows the elapsed time for each study. In size 10, Study 1 was faster than Study 2 was in the two combinations of (10, 26) and (10, 50). In size 20, they were similar. In size 50, Study 3 took less time than Study 1 did in all cases.

Figure 11. Number of FEs obtained in Study 1 and Study 3 over the course of all runs for each combination of board sizes n (10, 20, 50,100) in combination with (small, medium, large) percentage of removed tokens m

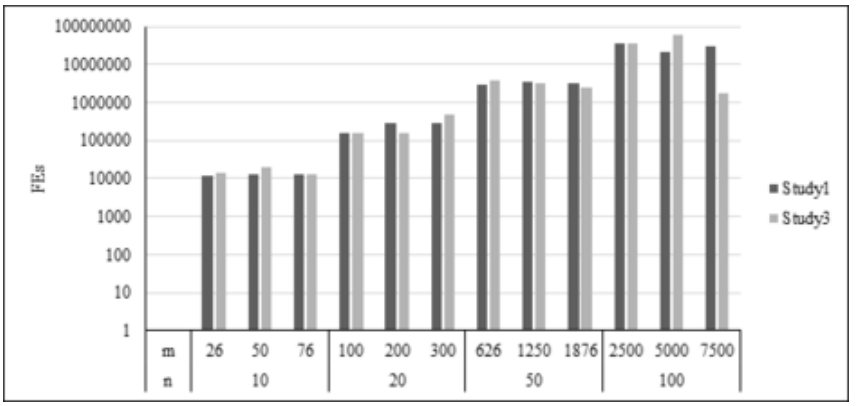
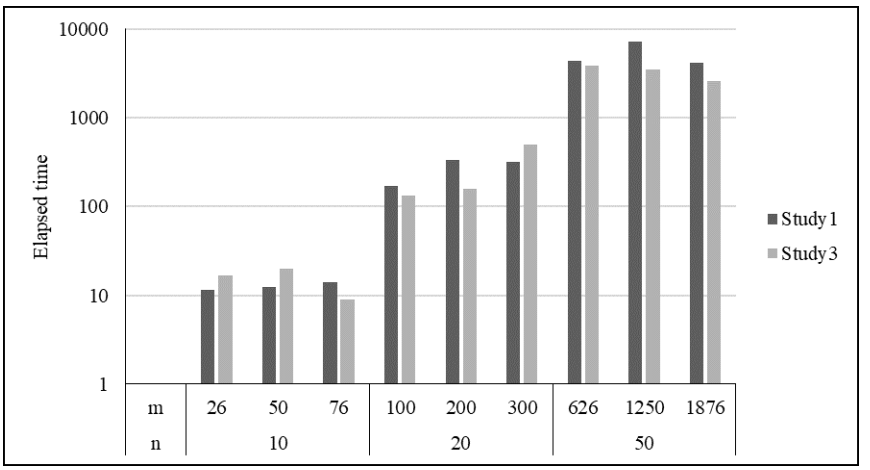


Figure 12. Time obtained in seconds for Study 1 and Study 3 over all runs for each combination of board sizes n (10, 20, 50) with three values of m



Based on the FE metric, it was concluded that BSwap mutation in combination with the BPMX crossover operator performs more efficiently than BSwap does in combination with the BMOC crossover operator. Moreover, the combination of BSwap mutation and the BMOC crossover operator takes less time compared with the combination of BSwap mutation and the BPMX crossover operator.

Evaluation of Best GA Model Compared With SA Model

Based on earlier comparisons, the results of Study 1 were better than those of Study 3. This subsection compares the SA-based solution to the Casse-tête problem (Study 5) with the GA model in Study 1. Board sizes n were in (10, 20, 50, 100). For each board size, the study involved testing three values of m (small, medium, large) and removing the number of tokens. The measure was performed between two models via calculating SRs and FEs. Figure 13 shows the FEs for n in (10, 20, 50, 100) over all runs.

The study used the elapsed time as an additional performance measure for board sizes n in (10, 20, 50). Figure 14 shows the elapsed time for each study over all runs for each combination. In all sizes, GA took more time than SA did.

Performance Analysis

GA-Based Model (Study 1):

- **Effectiveness:** The SR was 100% in every case.
- **Efficiency:** For FEs, when $n =$ (10, 20, 50, 100), the most efficient result was that with the smallest value of m. Based on the elapsed time, when $n = 10$ and $n = 20$, the smallest value of m yielded the best result. When $n = 50$, the largest value of m yielded the best result.

SA-Based Model (Study 5):

- **Effectiveness:** The SRs were 100% for board sizes of up to 50.
- **Efficiency:** Based on the FEs, when $n = 10$ and $n = 50$, the most efficient result was that with the largest value of m. When $n = 20$, the most efficient result was that with the medium value of m. When $n = 100$, the output was obtained only for $m = 2500$. Moreover, based on the elapsed time, $n = \left(10, 20, 50\right)$ returned the best results in less time with the largest value of m.

Figure 13. Number of FEs obtained in GA (Study 1) and SA (Study 5) over all runs for each board size n in (10, 20, 50, 100) and in combination with (small, medium, large) percentage of removed tokens m

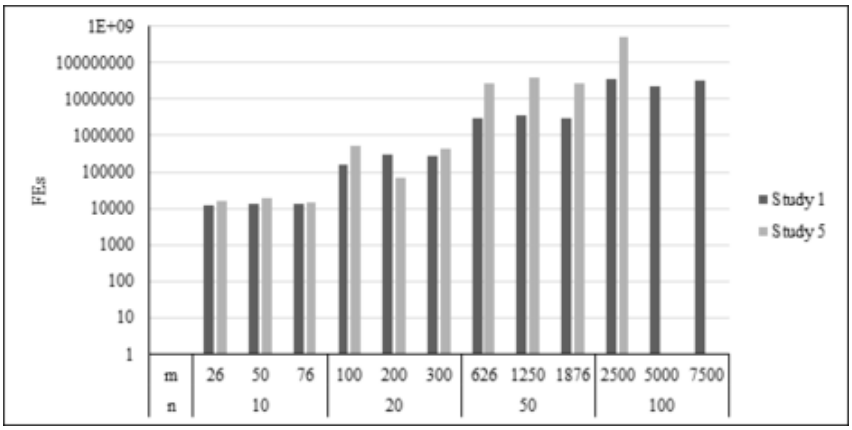
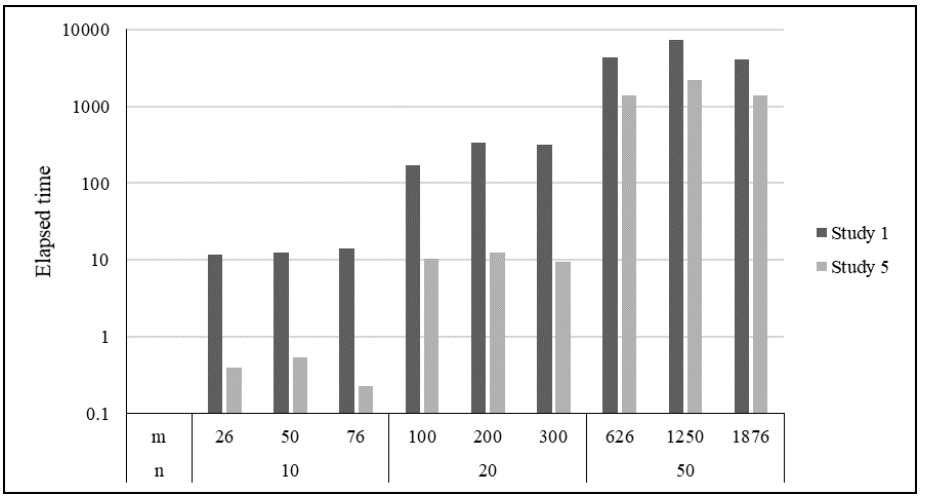


Figure 14. Time obtained in GA (Study 1) and SA (Study 5) over all runs for each combination of board sizes n (10, 20, 50) with three different values of m



DISCUSSION

The most effective studies were those using BSwap as the mutation. For example, in Studies 1 and 3, the Casse-tête problem was solved even when n increased. The studies using BInversion failed to produce solutions with larger values of n. A suitable explanation is that BSwap makes small changes, whereas BInversion selects a whole section of the board at random and reverses the chromosome, which makes larger changes.

Regarding the crossover operators, both BPMX and BMOC could solve Casse-tête problem. No clear difference was found in terms of the FEs. Therefore, the study used a Wilcoxon signed-rank test as a paired difference test. The results showed minimal differences between the two operators of the crossover; therefore, we chose BPMX based on the average of FEs because it required less computing than BMOC did in most cases. Similarities between the results of the two crossover algorithms might be due to the crossover operator's aim of taking information from both parents. Thus, the produced offspring were similar.

The final step involved evaluating the selected GA model against the SA model, which is critical because it helps with determining which is more effective and efficient based on the SRs and FEs. Both models could solve the problem and have 100% SRs for board sizes of up to 50. However, as in Figure 13, the SA model requires more computing than the GA model does to produce a workable solution. Furthermore, Figure 14 shows that the SA model consumes less time. However, the GA model is more easily scaled to larger problem sizes. In fact, for board sizes $n = 100$, only the GA was able to find a workable configuration for all examined instances. A possible explanation is that the GA has more to handle than the SA does, including a whole population and crossover, as well as the selection of parents and survivors. More trials were conducted with the SA according to the FEs. Consequently, the SA allocated greater focus to generating and trying new candidate solutions (with lower exploration ability compared with GA). The study revealed that the difference in FEs among the SA and GA models was statistically significant based on the Wilcoxon signed-rank test (p =0.038).

CONCLUSION

The Casse-tête problem is an interesting puzzle that was not attempted prior to this study. The study investigated which chromosome representation is suitable for the problem and found that a

combination of three representations were necessary: (1) binary, (2) permutation, and (3) matrix. The study proposed crossover methods to fit the problem (BPMX and BMOC) as well as mutation methods (BSwap and BInversion). The fitness evaluation penalized individuals with odd numbers of tiles in rows or columns. The dataset was artificially generated for each run. The study generated a new random initial population. It then focused on comparing the GA with the rival algorithm (the SA) by using different performance metrics and relying on solid statistical tools. The proposed model outperformed the simulated annealing-based model for the Casse-tête problem in terms of efficiency (FEs) and effectiveness (SRs).

The research provides new evidence that the Casse-tête problem can be solved in a reasonable amount of time using GAs for further enquiry. It also showed results for puzzle sizes of up to 100. As the SA showed better efficiency for smaller instances, our next step would be to implement a mobile application version of the puzzle for small instances using the SA. For instances larger than size ($n = 50$), we intend to try other parameters for the SA, such as using a different cooling schedule, equilibrium state condition, or initial temperature to improve the scalability of the SA model. It is also possible to examine this using moves other than random swap that are suitable for permutation problems. Future research may investigate the design using metaheuristics, such as GRASP and tabu search, as well.

REFERENCES

Al-Gburi, A. F., Naim, S., & Boraik, A. N. (2018). Hybridization of Bat and Genetic Algorithm to Solve N-Queens Problem. *Bulletin of Electrical Engineering and Informatics*, *7*(4), 626–632. doi:10.11591/eei.v7i4.1351

Berghman, L., Goossens, D., & Leus, R. (2009). Efficient solutions for Mastermind using genetic algorithms. *Computers & Operations Research*, *36*(6), 1880–1885. doi:10.1016/j.cor.2008.06.004

Bhasin, H., & Singla, N. (2012). Genetic based Algorithm for N - Puzzle Problem. *International Journal of Computers and Applications*, *51*(22), 44–50. doi:10.5120/8347-1894

Brown, J., & Valtchanov, V. (2017). Tile Based Genetic Programming Generation for Diablo-like games. Academic Press.

Browne, C., & Maire, F. (2010). Evolutionary Game Design. *Computational Intelligence and AI in Games. IEEE Transactions On*, *2*(1), 1–16. doi:10.1109/TCIAIG.2010.2041928

Cook, S. A. (1971). The complexity of theorem-proving procedures. *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, 151–158. doi:10.1145/800157.805047

Crawford, K. D. (1992). Solving the N-queens Problem Using Genetic Algorithms. In H. Berghel, E. Deaton, G. E. Hedrick, D. Roach, & R. L. Wainwright (Eds.), *SAC* (pp. 1039–1047). ACM. doi:10.1145/130069.130128

Davis, L. (1985). Applying Adaptive Algorithms to Epistatic Domains. *Proceedings of the International Joint Conference on Artificial Intelligence*, *1*, 162–164. doi:10.1.1.76.4906

Dorronsoro, B., Ruiz, P., Danoy, G., Pigné, Y., & Bouvry, P. (2014). *Evolutionary Algorithms for Mobile Ad Hoc Networks* (1st ed.). Wiley Publishing. doi:10.1002/9781118833209

Eiben, A. E., & Smith, J. E. (2003). *Introduction to Evolutionary Computing* (2nd ed.). Springer Publishing Company. doi:10.1007/978-3-662-05094-1

Glover, F. (1977). Heuristics for integer programming using surrogate constraints. *Decision Sciences*, *8*(1), 156–166. Advance online publication. doi:10.1111/j.1540-5915.1977.tb01074.x

Goldberg, D. E., & Lingle, R. Jr. (1985). AllelesLociand the Traveling Salesman Problem. *Proceedings of the 1st International Conference on Genetic Algorithms*, 154–159.

Habiboghli, A., & Jalali, T. (2017). A Solution to the N-Queens Problem Using Biogeography-Based Optimization. *International Journal of Interactive Multimedia and Artificial Intelligence*, *4*(4), 22–26. doi:10.9781/ijimai.2017.443

Heris, J. E. A., & Oskoei, M. A. (2014). Modified genetic algorithm for solving n-queens problem. *2014 Iranian Conference on Intelligent Systems (ICIS)*, 1–5. doi:10.1109/IranianCIS.2014.6802550

Holland, J. H. (1992). Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. University of Michigan Press. doi:10.1137/1018105

Katoch, S., Chauhan, S. S., & Kimar, V. (2020). A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications, 80*(2021), 8091–8126.

Kirkpatrick, S., Gelatt, C. D. Jr, & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, *220*(4598), 671–680. doi:10.1126/science.220.4598.671 PMID:17813860

Kordemsky, B. (1972). *The Moscow Puzzles*. Charles Scribner's Sons.

Korst, J. H. M., & Aarts, E. H. L. (1989). Combinatorial optimization on a Boltzmann machine. *Journal of Parallel and Distributed Computing*, *6*(2), 331–357. Advance online publication. doi:10.1016/0743-7315(89)90064-6

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press.

Maestro-Montojo, J., Merelo, J. J., & Salcedo-Sanz, S. (2013). Comparing Evolutionary Algorithms to Solve the Game of MasterMind. *Applications of Evolutionary Computation. Lecture Notes in Computer Science*, *7835*, 304–313. doi:10.1007/978-3-642-37192-9_31

Masehian, E., Akbaripour, H., & Mohabbati-Kalejahi, N. (2013). Landscape analysis and efficient metaheuristics for solving the n-queens problem. *Computational Optimization and Applications*, *56*(3), 735–764. doi:10.1007/s10589-013-9578-z

McCall, J. (2005). Genetic Algorithms for Modelling and Optimisation. *Journal of Computational and Applied Mathematics*, *184*(1), 205–222. doi:10.1016/j.cam.2004.07.034

Shah-Hosseini, H. (2008). The intelligent water drops algorithm: A nature-inspired swarm-based optimization algorithm. *International Journal of Bio-inspired Computation*, *1*(1–2), 71–79.

Sharma, R. G., & Keswani, B. (2013). Implementation of n-Queens Puzzle using Meta-heuristic algorithm (Cuckoo Search). *International Journal of Latest Trends in Engineering and Technology*, *2*(3), 343–347.

Sivanandam, S. N., & Deepa, S. N. (2007). *Introduction to Genetic Algorithms* (1st ed.). Springer Publishing Company., doi:10.1007/978-3-540-73190-0

Sturm, T. (2009). Sudoku and AI. *IC-AI*, 910–911.

Talbi, E. G. (2009). Metaheuristics: From Design to Implementation. Metaheuristics: From Design to Implementation. doi:10.1002/9780470496916

Uke, N. J., & Thool, R. C. (2011). Playfully teaching artificial intelligence by implementing games to undergraduates. *Proceedings of the International Conference & Workshop on Emerging Trends in Technology*, 1377. doi:10.1145/1980022.1980408

Ville, G. (2013). *An optimal mastermind (4, 7) strategy and more results in the expected case.* ArXiv Preprint ArXiv:1305.1010.

Wan, G., & Li, R. (2017). DSolving: A novel and efficient intelligentalgorithm for large-scale sliding puzzles. *Journal of Experimental & Theoretical Artificial Intelligence*, *29*(4), 809–822. doi:10.1080/0952813X.2016.1259270

Yannakakis, G. N., & Togelius, J. (2018). *Artificial intelligence and games* (Vol. 2). Springer. doi:10.1007/978-3-319-63519-4

APPENDIX A

The detailed results for Study 1 through Study 4 are shown in Table 2, including the number of fitness evaluations (FEs) success rates (SRs), and computational time in seconds, averaged over all runs for each combination of board sizes n (10, 20, 50) with three different values of m.

Table 2. Comparison among GA-based studies

n	*m*	Study 1			Study 2		Study 3			Study 4	
		FEs	**SR**	**Average time (s)**	**FEs**	**SR**	**FEs**	**SR**	**Average time (s)**	**FEs**	**SR**
10	26	12002	100%	11.64	12002	100%	14202	100%	16.58	12602	100%
	50	13402	100%	12.34	12002	100%	20602	100%	19.84	20202	100%
	76	13202	100%	13.94	10602	100%	13201.8	100%	8.95	15402	100%
20	100	155602	100%	170.76	0	0%	162002	100%	133.55	3361252	80%
	200	297402	100%	337.46	0	0%	155002	100%	160.2	104002	40%
	300	275602	100%	317.75	136002	20%	497402	100%	504.38	145002	20%
50	626	2989802	100%	4367.39	0	0%	3809402	100%	3827.17	0	0%
	1250	3437402	100%	7298.35	0	0%	3087202	100%	3514.15	0	0%
	1876	3074402	100%	4139.89	0	0%	2542402	100%	2561.71	0	0%
Average		1140979.78		1852.17	42652		1144601.9		1194.0589	609743.67	
Standard deviation		1528037.83		2712.20	62236.83		1541655.9		1621.17	1349069.1	

APPENDIX B

The detailed results for Study 5 are shown in Table 3, including the number of fitness evaluations (FEs) success rates (SRs), and computational time in seconds, averaged over all runs for each combination of board sizes n (10, 20, 50) with three different values of m.

Table 3. Results for the simulated annealing (SA) Study

n	*m*	Study 5		
		FEs	**SR**	**Average time (s)**
10	26	15964.2	100%	0.4
	50	18413.2	100%	0.54
	76	15216.4	100%	0.23
20	100	531060.8	100%	10.47
	200	69469.4	100%	12.6
	300	426826.58	100%	9.54
50	626	28258587.6	100%	1374.19
	1250	37389266.8	100%	2175.74
	1876	28164018	100%	1381.83
Average		10543202.55		551.73
Standard deviation		15770838.33		850.91

Sarab AlMuhaideb is an Assistant Professor in Computer Science at King Saud University. In 1999, she received a Master of Science in Computer Science from Stevens Institute of Technology, New Jersey. She obtained her PhD in Computer Science from King Saud University in 2014. In 2017, she was certified an IBM Predictive Analytics Modeler Mastery Award. Dr. AlMuhaideb obtained several honors including Shiekh Rashid Al-Maktoom Award for Scientific Achievement (UAE, 1991 and 1999), Scientific Achievement Certificate from the UAE Embassy (USA, 1998), and Computer Science Outstanding Faculty Award in recognition for Outstanding Teaching (Prince Sultan University, 2016). Her research interests include Computational Intelligence, Machine Learning, Metaheuristics, Hybrid Methods, and Optimization Methods.

Atheer Saad Alrosayes is analyzing and developing (web and mobile) applications, working at KAUH hospital with a health care database. As a developer, she is responsible for developing, testing, systems modifications, and enhancements the applications, and dealing with a variety of programming languages and environments. Atheer has B.Sc in computer sciences from the KSU (2019).