AI-Based Puzzle Game Solver

Yaqoob Yaghoubi
https://www.linkedin.com/in/yaqoobs/

***Abstract*—Puzzle games such as Sudoku have been widely investigated for a variety of purposes such as evaluating optimization algorithms. They present great computational challenges. This research aims to demonstrate the application of the Skyscraper puzzles as a benchmark test bed for Genetic Algorithms. This type of puzzle has the potential to be further investigated for several optimization algorithms which offer interesting challenges. A puzzle simulator with fitness function using Java is developed with the help of off-the-shelf convention Genetic Algorithm which talored and adapted to the problem presented through the simulator to the GA. It has been found that Skyscraper has great potential in challenging these types of optimization algorithms. Increasing the puzzle size, GA has difficulty finding optimal solutions. This research contributes to the literature on algorithm evaluation and optimization using puzzle games.**

***Index Terms*—Genetic Algorithm, Skyscraper Puzzle, Puzzle Solver**

I. INTRODUCTION

FOR a considerable amount of time, puzzle games have been a prominent entity in computational science. Japanese puzzles which are considered a subclass of these puzzles, have been around since the 1700s and the famous one is Sudoku, offering a unique blend of computational challenge, and educational benefits [1], [2]. These puzzles seem easy compared to board games such as chess, however, they present a rich set of challenges that could be taken as an advantage to evaluate various algorithms [3]. Add a bit to the skyscraper description. The significance of puzzle games extends to the realm of Artificial Intelligence (AI) where they were served for several research purposes. There are several research has been done on this topic, using puzzle games, notably Sudoku as a benchmark set for AI algorithms, procedural content generation, AI-assisted game design and other research areas [4], [5], [6], [7]. Researchers utilized puzzle games to test problem-solving capabilities and pattern recognition of AI algorithms that are transferable to the real world such as strategic reasoning and probabilistic reasoning. Puzzle games provide opportunities to isolate and practice these skills which makes them valuable training grounds for AI systems [8]. Within this domain, Sudoku has predominantly gained attention [7], [9], [10], [11], [12], with limited exploration into alternative puzzles such as Skyscraper. Consequently, for this research work, the Skyscraper puzzle is chosen as a novel problem for assessing AI algorithms which in this research is Genetic Algorithm. Skyscraper puzzle is an example of a Japanese puzzle which is a subclass of puzzle games used in this project to apply a genetic algorithm to solve the puzzle and assess its efficiency and effectiveness. Skyscraper is a Latin-Square-driven puzzle [6], [13]. A Latin square is an *n*x*n* array, in this case, is the size of the puzzle i.e. 4x4, 5x5 etc, filled with n different digits each occurring exactly once in each row and exactly once in each column [14], [15]. The puzzle has been proven to be NP-complete [5], [6], meaning that a solution to this puzzle can be verified in polynomial time, but there is no known to solve the problem in polynomial time.

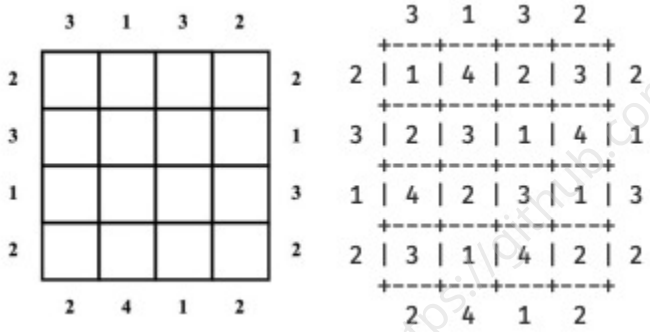

Fig. 1. Structure of a Skyscraper puzzle instance (left), and its solution from the solver (right)

In the simplest form of the puzzle, it began with an empty *n* x *n* grid (or cells) with numbers between 1 to n on each edge referred to as clues that determine how many skyscrapers can

This paragraph of the first footnote will contain the date on which you submitted your paper for review, which is populated by IEEE. It is IEEE style to display support information, including sponsor and financial support acknowledgment, here and not in an acknowledgment section at the end of the article. For example, "This work was supported in part by the U.S. Department of Commerce under Grant 123456." The name of the corresponding author appears after the financial information, e.g. *(Corresponding author: Second B. Author)*. Here you may also indicate if authors contributed equally or if there are co-first authors.

The next few paragraphs should contain the authors' current affiliations, including current address and e-mail. For example, First A. Author is with the National Institute of Standards and Technology, Boulder, CO 80305 USA (e-mail: author@ boulder.nist.gov).

Second B. Author Jr. was with Rice University, Houston, TX 77005 USA. He is now with the Department of Physics, Colorado State University, Fort Collins, CO 80523 USA (e-mail: author@lamar.colostate.edu).

Third C. Author is with the Electrical Engineering Department, University of Colorado, Boulder, CO 80309 USA, on leave from the National Research Institute for Metals, Tsukuba 305-0047, Japan (e-mail: author@nrim.go.jp).

Mentions of supplemental materials and animal/human rights statements can be included here.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org

be seen from that direction. Fig. 1 (left) illustrates the simplest puzzle instance. In this puzzle, for a natural number n, where $[n] = \{1,\ldots,n\}$, there is an nxn grid of cells, along with edge clues in $[n]$. At most, there are 4n clues. The rule for the puzzle is to fill all n^2 cells with integers in $[n]$ so that:

- The integers altogether from an nxn puzzle satisfy the Latin square (i.e. in each row and column every integer in $[n]$ appears exactly once)
- The edge constraints are satisfied.

The edge constraints determine the number of buildings that can be seen from each direction of the column and row. Suppose that each digit represents the height of each building. For the edge constrain *b*, one should exactly see b building in from the row or column where number be is stated as constrain. The point is that the lower building places behind higher buildings are not visible. In Fig. 1 (right), a complete solution to the instance of Fig. 1 (left) is illustrated.

The research goal is to explore the application of AI in puzzle games and establish a benchmark of logical computation for future tests and comparative evaluation of the chosen puzzle solver. The investigation is on a puzzle type that has not been previously attempted with a Genetic Algorithm, in this case, is a skyscraper. It aims to assess the efficiency and effectiveness of GA in solving a skyscraper puzzle thereby enhancing puzzle-solving strategies. The research contributes to a deeper understanding of how GA can be optimized and applied to complex problem-solving tasks.

The scope is delimited to the evaluation of a GA-based solver in a Java-based simulator. To achieve this goal, a simulator will be developed using Java which implements the game rules initializes the instance of the game and applies the algorithm to solve it.

A combination of GA with a heuristic approach was implemented to effectively solve the skyscraper puzzle. The foundation of methodology lies in the creation of the puzzle simulation including puzzle rules and a validation function that serves as a Fitness function to the GA using Java, facilitating the evaluation of puzzle instances. A structured software development life cycle is followed:

1. Design: This phase includes defining the basic functionality of the puzzle, fitness functions, and performance metrics.
2. Development: Includes the implementation of the full puzzle simulation and GA integration to apply the algorithm to solve various scale puzzle instances.
3. Testing: Includes functionality and performance testing. Multiple instances of skyscrapers with different sizes are fed to the software which serves as proof of concept.

The GA implemented in this study will be sourced as an off-the-shelf solution from established literature while tailoring it to the specific requirements and constraints of the puzzle solver framework. Subsequently refined for optimization for compatibility with the existing puzzle-solving framework utilized in the project. The final phase of the research involves cross-comparing the performance of the proposed methodology in solving the skyscraper puzzle in terms of both solution quality and time taken to find the solution with other available literature that is done within the same programming language (Java). The rest of this dissertation is structured as follows. Section II includes the literature on AI algorithms applied to puzzle games and closely related work related to the application of various algorithms on skyscrapers. Section III details the design, development, and testing phase of the research approach and methodology, including the implementation of GA. In section IV the experimental findings from applying the GA to the skyscraper which includes the performance metrics and comparative analysis will be presented. The effectiveness of the GA approach is discussed in section V which interprets the results and discusses the implications. In section VI a summary of research findings is presented and contributions along with direction suggestions for future research.

II. Literature Review

There have been investigations of applying several algorithms to puzzle games to tackle challenges presented by puzzles such as Sudoku, Futoshiki, and Hashi [2], [7], [9], [11]. The intersection of AI and games has historically driven the progression of the development of artificial intelligence, in which puzzle games provided a platform for creativity and evaluation. Which leads to advancement in AI improvement in game design and user experience [16]. Authors in [4] provide a comprehensive overview of the field of Artificial Intelligence and Computational Intelligence in games, presented with the core research areas and their interactions. The papers aim to improve understanding of the current nature of game AI/CI research as well as to identify potential future research projects and unexplored subfields. Namely, the author categorises game AI/CI into ten development areas of nonplayer character (NPC) behaviour learning, procedural content generation, search and planning, and games as AI benchmarks, to enumerate but a few instances. In this project, the focus is specifically on using games as benchmark platforms for AI algorithms. The puzzles such as Skyscraper and Sudoku offer significant computational challenges for general-purpose AI models due to their combinatorial nature, making them a valuable testbed for comparing algorithms and generating novel ideas to be applied to real-world scenarios [12] where authors in [17] developed methodology to use Skyscraper approach, enhancing power production in PV arrays.

This project focuses on the application of GA to the solution of Skyscraper. GA is a type of evolutionary algorithm that emulates the natural selection process, leading to biological evolution [18], [19]. It is a population-based search method that is being used to solve complex problems by generating a population of candidate solutions, called chromosomes (individuals). Various biologically inspired operations such as crossover, mutation, and selection applied to the chromosomes (specifically genes which represent a single component of a solution) which gradually led to the

optimum solution. Each chromosome encodes a solution to a problem and evaluates it through fitness function, and the evolution process starts through major operations of selection, crossover, and mutation [18] these evolutionary processes equip GA with the ability to explore a wide range of potential solutions and adapt to the characteristics of the problem space. Despite these advantages, as the size of the problem increases, it may result in the GA becoming trapped in local optima, posing challenges for parameter tuning [20].

In the basic form of GA, the process starts with generating an initial set of solutions (population) which could be random. Various methods for generating the initial population are explored such as pseudo-random number generators and quasi-random sequences the main point is to create as much diversity as possible [21]. The initial population is encoded to the problem and evaluated through the fitness function which checks the correctness of the solution based on the problem constraints. The evolution process starts based on the score of the fitness function and the GA parameters i.e. mutation rate, crossover rate, and the selection method. Selection consists of probabilistically choosing two chromosomes from the current population based on their fitness score. There are several selection mechanisms, namely, tournament selection, roulette wheel selection, and binary tournament selection [12], [22], among which tournament is the most popular due to its simplicity and efficiency [23]. At each generation, a new population of potential solutions emerge from the existing population. The process of generating a new population is through a crossover process. In this process, characteristics between pair individuals will be exchanged to generate new offspring individuals. Various crossover techniques have been proposed and evaluated in the context of GA. These techniques include single-point, n-point, and mixed-crossover, to enumerate a few. Crossover helps to create more diversity within the population leading to better exploration within the search space [24]. The efficiency of crossover techniques plays a crucial role in locating global optimum by balancing exploration and exploitation and avoiding premature convergence in the GA. The idea behind crossover is to diversity search space which is a vital strategy for achieving an efficient search [25]. Mutation is similar to crossover operation [26] and helps to maintain the genetic diversity in the search space [27], [28] with the difference that the crossover involves combining genetic material from two parent chromosomes. On the other hand, the mutation process involves making small changes in the genome of a single chromosome [9], [12]. Numerous mutation techniques have been explored and proposed, including random mutation (RM), uniform mutation, and non-uniform mutation (NUM), among others [29]. The final stage in GA is to terminate the search once the satisfactory solution quality is found or auto termination once reached the maximum number of the generations which sounds simple; however, this has been a subject of significant research [30], [31]. Authors in [31] proposed a genetic algorithm with automatic accelerated termination (G3AT) and emphasized the importance of developing a GA that can terminate without prior knowledge of any desirable or available solution range. evaluated the performance of the G3AT algorithm in various dimensions and compared it with other existing algorithms, demonstrating the effectiveness of the automatic termination criteria. This emphasises on the importance of automatic termination criteria, the challenges of balancing exploration and exploitation, and the need for robust and efficient termination detection algorithms.

Evolutionary algorithms such as GA are one of the popular approaches for combinatorial optimization problems [32]. There are numerous studies on the application of GA in optimization and complex problem-solving in puzzle games, demonstrating their effectiveness in finding near-optimal solutions for complex problems. They have been utilized to address a variety of problems [10], [20], [26], [33], [34], [35]. Despite GAs being effective in solving NP-complete problems, particularly optimization problems, they do have some limitations. These include slow convergence and difficulty in escaping local minima which does not guarantee the exact solution to the problem [36].

While famous Japanese puzzles such as Sudoku and similar puzzles have been widely investigated and studied for several purposes in AI algorithms. Skyscraper puzzles have not received the same attention and there is space for more exploration of this puzzle type. This gap in the literature presents an opportunity to further investigate the application of GA to skyscrapers and assess their potential as a benchmark for AI research. The complexity and NP-complete nature of the skyscraper puzzles make them an ideal candidate for this research study. The rationale for this research lies in the potential of puzzle games, in particular, Skyscraper, as AI benchmarks and the focus on search and planning in the context of games as a test bed for AI performance evaluation.

This project adheres to the concept approach from the research done by authors in [7] which authors developed a unified framework for the study of combinatorial black-box optimization algorithms that connect several puzzles to various solvers in a way that a puzzle could be solved using one of the existing solvers and vice versa. However, in this project, there is a single type of puzzle with a specific solver. The authors implemented ant colony optimization (ACO), genetic algorithms, simulated annealing, backtracking, and random search solvers to solve the Japanese puzzles which are Sudoku, Futoshiki, Hashiwokakero, Nurikabe, Slitherlink. The research approach is comprehensive, providing a new practical unified platform for rapid development and application of solvers with potential in algorithm benchmarking. The research approach is innovative as it provides an open-source framework that can be extended to further include solvers and puzzles. Additionally, the framework allows for the direct comparison of puzzles and solvers which makes it a platform for both practical applications and theoretical studies.

III. Methodology

The approach combines Genetic algorithms with heuristics

to effectively solve the Skyscraper puzzle. The foundation of the methodology lies in the creation of the puzzle rules which will be referred to as fitness function using Java, facilitating the evaluation of puzzle instances. A software development life cycle will be implemented such as a design that includes solver functionality and performance metrics, development that includes puzzle and Genetic algorithm implementation, and testing which includes functionality and performance testing approach.

A. Skyscraper Puzzle Simulator

Skyscraper puzzle is a combinatorial optimization problem where the goal is to place skyscrapers of varying heights in a grid such that each row and each column contains unique heights, and additional constraints related to visibility from the edges of the grid are satisfied [6], [13].

TABLE I shows the correspondence between terms used in GA and their equivalents in the context of problem-solving, in particular, puzzle games. Each solution is assessed through a cost function called the fitness function. The fitness function is the main part of the puzzle simulation design and plays a crucial role in the performance of the whole software. Algorithms 1 and 2 illustrate the flow and design of the fitness function implemented in this research. Let n denote the size of the grid (i.e., an $n \times n$ grid), and let S represent a candidate solution, where S_{ij} is the height of the skyscraper in the cell at row i and column j. The objective of the fitness function in our design is to maximize the fitness score $F(S)$, where $F(S) \leq 0$ (initiated to 0) and defined as a composite measure of constraint violations, with a fitness score of 0 representing a correct solution and negative values indicating incorrect solutions with the magnitude representing the number of broken constraints. The fitness score decreases by 1 each time a cell in the row or column violates the puzzle's rules. There are two constraints in the context of the Skyscraper puzzle.

1) **Uniqueness computation:**
Each row and each column must contain unique heights (value) from the set { 1, 2, …, n } represented by $V(i,j)$. The algorithm 3 illustrates the procedure for each row, the same process applies to each column.

Algorithm 3 Uniqueness fitness and cell entry validation

```
F = 0
initialize an empty set S
for each row i from 1 to n, let V(i,j) = S_ij
    If V(i,j) is empty || V(i,j) ∈ S || V(i,j) < 1 || V(i,j) > n then
        decrement F by 1
    add V(i,j) to the set S
end
```

2) **Visibility Constraint Evaluation:**
For each row and column, the simulator evaluates the visibility from both ends (left to right, right to left for rows; top to bottom, bottom to top for columns). Algorithm 4 depicts the process of evaluating the clue constraints. The fitness score initiated to zero. Let M denote the maximum seen value in the line (row or column) and S as the count of seen values in the line. P represents the position index within the line.

Algorithm 4 Visibility fitness and Clue-based Validation

```
F = 0
for each row L from 1 to n do
    M = 0, S = 0
    for each position P from 1 to n do
        if from start = true do
            let idx = P
        else
            let idx = n – 1 – P
        end
        let value = S_{L, idx}
        if value > M then
            M = value
            S = S + 1
        end
    end
    if C_left[L] ≠ 0 and S ≠ C_left[L] then
        F = F – 1
    end
end
```

The process of the cell evaluation for uniqueness will be conducted for each row and column and the clue constraint satisfaction for all four edges using algorithms 3 and 4 respelled and sum the fitness score for each solution candidate as follows:

$$F(S) = F_{unique}(S) + F_{visibollity}(S) \quad (1)$$

TABLE I
EQUIVALENTS OF GA TERMS IN PUZZLE GAME CONTEXTS

Evolution	Problem Solving
Chromosome (Individual)	Candidate Solution
Fitness	Quality
Population	Set of Candidate solutions
Gene	Componenet or Attribution od the solution

B. Genetic Algorithm

The GA is sourced as an off-the-shelf solution for the project [37], [38], [39]. To enhance the Genetic algorithm's compatibility with the research objectives, it will subsequently undergo refinement for optimization. This strategy unfolds with the execution of multiple instances of the Skyscraper puzzle which serves as a compelling proof of concept which includes selecting puzzle instances, dividing them into training and test sets, tuning solvers using the training set and evaluating the solver against the test set. It is worth mentioning that in this research the GA is treated as a black-box general-purpose optimization method. Selecting the optimal parameter configuration is a crucial and challenging task in evolutionary algorithms, in this research, a manual approach was employed, incorporating random sampling and comprehensive data collection, followed by an evaluation of the results from the GA. Despite GAs success in optimization and solving complex problems, adapting GAs to new problems is not an easy task, particularly, since the inception of GAs, premature convergence has been acknowledged as one of their recurrent issues [40], therefore when applying GAs to new problems, it is crucial to be mindful of this matter.

Premature convergence occurs when all members of the population are located in a small, suboptimal part of the search space — different from the optimal region — and the chosen components do not facilitate escaping from this region. In this work, the tuning approach is a randomized search that involves randomly selecting hyper-parameter combinations to evaluate a machine learning model. The method is particularly effective in a scenario where there are few hyper-parameters involved [41]. The approach taken in this research is conducting small-scale experiments with specific parameter settings to examine and understand the significant effect of each parameter, as well as the interactions between them. The outcomes of these small-scale experiments are then used to select optimal parameter values, which are subsequently tested through large-scale runs to evaluate the efficiency of the tuning process. The idea behind this approach is to consider the main effects related to parameter and value setting, to determine the most efficient parameter settings for the GA [42]. A classical Genetic Algorithm [19], [43] is used in this research work as shown in algorithm 1.

Algorithm 1 Genetic Algorithm

```
Initialize population with random individuals
While termination condition is not met do
    newPopulation <- empty list
    While size of newPopulation < populationSize do
        parent1 ← selectParent()
        parent2 ← selectParent()
        offspring ← crossover(parent1, parent2)
        Add offspring to newPopulation
    For each individual in newPopulation do
        mutate(individual)
    Population ← newPopulation
    BestInGeneration ← getBestSolution()
    If fitness of bestInGeneration >= fitnessThreshold then
        break
    If generation >= maxGenerations then
        break
    If no improvement in fitness for certain number of generations  then
        break
```

C. Initial Population and Operators in GA

In traditional GA implementations [43], [44], the problem is encoded as binary strings, namely, binary coded genetic algorithm (BCGA). This approach yields satisfactory performance on small- and moderate-size problems requiring less precision in the solution, however, BCGA requires huge computational time and memory [45] for complex problems that call for precise solutions. To address this limitation when implementing BCGA to multidimensional and high-precision numerical problems, decision variables can be encoded as real numbers [46]. In our approach, we utilized this method to encode the cell values into GA as a list of integers i.e. each gene is real value (integer) which represents a cell value within the board. A sequence of genes, which is called a chromosome, forms a candidate solution [47]. The algorithm manipulates iteratively a finite set of chromosomes called population, using the mechanism of evolution. At each generation, chromosomes are subjected to certain operators, such as crossover and mutation, which are similar processes which occur in natural reproduction [43].

In GAs, once the random population of chromosomes is generated, the selection operator is applied immediately, as soon as the reproduction operation ends. There are various approaches suggested for selecting parent strings, which may vary depending on the difficulty level of the problems. This operation has a significant effect on the speed of evolution and the convergence of the GA. In the selection operator, chromosomes are copied into the mating pool implementing various techniques and approaches such as tournament, rank selection, and roulette wheel selection [22]. By transferring highly fit chromosomes to the next generation mating pool, the selection process emulates Darwinian survival-of-the-fittest principles from natural biology, where an individual's fitness is determined by its ability to survive predators, diseases, and other challenges until maturity and subsequent selection. In this stage, no new chromosome is created [48]. In this work, we considered the tournament selection process for its simplicity, efficiency, and flexibility. Tournament selection is straightforward to implement, involving the random selection of a few individuals determined by the tournament size, comparing their fitness, and choosing the best one. Compared to the proportional selection approaches such as roulette wheel selection, which requires scanning the entire population to determine the cumulative fitness and select individuals, a tournament is more efficient, specifically when the search space increases for the larger instance of the game. This efficiency, combined with the control over selecting the tournament size, enhances its robustness and effectiveness [22], [49]. The tournament selection implemented in this research is as follows:

Algorithm 2 Tournament Selection

```
selectParent():
    Best ← null
    for i=1 to k do
        Competitor ← p[random individual i as 0  <= i <=  k];
        if Best = null || Fitness (Competitor) > Fitness(Best)
        then
            Best ← Competitor;
    return Best;
```

The next phase of the evolution process is the crossover operation. In this research, a single-point crossover operation at a random point on two-parent puzzles is implemented to generate two child puzzles. The function takes the crossover rate which is one of the crucial GA parameters (later in the result the effect of the crossover rate will be discussed). The crossover rate in a GA determines how often crossover will occur between two parent individuals [25]. It is a probability value between 0 and 1. In our work, the crossover rate is checked against a randomly generated number between 0 and 1. If the random number is greater than the crossover rate, then crossover does not occur. Instead, two new individuals are created without any inheritance from the parent individuals. In the case of performing crossover, the point at which the genes are swapped is determined by a randomly chosen crossover point. the crossover rate controls the balance between

exploration (creating new, random individuals) and exploitation (building on the best-found solutions) in the genetic algorithm. A high crossover rate means that crossover will occur more often, leading to more exploitation, while a low crossover rate means that new, random individuals will be created more often, leading to more exploration [50]. A pseudocode of the crossover implemented in this work is given in algorithm 5. Let n denote the size of the grid and let $V1$ and $V2$ represent the two parent puzzles. The crossover function generates two new puzzles $C1$ and $C2$ based on a crossover point cp and a crossover rate r. Assuming we set the crossover rate to 0.9 which means that there is a 90% chance for crossover to occur between two parent puzzles. In the case of selecting the puzzle instance of size 5, then there are 25 cells in each puzzle. The crossover happens at a single point along the length of the puzzle. The crossover point is set to be chosen randomly which will be between 0 and the game size (exclusive). Assuming the crossover point is at row 2 this means that rows 0 and 1 (10 cells in total) would be taken from one parent for each offspring and the other remaining 15 cells in total would be taken from the other parent.

Algorithm 5 Single-point Crossover

```
if r is greater than the crossover rate cr then
    return C1=V1 and C2=V2;
else
    initialize two new puzzles C1 and C2 with size n × n;
    for each row i from 0 to n – 1 do
        for each column j from 0 to n – 1 do
            if i < cp do
                C1_ij = V1_ij
                C2_ij = V2_ij
            else do
                C1_ij = V2_ij
                C2_ij = V1_ij
            end
        end
    end
end
return C1 and C2;
```

Similar to crossover, mutation is used to maintain genetic diversity from one generation of a population of candidate solutions to the next [25]. It is ensuring that the population does not become similar over time which could lead to local minima and yield premature solutions. In our approach, each candidate solution is encoded as a list of real numbers. The mutation involves adding a small random value from set { 1, 2, …, n } in random places within the chromosome. Let n denote the size of the grid and let I represent the puzzle (individual) to be mutated. The mutation function alters the values of I based on a mutation rate m and a random value rv.

Algorithm 6 Random Mutation

```
for each row i from 0 to n – 1 do
    for each column j from 0 to n – 1 do
        Generate a random number r r uniformly distributed between 0 and 1;
        if r < m do
            I_ij = rv
        end
    end
end
```

To illustrate how the mutation method works, we picked a mutation rate of 0.09. This means there is a 9% chance for each cell (or gene) in the puzzle to be mutated. However, the exact number of genes that will be mutated and their positions are determined randomly and cannot be predicted exactly at which position and how many genes would be subjected to the mutation process. If we assume the game size is 5, therefore are 25 cells in the puzzle. On average, 9% of these cells will be mutated, which is approximately 2 cells (0.09 * 25 = 2.25, since the number of cells must be an integer, the number is rounded down). The positions of these 2 cells are determined randomly. Each cell has an equal chance to be chosen for mutation. For example, the cell at position (1, 1) has the same chance of 9% as the cell at position (4,3) to be mutated, and so on. This is to just explain the mutation process with actual probability; However, in practice, the exact number of mutated cells and their positions in each generation will vary due to the randomness introduced in the mutation function.

The final phase of the research involves cross-comparison of the Genetic algorithm's efficiency in solving the Skyscraper puzzle with other algorithms and methods in terms of both solution quality and time taken to find the solution. The reason that we did not choose high-rate mutation is to avoid applying extra randomness to the algorithm so that it could turn into a random search instead of an evolution process.

IV. Experimental Results

The difficulty of the Skyscraper puzzle is defined by the number of given edge clues. However, these are out of the methodology scope. In this approach, it had been tried to implement the standard skyscraper as a proof of concept. As mentioned in the related work section there is a literature gap in applying GA on Skayscarper in which there is no proper research result that could be used to compare to this research outcome except the work in [39] which is a GitHub repository implementing Backtracker on Skyscraper. Investigating this work shows that the work lacks proper reporting of how accurate is the final. Running multiple instances of the puzzle used in this work showed that the model does not return the correct solution. In contrast to this research, in the case of reaching the final generation and the GA did not find the absolute solution, it reports if the absolute solution found or not and the fitness score shows how inaccurate is the final solution. This provides insight details on the results as well as generation progress that includes fitness score and time taken per generation for further investigation. A brief comparison is made in the following section between the GA proposed in this work and the Backtracker in [39] using time and the

solution quality (success rate).

A. Fitness function validation

To ensure the functionality of the fitness function, which is the crucial part of the puzzle simulator it tested against several instances of standard puzzles with known solutions. The simulator has two entry modes, manual and automatic. The manual mode is for the testing purpose which loads an instance of the game along with its solution. The solution will be fed to the fitness function and validated against the constraints. In the case of absolute correctness, the fitness score will return zero and, in the case, of testing, some of the rules broken within the solution are known as the number of broken rules, the fitness score is supposed to decrease one point for each broken rule.

B. Puzzle Instance Data

The instance of puzzles used in this research is from [51] which hosts a variety of puzzles complete with solutions. The website offers a range of puzzles, including daily updates on the Skyscraper instances. The puzzles chosen for this study tried to have unique solutions, however, some instances could have multiple solutions, specifically in the larger scale of the puzzle. The selected instances for each puzzle were further divided (randomly) into 2 training instances and 5 test instances. The training instances were utilized for tuning algorithm parameters, while the test instances were employed for the experimental evaluation of the game simulator.

C. Parameter Tuning

Initially, a population size of 100, a crossover rate of 0.1, and a mutation rate of 0.01 were selected for preliminary runs were conducted to establish a performance baseline. The tuning started with the smallest instance of puzzle size (4x4), increasing population size while the other parameters were fixed. This continued till reached the point that increasing the population size did not have an improvement on the fitness score. Then the generation size increased in the same pattern. The mutation rate increased with cuties concerning the population size. The crossover rate is set to 0.9 to have the possibility of almost all solution candidates exchanging genes. Table II shows the GA parameters used for the experiment.

Table II
This is an experiment parameter for GA

Population Size	380
Generations	4500
Tournament Size	10
Crossover Rate	0.9
Mutation Rate	0.09

D. Experimental Environment

All the codes were compiled and run on the same machine with OpenJDK version "17.0.9" 2023-10-17, OpenJDK Runtime Environment Homebrew (build 17.0.9+0), OpenJDK 64-Bit Server VM Homebrew (build 17.0.9+0, mixed mode, sharing). The machine used in this experiment uses macOS Mentory (version 12.6.8) with processor Quad-Core Intel Core i7 with a clock speed of 2.7 GHz. In this experiment, a multi-threading approach was implemented. It was observed that executing GA in a separate thread from other concurrent processes, such as time tracking and data visualization, significantly enhanced the algorithm's performance.

E. Performance Evaluation of Genetic Algorithm

In contrast to algorithms such as backtracker, GA is more efficient for larger problem sizes, however, does not guarantee an exact solution. Instead, it yields an optimal solution. The performance of the GA is influenced by various factors, such as population size, mutation and crossover rate as well as other parameters that are introduced to GA based on the approach taken for each evolutionary part. The GA used in this work was evaluated using several test instances of the puzzle with different sizes. During the tuning parameters and evaluation, it was observed that increasing the mutation rate above 0.09 significantly degrades the solution quality, regardless of the population size. On the other hand, reducing the crossover rate below 0.9 reduces the genome diversity and GA trap into the local minimum. Fig. 2 and Fig. 3 illustrate the fitness score versus generation for various Skyscraper size that shows the fitness improvement over generation. Fig. 2 left shows that although GA fell into local minimum for generations, however, it managed to find a near-optimal solution. This always will not be the case, in which Fig. 3 shows that for the first 1000 generations, GA had optimization of the fitness score, however for the rest of the process it could not reach the optimal solution and shows how Ga struggling to find the absolute solution. This is due to the increase in the search space which is due to increment in the puzzle size. One approach to handle this situation can be increment in the population size. Although with the current setting for the GA parameters, it still can find a solution for the 6x6 puzzle. The success rate for these instances is relatively lower than the 4x4 and 5x5 as illustrated in Table III. To analyse the performance of GA for solving Skyscraper puzzle games, several factors were taken into consideration, the average time to find a solution, the success rate, and the convergence rate. The average time to find a solution is considered by averaging the elapsed time for all successful runs. In this experiment, it focused on the total time taken when the solution is found.

$$Average\ Time = \frac{1}{N}\sum_{i=1}^{N} T_i \qquad (2)$$

Where T_i is the time taken in the *i*-th successful run, and *N* is the number of successful runs.

The success rate is the ratio of successful runs where the solution is found (a run considered successful if the GA find a solution candidate with a fitness score of zero) to the total number of runs.

$$Success\ Rate = \frac{Number\ of\ \ Successf\,ul\ Runs}{Total\ Runs} \quad (3)$$

The convergence rate is the average number of generations required to find the solution's overall successful runs.

$$Convergence\ Rate = \frac{1}{N}\sum_{i=1}^{N} G_i \quad (4)$$

Where G_i is the number of generations needed to find the solutions to the *i*-th successful run?

F. Comparison with Backtracker Solver

Based on the literature review up to the date of this research, there is no existing literature—either academic or otherwise—that presented experiment results on applying GA to the Skyscraper puzzle. Although there are GitHub repositories, they require extra effort to compile, run, and extract results and they do not follow the same experimental setups. The only related work with a similar experimental setup is [39] which implemented a backtracker algorithm on Skyscraper. However, this work has several limitations: the output results are not properly documented and running the provided code using the instances used in this work revealed problems with the puzzle simulator, indicating either the reporting issue or the fitness function is not correctly implemented. It seems that the codes yield valid results only with the instances provided within the repository, some of which have pre-filled cells. It indicates that the backtracker is not able to solve the puzzle without assistance. Conducting experiments using some instances used in this research showed that the BT model is not able to produce absolut correct results and due to the lack of reporting, there is no indication of the quality of the output. Since the backtracker seeking to find a solution for the puzzle and the optimization does not apply to this algorithm [7], considering the time taken to come up with the final answer from this approach does not make much sense to compare with GA. After thoroughly investigating the work, it was decided not to conduct a direct comparison and instead to provide examples from the work that show the model output.

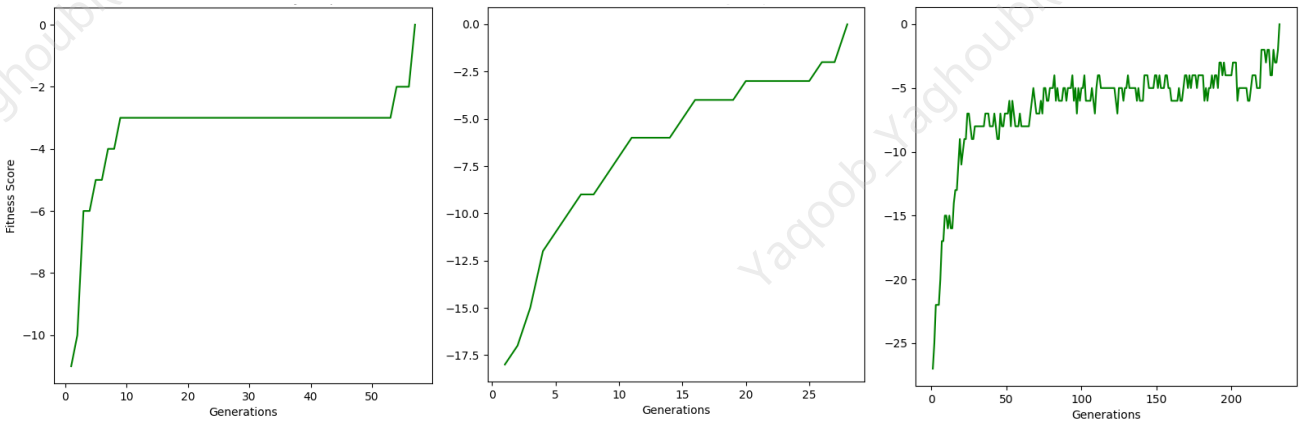

Fig. 2. Fitness score over generations for 4x4 (left), 5x5 (middle), 6x6 (right)

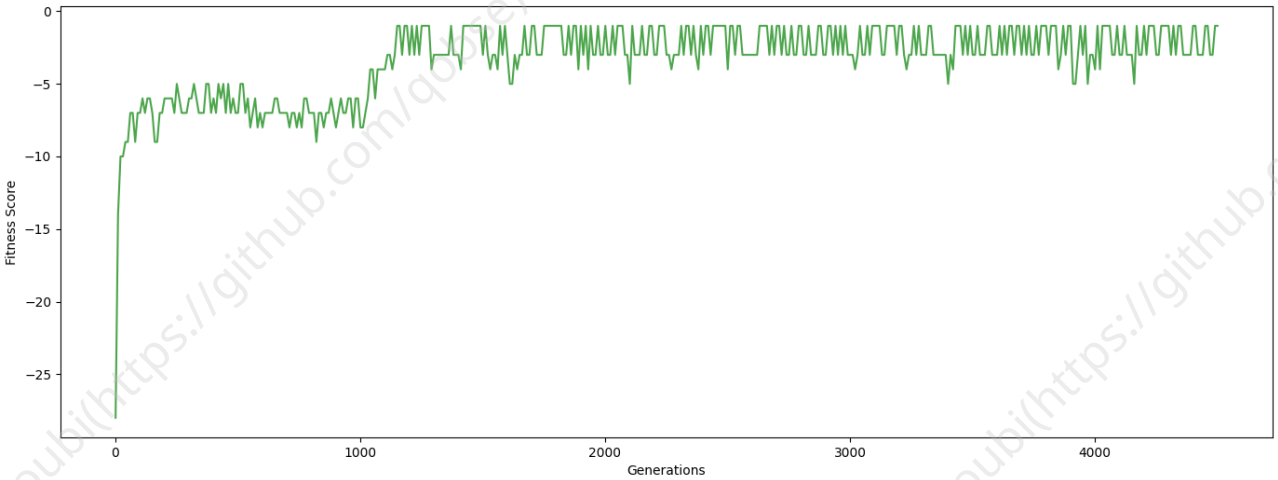

Fig. 3. Fitness score over generations for 6x6 instance

Table III
PERFORMANCE OF GA ON VARIOUS SIZE OF THE PUZZLE INSTANCE

instance	Success %	Mean Solution Time ms	STD DEV Time ms	Mean Convergence Generation	Mean Unseccessful Fitness Score	STD DEV Unseccessful Fitness Score
4x4_1	84	145.69	125.24	13.6	-2	0
4x4_2	98	725.02	2416.21	119.61	-2	0
4x4_3	90	735.8	2307.62	106.2	-2	0
4x4_4	90	142.2	113.96	11.13	-4	0
4x4_5	82	122.44	55.39	9.9	-2	0
5x5_1	62	4279.81	8699.85	515.1	-2.84	1.66
5x5_2	34	2697	3469.68	322.94	-3.39	1.1
5x5_3	20	7413	7510.21	937.4	-1.7	0.81
5x5_4	56	5912.89	9455.32	731.14	-2.36	0.57
5x5_5	36	5480.5	7716.1	657.11	-3.84	0.75
6x6_1	28	18568.29	14445.05	1598.29	-2.92	1.38
6x6_2	8	111651.25	156011.27	2605	-3.35	1.29
6x6_3	42	22931.86	38221.57	1124.14	-2.79	1.27
6x6_4	26	22594.15	13262.29	1877.69	-4.03	1.35
6x6_5	14	7516.57	3197.73	559.86	-3.26	1.48
7x7_1	0	0	0	0	-11.56	1.75
7x7_2	0	0	0	0	-11.72	1.89
7x7_3	0	0	0	0	-11.66	1.7
7x7_4	0	0	0	0	-14.66	1.9
7x7_5	0	0	0	0	-12.56	1.76
8x8_1	0	0	0	0	-25.78	1.81
9x9_1	0	0	0	0	-40.4	2.2

The data presented in the table are for the success runs in 50 runs. Success % is the number of successful solutions found in 50 runs. Times given as mean and standard deviation for the successful runs in mili second.

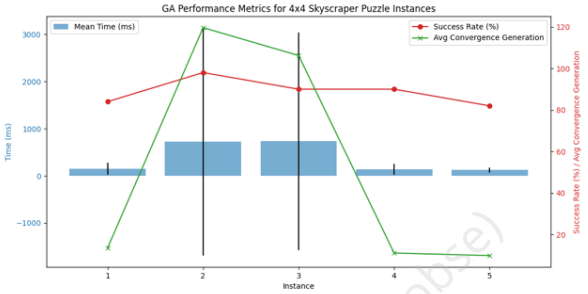

Fig. 4. GA performance metrics for 4x4 instances

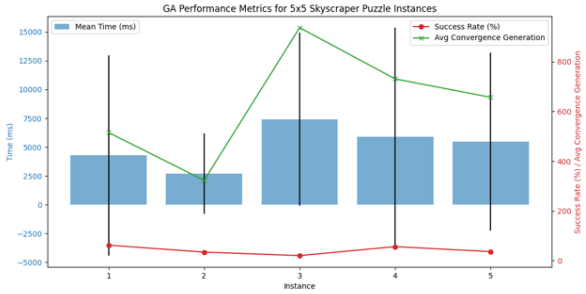

Fig. 5. GA performance metrics for 5x5 instances

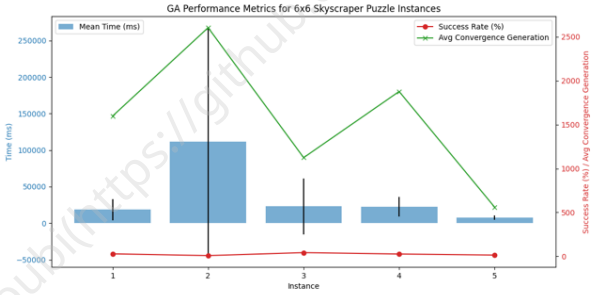

Fig. 6. GA performance metrics for 6x6 instances

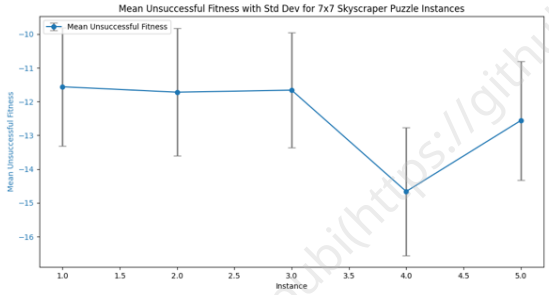

Fig. 7. GA performance metrics for 7x7 instances

V. Discussion

The experiment results illustrate the performance of GA on a range of puzzle games implemented in the skyscraper simulator. The results obtained from the experiments provided a comprehensive understanding of the GA performance when applied to the range of Skyscraper instances with a variety of sizes. The Fig. 4 to 7 illustrate the overall performance of GA. The GA demonstrated its ability to efficiently handle the small-size puzzles. However, as the puzzle size increased to 6x6 and behind that, the GA faced significant challenges. The success rate for the 6x6 instances dropped approximately by 60% compared to the 4x4 instance and the average time increased by about 3%. It can be seen that with the increase in the Skyscraper size, the computation cost significantly increases while the success rate reduced. This suggests that the Skyscraper is challenging to be completely understood by Ga and solved. Increasing the Skyscraper size above 6x6 with the GA parameters suggested in Table II yields a success rate of zero. It can be understood that these are behind the capability of GA to handle it. However, increasing the population and further tuning the GA parameter suggested that this range of problems could be solved. However, the computation cost significantly will be increased.

VI. Conclusion

This research presented the exploration of GA as a method for solving the Skyscraper puzzle. The comprehensive experimental details on the performance of Ga on skyscrapers are presented. The results illustrate that the GA will perform on

A smaller scale is the solution, while has significant challenges on a larger scale. This proves that this type of puzzle could be a great example in benchmarking optimization algorithms. To build on these findings, there are many aspects of the GA on Skyscraper that have not been investigated in this research, such as efficient approach in parameter optimization, and implementing various evolutionary approaches such as selection, crossover and mutation. From the experimental approach, implementing a wider range of puzzles with larger sizes with automated optimization and tuning. Considering the literature gap in investigating skyscrapers, there could be more optimization algorithms applied to this type of problem.

References

[1] Alex Bellos, *Puzzle ninja*. San Francisco: Chronicle Books, 2017.

[2] S. Salcedo-Sanz, E. G. Ortíz-García, A. M. Pérez-Bellido, A. Portilla-Figueras, and X. Yao, 'Solving Japanese Puzzles with Heuristics'.

[3] Marcel Danesi, *The puzzle instinct : the meaning of puzzles in human life*. Bloomington, Ind.: Indiana University Press; Chesham, 2004.

[4] G. N. Yannakakis and J. Togelius, 'A Panorama of Artificial and Computational Intelligence in Games', *IEEE Trans Comput Intell AI Games*, vol. 7, no. 4, pp. 317–335, Dec. 2015, doi: 10.1109/TCIAIG.2014.2339221.

[5] K. Haraguchi and R. Tanaka, 'The building puzzle is still hard even in the single lined version', *Journal of Information Processing*, vol. 25, pp. 730–734, Aug. 2017, doi: 10.2197/ipsjjip.25.730.

[6] L. Kolijn, 'Generating and Solving Skyscrapers Puzzles Using a SAT Solver', 2022.

[7] H. Lloyd, M. Crossley, M. Sinclair, and M. Amos, 'J-POP: Japanese Puzzles as Optimization Problems', *IEEE Trans Games*, vol. 14, no. 3, pp. 391–402, Sep. 2022, doi: 10.1109/TG.2021.3081817.

[8] P. R. Wurman, P. Stone, and M. Spranger, 'Improving artificial intelligence with games', American Association for the Advancement of Science, Jul. 2023. doi: 10.1126/science.adh8135.

[9] 'Solving_rating_and_generating_Sudoku_puzzles_with_GA'.

[10] X. Q. Deng and Y. Da Li, 'A novel hybrid genetic algorithm for solving Sudoku puzzles', *Optim Lett*, vol. 7, no. 2, pp. 241–257, Feb. 2013, doi: 10.1007/s11590-011-0413-0.

[11] H. Lloyd and M. Amos, 'Solving Sudoku with Ant Colony Optimization', *IEEE Trans Games*, vol. 12, no. 3, pp. 302–311, Sep. 2020, doi: 10.1109/TG.2019.2942773.

[12] F. Gerges, G. Zouein, and D. Azar, 'Genetic algorithms with local optima handling to solve sudoku puzzles', in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Mar. 2018, pp. 19–22. doi: 10.1145/3194452.3194463.

[13] T. Khovanova and J. B. Lewis, 'Skyscraper Numbers', Apr. 2013, [Online]. Available: http://arxiv.org/abs/1304.6445

[14] K. Haraguchi and H. Ono, 'Approximability of Latin Square Completion-Type Puzzles'. [Online]. Available: http://www.nikoli.co.jp/en/

[15] Ed. by Charles J. Colbourn and Jeffrey H. Dinitz., *Handbook of Combinatorial Designs, 2d Ed.*, 2d ed., vol. 31. 2007.

[16] GeorgiosN. Yannakakis and J. Togelius, *Artificial Intelligence and Games*. Springer, 2018.

[17] M. S. S. Nihanth, J. P. Ram, D. S. Pillai, A. M. Y. M. Ghias, A. Garg, and N. Rajasekar, 'Enhanced power production in PV arrays using a new skyscraper puzzle based one-time reconfiguration procedure under partial shade conditions (PSCs)', *Solar Energy*, vol. 194, pp. 209–224, Dec. 2019, doi: 10.1016/j.solener.2019.10.020.

[18] J. H. Holland, 'Genetic Algorithms', vol. 267, no. 1, pp. 66–73, 1992, doi: 10.2307/24939139.

[19] J. H. Holland, *Adaptation in Natural and Artificial Systems*. The MIT Press, 1992. doi: 10.7551/mitpress/1090.001.0001.

[20] J. M. Weiss, 'Genetic Algorithms and Sudoku'.

[21] H. Maaranen, K. Miettinen, and A. Penttinen, 'On initial populations of a genetic algorithm for continuous optimization problems', *Journal of Global Optimization*, vol. 37, no. 3, pp. 405–436, Mar. 2007, doi: 10.1007/s10898-006-9056-6.

[22] G. Singh *et al.*, *Comparative Review of Selection Techniques in Genetic Algorithm*.

[23] A. Lara-Caballero and D. González-Moreno, 'A Population-Based Local Search Algorithm for the Identifying Code Problem', *Mathematics*, vol. 11, no. 20, Oct. 2023, doi: 10.3390/math11204361.

[24] O. ! Guzhan, H. Ëebi, and F. Erbatur, 'Evaluation of crossover techniques in genetic algorithm based optimum structural design'. [Online]. Available: www.elsevier.com/locate/compstruc

[25] S. M. Lim, A. B. M. Sultan, M. N. Sulaiman, A. Mustapha, and K. Y. Leong, 'Crossover and mutation operators of genetic algorithms', *Int J Mach Learn Comput*, vol. 7, no. 1, pp. 9–12, Feb. 2017, doi: 10.18178/ijmlc.2017.7.1.611.

[26] H. Chel, D. Mylavarapu, and D. Sharma, 'A Novel Multistage Genetic Algorithm Approach for Solving Sudoku Puzzle'.

[27] D. Bhandari, N. R. Pal, and S. K. Pal, 'Directed mutation in genetic algorithms', *Inf Sci (N Y)*, vol. 79, no. 3–4, pp. 251–270, Jul. 1994, doi: 10.1016/0020-0255(94)90123-6.

[28] I. Korejo, S. Yang, and C. Li, 'A Directed Mutation Operator for Real Coded Genetic Algorithms'.

[29] K. Deep and M. Thakur, 'A new crossover operator for real coded genetic algorithms', *Appl Math Comput*, vol. 188, no. 1, pp. 895–911, May 2007, doi: 10.1016/j.amc.2006.10.047.

[30] D. K. Saxena, A. Sinha, J. A. Duro, and Q. Zhang, 'Entropy-Based Termination Criterion for Multiobjective Evolutionary Algorithms', *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 4, pp. 485–498, Aug. 2016, doi: 10.1109/TEVC.2015.2480780.

[31] B. T. Ong and M. Fukushima, 'Genetic algorithm with automatic termination and search space rotation', *Memet Comput*, vol. 3, no. 2, pp. 111–127, Jul. 2011, doi: 10.1007/s12293-011-0057-8.

[32] A. E. Eiben and J. E. Smith, 'Natural Computing Series Introduction to Evolutionary Computing'. [Online]. Available: www.springer.com/series/

[33] J. Wang and L. Huang, 'Evolving Gomoku solver by genetic algorithm', in *Proceedings - 2014 IEEE Workshop on Advanced Research and Technology in Industry Applications, WARTIA 2014*, Institute of Electrical and Electronics Engineers Inc., Dec. 2014, pp. 1064–1067. doi: 10.1109/WARTIA.2014.6976460.

[34] D. Sholomon, E. David, and N. S. Netanyahu, 'A Genetic Algorithm-Based Solver for Very Large Jigsaw Puzzles', Nov. 2017, doi: 10.1109/CVPR.2013.231.

[35] Y.-C. Xu, R.-B. Xiao, and M. Amos, 'A Novel Genetic Algorithm for the Layout Optimization Problem'.

[36] S. Katoch, S. S. Chauhan, and V. Kumar, 'A review on genetic algorithm: past, present, and future', *Multimed Tools Appl*, vol. 80, no. 5, pp. 8091–8126, Feb. 2021, doi: 10.1007/s11042-020-10139-6.

[37] V. Mallawaarachchi, 'Introduction to Genetic Algorithms', Towards Data Science. Accessed: May 23, 2024. [Online]. Available: https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3

[38] memento, 'GeneticAlgorithm'. Accessed: May 23, 2024. [Online]. Available: https://github.com/memento/GeneticAlgorithm

[39] H. Lynd, 'skyscraper-puzzle-solver', https://github.com/Helena-Lynd/skyscraper-puzzle-solver. Accessed: May 23, 2024. [Online]. Available: https://github.com/Helena-Lynd/skyscraper-puzzle-solver

[40] M. Crepinsek, S. H. Liu, and M. Mernik, 'Exploration and exploitation in evolutionary algorithms: A survey', *ACM Computing Surveys*, vol. 45, no. 3. Jun. 2013. doi: 10.1145/2480741.2480752.

[41] J. Bergstra, J. B. Ca, and Y. B. Ca, 'Random Search for Hyper-Parameter Optimization Yoshua Bengio', 2012. [Online]. Available: http://scikit-learn.sourceforge.net.

[42] M. Mosayebi and M. Sodhi, 'Tuning genetic algorithm parameters using design of experiments', in *GECCO 2020 Companion - Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, Association for Computing Machinery, Inc, Jul. 2020, pp. 1937–1944. doi: 10.1145/3377929.3398136.

[43] D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, 1989.

[44] D. A. Coley, *An Introduction to Genetic Algorithms for Scientists and Engineers*. WORLD SCIENTIFIC, 1999. doi: 10.1142/3904.

[45] P. H. Tang and M. H. Tseng, 'Adaptive directed mutation for real-coded genetic algorithms', *Applied Soft Computing Journal*, vol. 13, no. 1, pp. 600–614, Jan. 2013, doi: 10.1016/j.asoc.2012.08.035.

[46] S. H. Ling and F. H. F. Leung, 'An improved genetic algorithm with average-bound crossover and wavelet mutation operations', *Soft comput*, vol. 11, no. 1, pp. 7–31, Jan. 2007, doi: 10.1007/s00500-006-0049-7.

[47] R. Potolea, R. R. Slavescu, IEEE Romania Section, and Institute of Electrical and Electronics Engineers, *Tuning model parameters through a Genetic Algorithm approach*.

[48] Z. Hussain Ahmed, 'Adaptive Sequential Constructive Crossover Operator in a Genetic Algorithm for Solving the Traveling Salesman Problem', 2020. [Online]. Available: www.ijacsa.thesai.org

[49] S. I. Ao and International Association of Engineers., *Genetic Algorithm Performance with Different Selection Strategies in Solving TSP*. Newswood Ltd., 2011.

[50] F. Herrera, M. Lozano, and A. M. Sánchez, 'A taxonomy for the crossover operator for real-coded genetic algorithms: An experimental study', *International Journal of Intelligent Systems*, vol. 18, no. 3, pp. 309–338, Mar. 2003, doi: 10.1002/int.10091.

[51] K. Stone, 'Daily Skyscrapers'. Accessed: May 28, 2024. [Online]. Available: https://www.brainbashers.com/skyscrapers.asp