A Multitask Evolutionary Framework for Procedural Content Generation in Games

University of Turku
Department of Computing
Master of Science in Technology Thesis
Software Engineering
May 2026
Hoang Long Nguyen

Supervisors:
Taneli Nyyssönen
Tuomas Mäkilä

UNIVERSITY OF TURKU
Department of Computing

HOANG LONG NGUYEN: A Multitask Evolutionary Framework for Procedural Content Generation in Games

Master of Science in Technology Thesis, 76 p., 16 app. p.
Software Engineering
May 2026

Procedural Content Generation (PCG) has become a widely used technique for creating content in various types of games. In recent years, many approaches have been proposed to tackle different PCG problems. Despite the diversity of these methods, most existing generators are designed for specific games, even though many games share structural similarities. In this study, we explore the use of Evolutionary Multitask Optimization (EMTO) for PCG. We introduce a framework based on a Multifactorial Evolutionary Algorithm (MFEA) that systematically groups PCG problems by the compatibility of their search spaces, allowing for the simultaneous optimization of multiple problems. To assess the effectiveness of the proposed framework, we conducted experiments on a range of PCG benchmark problems and compared the results to a single-task evolutionary approach. The findings highlight both the potential benefits and the challenges of applying EMTO to PCG, particularly regarding task compatibility. [1]

Keywords: procedural content generation, computer games, multifactorial evolutionary algorithm, evolutionary algorithm

---

[1] AI has been used in the writing of this thesis for grammar correction and paraphrasing, as documented in Appendix B.

Contents

List of Figures

List of Tables

1 Introduction

Game content plays a crucial role in maintaining player engagement, while modern games increasingly demand large, diverse, and customizable content. For example, *Minecraft* [1] keeps player interest through procedurally generated worlds and extensive modding support, allowing players to create personalized experiences. Similarly, *No Man's Sky* [2] features a massive game world containing over 18 quintillion planets, which is impossible to create manually. At the same time, manual content creation is often expensive, time-consuming, and difficult to scale, as it typically involves many people with various roles working together for a long period. As a result, content production has become a major bottleneck in game development [3].

To address these challenges, Procedural Content Generation (PCG) has been introduced as a technique for generating game content using computer algorithms [4] rather than manual approaches. Since its initial use in the 1980s, it has been employed to create a variety of content, including textures, music, vegetation, buildings, entity behavior, world elements, levels, narrative, and stories [3]. Accordingly, PCG offers the potential to support scalable content production, enhance games' replayability, and facilitate game development in various aspects, especially saving both cost and time.

One common challenge when generating game content via PCG is assessing the validity and suitability of generated instances for the game [3], thereby evaluating the generator's efficiency. This process is frequently carried out through human

playtesting. While playtesting is one of the most reliable methods to gauge player experience, it can be expensive, time-consuming, and inherently subjective. One attempt to standardize and automate this assessment was made in 2025 by Khalifa *et al.*, with the proposal of the PCG Benchmark [5]. The PCG Benchmark is a framework containing an extendable set of PCG problems, including the generation of game rules, levels, buildings, word-based puzzles, and patterns. Each problem uses the same evaluation scheme: it takes an array of content as input and outputs scores based on three different criteria, which are quality, diversity, and controllability. As a result, the benchmark provides a controlled and reproducible environment for effectively comparing generative methods.

One of the most popular algorithms that has been widely studied in the PCG research community are Evolutionary Algorithms (EAs). Thanks to their flexibility and ease of use, EAs have been applied to a wide range of PCG problems, such as dungeon and platformer level design, puzzle generation, and building layout generation [3], [6]. However, the weaknesses of conventional EAs are their large resource consumption and their typical design to solve one individual problem at a time [7]. In practice, many tasks, particularly those within the same game or domain, often share underlying structural patterns. Leveraging these shared characteristics could enable more efficient content generation. Meanwhile, a long-term goal of PCG research is the development of general content generators, which are systems capable of creating content for different games without being specifically tailored for each one [3]. Achieving this vision requires methods that can learn from multiple generative tasks simultaneously and generalize beyond a single game.

Over the past few years, Evolutionary Multitask Optimization (EMTO) has emerged as a novel research direction within evolutionary computation. Building upon the principles of EAs, EMTO aims to solve multiple optimization problems simultaneously within a unified framework. By optimizing related tasks jointly, the

approach enables knowledge transfer across tasks. Such transfer can enhance solution quality, accelerate convergence, and improve computational efficiency compared to optimizing each task independently.

Multifactorial Evolutionary Algorithm (MFEA) [7], which emerged as one of the most influential method among various EMTO approaches, can solve multiple problems simultaneously within a single evolutionary process. One key innovation of MFEA lies in the concept of encoding all solutions within a unique representation of population called Unified Search Space (USS). By placing individuals from different tasks into a common representation, MFEA allows the transfer of potentially useful genetic materials between related problems, thereby improving search efficiency and facilitating the achievement of optimal solutions for each task. This capability is particularly relevant for PCG, where multiple related content generation tasks often coexist, such as different modes or parameter settings within the same game, as well as structurally similar tasks across different games. Despite such advantages of this algorithm, directly applying MFEA to PCG remains challenging due to the heterogeneous and structured nature of PCG search spaces.

1.1 Goal and Research questions

This thesis aims to explore the use of EMTO to tackle the PCG Benchmark, with MFEA adopted as a representative multitask evolutionary algorithm. Another goal of this thesis is to develop a framework for systematically grouping problems based on the compatibility of their search spaces, which facilitates the application of MFEA to solve multiple PCG problems concurrently. Finally, the effectiveness of proposed framework will be evaluated through experiments on the PCG benchmark and comparisons with a single-task evolutionary approach.

This thesis focuses on addressing the following research questions.

RS1: Representation

How can different game content spaces be unified into a common search representation that allows knowledge transfer between different PCG tasks?

RS2: Knowledge transfer

Can MFEA improve current PCG generators by allowing knowledge transfer between related tasks?

- Does the transfer help when tasks are similar?
- What happens when tasks are different and how to reduce negative transfer?

RS3: Evaluation

How to evaluate MFEA's benefits and drawbacks in PCG?

- Compare single task vs. multitask evolution in terms of convergence, quality fitness, controllability fitness and population diversity.

1.2 Research Methods

This thesis designs and experiments with an algorithm to investigate the application of EMTO to PCG.

First, a multitask evolutionary framework based on MFEA is designed to enable the simultaneous optimization of multiple PCG tasks. This includes proposing an USS construction and grouping tasks based on representational compatibility. Following the design phase, the proposed framework is implemented and applied to the PCG benchmark 2025. The proposed approach is then evaluated through computational experiments, followed by a comparison against a single-task evolutionary algorithm baseline under the same total evaluation budget. The evaluation focuses on multiple performance criteria, including solution quality, controllability, and diversity. To ensure statistical validity, each experiment is simulated multiple times, and the results are analyzed using non-parametric statistical tests, including

the Wilcoxon rank-sum test and the Vargha–Delaney A12A (See Section 5.1.1 and 5.1.2) effect size measure.

Through this research method, the study aims to assess both the effectiveness and limitations of applying evolutionary multitasking to PCG.

1.3 Main Tasks

The main tasks of this thesis can be summarized as follows:

- Design a multitask evolutionary framework for PCG.
- Implement MFEA as a representative multitask optimizer to demonstrate the effectiveness of the framework.
- Analyze and evaluate experimental results on the latest PCG Benchmark with multiple multitask scenarios.
- Compare with the previous EA single-task approach to examine the efficiency of the used algorithm.

1.4 Outline

This thesis is organized into multiple chapters. Chapter 2 covers the theoretical background relevant to this work, including a preliminary understanding of MFEA and a literature review on PCG. In Chapter 3, we provide a summary of the PCG Benchmark. Chapter 4 elaborates on the proposed framework. Chapter 5 includes the setups of our experiment and its computational results, including a performance comparison with another EA single-task algorithm. Finally, conclusions and future extensions are discussed in Chapter 6.

2 Theoretical Background

This chapter provides an overview of all the relevant fields related to this work. We begin by exploring various common approaches to addressing PCG problems, followed by a detailed explanation of MFEA. Finally, we provide a brief literature review on PCG and discuss the lack of research on MFEA within this domain.

2.1 Search-based Procedural Content Generation

PCG is a research domain that has been studied for several decades, with numerous methods proposed for generating diverse types of game content [3]. Among these, Search-based Procedural Content Generation (SBPCG) has become a prevalent approach to many PCG problems. First introduced by Togelius *et al.* [8], SBPCG formulates content creation as an optimization problem defined by a specific goal or a set of goals, along with a fitness function. This method, hence, enables search-based methods to explore complex spaces of game content. SBPCG includes various common forms of heuristics, stochastic, and typically metaheuristic methods, such as EAs. According to the authors in [8], two qualifications of SBPCG are:

- After a candidate content instance is generated, it undergoes testing based on certain criteria (for example, is there a path that connects the entrance to the exit of the dungeon?). These criteria are often represented by one or more test functions. Instead of merely accepting or rejecting the candidate's content,

the test function assigns a grade using one or more real numbers or a vector of real numbers.

- New candidate content is generated based on the fitness evaluations of prior instances, with the goal of producing higher quality outcomes.

2.1.1 Metaheuristic Algorithms

Metaheuristics are a method that employs random factors to find the globally optimal solution for complex optimization problems, especially within large or irregular search spaces. Every metaheuristic algorithm consists of two major components, i.e., exploration and exploitation [9]. Exploration involves discovering a variety of solutions within a search space, while exploitation focuses on improving the quality of an elite solution by searching locally around it. Generally, the trade-off between exploration (collecting new information) and exploitation (using existing information) is the main difference between meta-heuristic algorithms. These two components work together with the selection of the best solutions to ensure convergence towards optimality. Additionally, incorporating randomization in the exploitation process helps prevent solutions from becoming trapped in local optima and also enhances their diversity. Overall, this class of algorithms is capable of exploring the entire search space and providing an optimal solution within an adequate amount of time.

2.1.2 Evolutionary Algorithms

One of the most typical classes of metaheuristic algorithms is EAs. The objective of EAs is to model natural evolution, with survival of the fittest as the core concept [10]. Generally, survival is achieved through reproduction. Offspring result from the combination of genetic material from two (or more) parents, aiming to inherit the best traits from each. Those who inherit negative traits may be weaker and could

lose the battle for survival.

The process of evolutionary algorithms often follows a structured scheme. Initially, a fixed number of solutions, also called *individuals* or *chromosomes*, are randomly generated. Each individual is then evaluated using a *fitness function*, which measures their performance in solving the given problem. Next comes the reproduction phase, characterized by two operators, i.e, *recombination* and *mutation*. In recombination, some individuals are selected as parents to generate one or more offspring for the next *generation*. On the other hand, mutation is applied to a single chromosome and results in only one new child. Following evaluation, some individuals are selected for the next cycle, and this process repeats until a terminal condition is met, typically defined as a fixed number of generations. Finally, the individual with the highest fitness value is decided as the best found solution to the problem. Figure 2.1 depicts the general scheme of EAs.

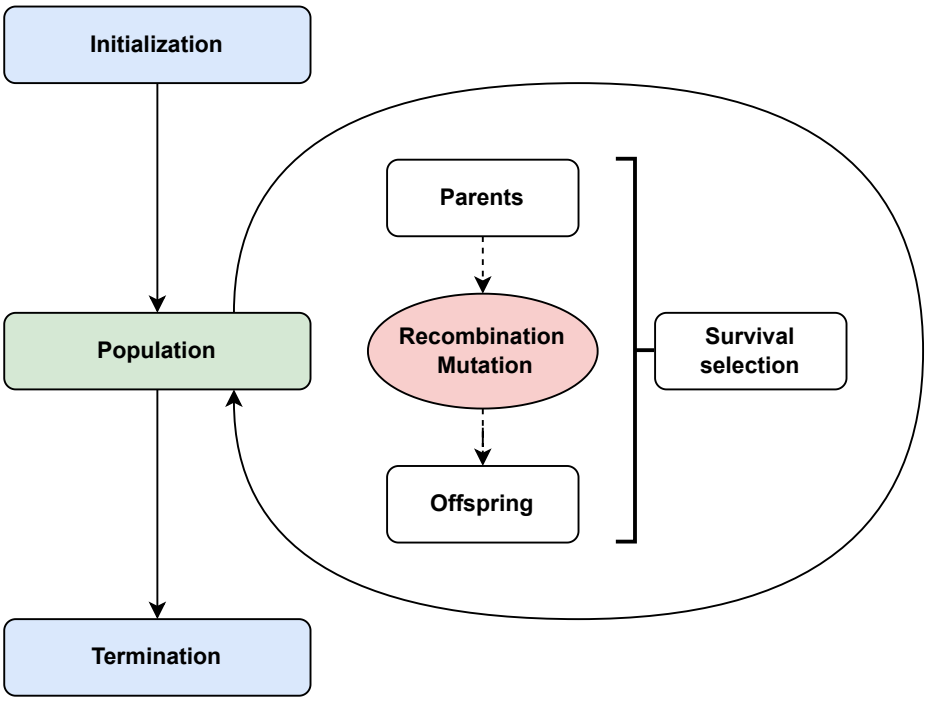

Figure 2.1: Overall structure of an evolutionary algorithm

Over the years, different classes of EAs have been developed [11], [12]. Some

typical examples include:

- **Genetic Algorithm (GA):** A computational model that simulates the process of genetic evolution.
- **Genetic programming:** An extension of GA where the individuals are represented as programs, commonly structured as trees.
- **Evolutionary programming:** Based on simulating adaptive behaviors observed in evolution.
- **Evolution strategies:** Aims to model the strategic parameters that control variation in evolution.
- **Differential evolution:** Similar to genetic algorithms, but it differs in the mechanisms used for reproduction.
- **Co-evolution:** A process in which individuals develop traits for survival through cooperation or competition with each other.
- **Cultural evolution:** A model that examines the evolution of a population's culture and its impact on the genetic and phenotypic evolution of individuals.

2.1.3 Genetic Algorithm

Genetic Algorithm (GA) is one of the most popular representations of EAs. First proposed by John Holland in 1992 [13], GA has been widely applied to solve a host of complex real-world problems as well as computationally theoretical ones. Inspired by Darwin's Theory of Evolution, the algorithm commences with an initial population that evolves through reproduction across successive generations. During this iterative process, beneficial genetic material is preserved, leading to fitter individuals with respect to the target environment, while gradually eliminating weaker

Algorithm 1 The skeleton of GA

Input: An optimization problem
Output: The best solution found for this problem

```
P(0) ← Generate an initial population with N individuals
for each individual p_i in P(0) do
    Evaluate the fitness of p_i
end for
t ← 1
while terminate conditions are not satisfied do
    P'(t) ← Select individuals from P(t) for reproduction
    O(t) ← Generate offspring with N individuals using reproduction on P'(t)
    for each individual o_i in O(t) do
        Evaluate the fitness of o_i
    end for
    P(t + 1) ← Select N fittest individuals from (P(t) ∪ O(t))
    t ← t + 1
end while
```

ones. The optimization of evolutionary progress ensures that later generations are improved compared to previous ones. Algorithm 1 illustrates the GA framework, with detailed explanations provided in subsections below.

Chromosome Encoding

GA utilizes a population of individuals, where each individual represents a potential solution to a given problem. The characteristics of an individual are represented by a chromosome, which can be partitioned into two types of evolutionary information [11]:

- **Genotype:** Refer to an individual's genetic makeup inherited from its parents. In other words, genotypes serve as a mechanism for preserving the experiential knowledge accumulated from previous generations.
- **Phenotype:** Defined as the behavioral traits of an individual within a specific environment.

Genotype and phenotype always have a close relationship with each other. One

important consideration when searching for a method to encode a solution is how to map genotypes to phenotypes. Meanwhile, the representation should exhibit high locality, meaning that a slight change to the genotype should lead to a small change to the phenotype as well as the fitness value [8]. In a scenario for generating game content, the genotype represents the instructions for creating a game level, while the phenotype refers to the actual game level itself.

Encoding a solution is considered the most challenging step in GA. This step requires a suitable encoding method so that each chromosome accurately represents a corresponding solution to the real-world problem. There are various ways to represent a chromosome, such as using a bit string or a multidimensional array of numbers. Figure 2.2 illustrates an example of encoding game level structures. On the left, the genotype is represented as a binary matrix, where the value of 1 denotes a brick tile, and 0 indicates empty space. On the right, the phenotype is shown as the corresponding game level.

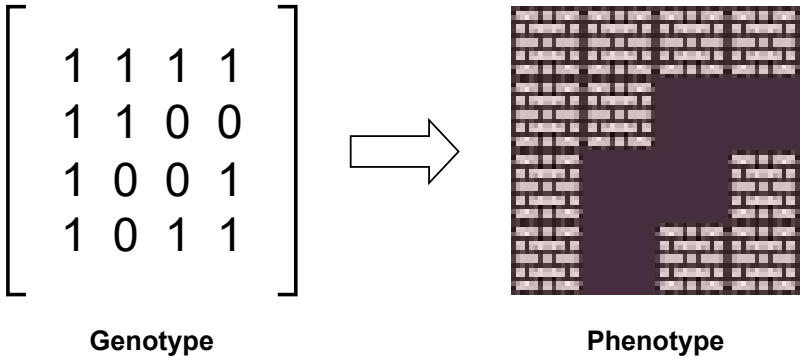

Figure 2.2: An example of encoding game level structures

Fitness Function

The fitness function is a vital component of a GA as it translates a chromosome into a scalar value. Each chromosome represents a potential solution, and the fitness function evaluates the quality of that solution. Additionally, all evolutionary

operators rely on the fitness evaluation of chromosomes. For instance, selection operators use the fitness values of individuals to choose parents for reproduction. When designing a fitness function, it is important to consider its practicality and implementation costs.

Initial Population

Initialization is the process of selecting individuals for the first population. The conventional method of forming the initial population is to randomly choose gene values from a predetermined set of values. However, in some cases, it is beneficial to use heuristic approaches based on available information about the search space to focus on selecting potentially good solutions. Due to the large dimension of the search space, not all elements may have an opportunity to be selected, which can lead to the population prematurely converging on a local optimum. Therefore, the size of the initial population is crucial for overall performance, which affects both accuracy and the time required for convergence.

Selection Operators

The selection operator takes the lead in guaranteeing that the next generation in the evolutionary progress is always better than the previous one. New generations are formed primarily through reproduction and elitism. In reproduction, the selection operator aims to choose fitter individuals to become parents. Similarly, the elitism operator retains a set of the fittest individuals to the next generation, facilitated by the selection operator.

Generally, some of the most commonly used selection operators include:

- **Random Selection:** All individuals are selected randomly, giving them an equal probability of being chosen, regardless of their qualities.
- **Roulette Selection:** The i^{th} individual in the population of N chromosome

has a selection probability p_i based on their fitness value $f(i)$ (See Equation 2.1). A random number r is generated between 0 and 1. The i^{th} individual is selected if $r < \sum_{j=1}^{i} f(j)$.

$$p_i = \frac{f(i)}{\sum_{j=1}^{N} f(j)} \tag{2.1}$$

- **Tournament Selection:** A group of individuals is randomly selected and compared in pairs. From these k individuals, the best one is chosen. The advantage of tournament selection is that the weakest individuals are excluded from selection, while the best individual will not dominate the reproduction process.

- **Rank-Based Selection:** The probability of being selected is based on the ranking of fitness values rather than the values themselves. This approach prevents a highly fit individual from dominating the selection process while providing a more balanced opportunity for all individuals in the population.

Reproduction Operators

Crossover and mutation are two essential operators in the reproduction process that help produce new offspring from selected parents. Crossover involves creating new offspring by combining the genetic material of two parents. The design of crossover operators may vary or be modified based on the specific characteristics of the problem being addressed. In contrast, mutation randomly alters the values within a chromosome's genes. While the crossover aims to preserve the dominant traits of the parents in the offspring, mutation introduces new genetic material into the population.

Terminate Condition

An ideal termination condition for GA is when the fitness value of the individuals reaches a predetermined optimal value for the objective function. However, due to the inherent randomness of this algorithm, meeting this condition is often very challenging and, in many cases, can be impossible because of convergence problems. Consequently, more commonly used termination conditions include: exceeding a specified CPU time limit, reaching a specific total number of population fitness evaluations, or when the population diversity decreases, indicating that the algorithm has converged below a certain threshold.

2.2 Multifactorial Optimization & Multifactorial Evolutionary Algorithm

Inspired by the bio-cultural concept of multifactorial inheritance, MFEA was the canonical algorithm designed to solve the Multifactorial Optimization (MFO) [7]. MFO, a specific formulation within EMTO, is characterized by the use of a USS and task-specific evaluation through factorial metrics.

MFO can be modeled as a multitasking environment, where K independent optimization tasks are to be performed simultaneously. Initially, there is no knowledge of relationships among tasks, and the goal is to completely and concurrently optimize each task, leveraging the implicit parallelism of population-based search. Assume that all tasks are maximization problems. The j^{th} task T_j has a search space X_j and an objective function $f_j : X_j \to \mathbb{R}$, and can be subject to various equality and/or inequality constraints. Accordingly, MFO is defined as an evolutionary multitasking paradigm that operates the implicit parallelism of population-based search, which

aims to find a set of feasible solutions $\{x_1, x_2, \ldots, x_{K-1}, x_K\}$ that satisfy:

$$x_i = \arg\max f_i(x), \quad i = 1, 2, \ldots, K \tag{2.2}$$

The design of MFO solvers, and particularly MFEA, is structured around four fundamental components:

- *Factorial Cost* f_j^i: For each individual p_i in population P, the factorial cost f_j^i represents its objective function value when evaluated on task T_j.
- *Factorial Rank* r_j^i: The factorial rank r_j^i indicates the ranking of individual p_i on task T_j, determined by sorting individuals in ascending order of their factorial cost f_j^i.
- *Scalar Fitness* φ_i: The scalar fitness φ_i of individual p_i is calculated based on its best factorial ranks across tasks: $\varphi_i = \frac{1}{\min_{j\in\{1,\ldots,K\}} r_j^i}$. A higher scalar fitness indicates the significance of an individual within the unified population.
- *Skill Factor* τ_i: The skill factor τ_i identifies the specific task to which individual p_i is assigned, corresponding to the task where it demonstrates the best performance.

During the evolutionary process, individuals are differentiated according to their assigned skill factors, which partitions the unified population into task-specific subpopulations responsible for optimizing their respective tasks. Furthermore, MFEA incorporates two key cultural effects, i.e., *assortative mating* and *vertical cultural transmission*, to promote implicit knowledge transfer across tasks. These mechanisms enable genetic material to be exchanged between individuals associated with different skill factors, thereby facilitating cross-task interaction and enhancing the overall search capability. Overall, the procedure of MFEA largely follows the conventional framework of GA, as shown in Algorithm 2.

Algorithm 2 The skeleton of MFEA

Input: A set of K optimization problems
Output: A set of solutions found for each problem

$P(0) \leftarrow$ Generate an initial population with N individuals
for each individual p_i in $P(0)$ **do**
    Evaluate p_i with respect to all K tasks
end for
Calculate the skill factor τ_i of each individual p_i in $P(0)$
$t \leftarrow 1$
while terminate conditions are not satisfied **do**
    $O(t) \leftarrow$ Generate offspring with N individuals using *assortative mating* on $P(t)$
    for each individual o_i in $O(t)$ **do**
        Assign the skill factor τ_i by *vertical cultural transmission*
        Evaluate o_i on task τ_i only
    end for
    $P(t+1) \leftarrow$ Select N fittest individuals from $(P(t) \cup O(t))$
    $t \leftarrow t+1$
end while

2.2.1 Unified Search Space

In the context of evolutionary multitasking, different optimization tasks usually involve heterogeneous characteristics, such as varying dimensionalities, variable ranges, or encoding schemes. To enable effective knowledge transfer across tasks while preserving their individual characteristics, MFEA constructs a unified representation referred to as USS.

Let D_j denote the dimensionality of the j^{th} task. The dimensionality of the USS is therefore defined as:

$$D_{max} = \max_{1 \leq i \leq K} D_i \tag{2.3}$$

Accordingly, each individual in the population is encoded as a vector:

$$I = (c_1, c_2, \ldots, c_{D_{\max}}) \tag{2.4}$$

where each variable is normalized to a common interval, e.g., $c_i \in [0, 1]$. When

evaluating an individual with respect to task T_j, only the first D_j variables are utilized, while the remaining dimensions are ignored. This unified encoding allows all tasks to exist within a single population, thereby facilitating implicit genetic transfer during the evolutionary process.

2.2.2 Task-Specific Decoding

All individuals in the unified population are assigned a skill factor τ_i, which specifies the task for which they are primarily responsible. During the evaluation phase, each individual is decoded with respect to its designated task, such that only the dimensions relevant to that task are taken into account. The factorial cost is then computed by applying the corresponding task-specific fitness function. This mechanism ensures that, although individuals share a common representation, the optimization objectives of each task are still maintained.

2.2.3 Assortative Mating

Assortative mating is one of the key cultural mechanisms in MFEA that controls how genetic operators occur within the unified population. It directly influences the degree of knowledge transfer across tasks. In complex multitask environments, unrestricted crossover between individuals assigned to different tasks may result in *negative transfer* (See Section 2.2.5), where incompatible genetic material degrades performance. Conversely, strictly isolating tasks would eliminate the potential benefits of cross-task knowledge sharing. Assortative mating balances this trade-off by probabilistically controlling inter-task reproduction.

In MFEA, the skill factor τ_i represents an individual's cultural bias, meaning that individuals tend to prefer mating with those of the same cultural background [7]. During reproduction, two randomly selected parents can freely undergo crossover if they share the same skill factor. When their skill factors differ, crossover happens

Algorithm 3 The basic rules of assortative mating

Input: Two randomly selected parents p_a and p_b with skill factors τ_a and τ_b
Output: Two offspring o_a and o_b

```
rand ← Draw a random number between 0 and 1
if (τ_a == τ_b) or (rand < rmp) then
    o_a, o_b ← p_a and p_b crossover to produce two offspring
else
    o_a ← p_a undergoes mutation
    o_b ← p_b undergoes mutation
end if
```

only with a predefined random mating probability (rmp); otherwise, mutation is applied independently. Algorithm 3 outlines the fundamental rules of assortative mating.

The parameter $rmp \in [0, 1]$ controls the probability of implicit knowledge transfer and balances exploration and exploitation within the search space. A value of rmp close to 0 restricts crossover primarily to individuals with identical skill factors, which facilitates the exploitation of small areas within the search space. In contrast, a value of rmp close to 1 permits frequent random mating, which enhances global exploration. This approach also helps prevent solutions from being trapped in local optima and facilitates their escape. While insufficient inter-task mating may lead to the loss of diverse genetic material from other cultural backgrounds, excessive crossover between weakly related tasks may introduce disruptive genetic material. Therefore, the selection of rmp is very important, since it significantly influences convergence behavior, transfer effectiveness, and search stability. In practice, rmp is typically determined empirically and may vary depending on the problem.

2.2.4 Vertical Cultural Transmission

Vertical cultural transmission is another essential cultural mechanism in MFEA, designed to determine how offspring inherit skill factors after assortative mating. While assortative mating controls how genetic material is exchanged, vertical cul-

Algorithm 4 The procedure rules of vertical cultural transmission

Input: An offspring o and its single parent p_a, or two parents p_a and p_b with skill factors τ_a and τ_b
Output: Offspring o is evaluated based on its skill factor τ_o

```
if o has two parents then
    rand ← Draw a random number between 0 and 1
    if rand < 0.5 then
        τ_o ← τ_a
    else
        τ_o ← τ_b
    end if
else
    τ_o ← τ_a
end if
Evaluate o for its associated task
```

tural transmission regulates how task identity is passed to the next generation. In multitask optimization, it is generally unrealistic for an individual to perform competitively on all tasks simultaneously. Evaluating each individual on every task would also incur a high computational cost. Therefore, MFEA adopts this strategy in which each offspring is evaluated only on the task it is most likely to perform well.

This mechanism takes place after crossover or mutation, when each offspring must be assigned a skill factor that specifies the task on which it will be evaluated. If both parents share the same skill factor or there is only one parent, as in the case of mutation, the offspring inherits that skill factor. When parents possess different skill factors, the offspring randomly inherits one of the skill factors with equal probability. The detailed procedure of this mechanism is presented in Algorithm 4.

2.2.5 Negative Transfer and Solutions to this phenomenon

MFEA is typically applied to black-box optimization problems, where neither the properties of the objective functions nor the relationships between tasks are known in advance. As a result, there is no guarantee that knowledge transferred between

tasks will be beneficial. When tasks are not closely related, such interactions may introduce misleading information into the search process. This issue is referred to as *negative transfer*, which poses a possible weakness of MFEA. This issue becomes more apparent in complex multitasking environments, particularly as the number of tasks increases.In such cases, the problem setting approaches what is commonly referred to as Many-task Optimization (MaTO), which is a multitasking environment containing more than 3 tasks [14]. Prior studies have shown that the performance of MFEA may degrade as the number of tasks grows, especially when task similarity is low. Therefore, MFEA can effectively handle only two or three tasks simultaneously [15].

To address negative transfer, several approaches have been proposed in the literature, which can generally be categorized into two main directions [14]. The first focuses on adaptive control of knowledge transfer, typically by adjusting the *rmp*. One of the first methods that followed this was the Multifactorial Evolutionary Algorithm with Online Transfer Parameter Estimation (MFEA-II), proposed by Bali *et al.* [16]. MFEA-II introduces a dynamic mechanism that adapts the degree of transfer using a learned *rmp* matrix. Similarly, Thang *et al.* [17] proposed LSA-MFEA, which adjusts the *rmp* based on the historical memory of successful *rmp*, inspired by adaptation mechanisms in Differential Evolution. The second direction is to limit transfer to similar tasks only. One representative method is Group-based MFEA [18], which clusters tasks into groups and restricts knowledge transfer within each group. In addition, Zheng *et al.* [19] proposed the Self-regulated EMTO algorithm, which employs ability vectors to characterize each individual's performance across tasks and adaptively regulate the intensity of knowledge transfer.

In this thesis, the second approach is adopted through a compatibility-based task grouping strategy, where tasks are grouped according to the similarity of their content and control search spaces. This design provides a simple yet effective mechanism

to encourage positive transfer while reducing harmful interactions. More advanced techniques for adaptive or learning-based transfer are not considered in this work, as the primary objective is to evaluate the feasibility of applying EMTO to PCG under a unified representation framework.

2.3 Search-based methods in Procedural Content Generation

Search-based methods, or SBPCG, constitute one of the most established approaches in PCG, where content is generated through iterative optimization guided by an explicit evaluation function. One typical example of a PCG problem solved by many search-based methods is generating levels for Super Mario Bros (SMB), a 2D platformer game developed by Nintendo [20] (See Figure 2.3).

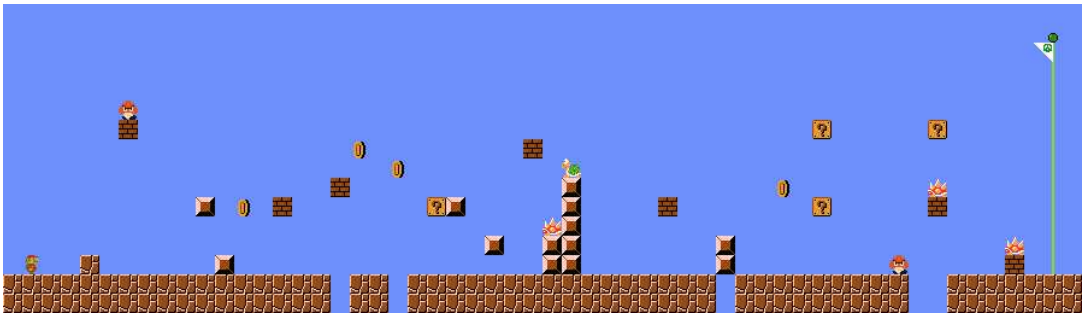

Figure 2.3: An example of a SMB level

Dahlskog *et al.* devised a novel pattern-based representation for a GA used to generate SMB levels [21]. In this approach, levels are encoded as fixed-length strings, where each symbol corresponds to one of 24 reusable vertical slices extracted from original SMB patterns. Each chromosome consists of 200 symbols, and the resulting level is formed by concatenating the corresponding slices to construct a complete level. Beyond direct evolutionary encoding, SBPCG has also been combined with generative models. Volz *et al.* utilized Covariance Matrix Adaptation Evolution

Strategy (CMA-ES) to evolve new SMB levels generated by a Generative Adversarial Network (GAN), which is a neural generative model trained on an existing SMB level [22]. Although this latent-variable evolution approach successfully produces high-quality levels, the authors reported that it often delivers broken structures. SBPCG has further extended to coevolutionary settings. Instead of generating levels alone, Flimmel *et al.* used coevolution to train both Artificial Intelligence (AI) players and level generators for SMB [23], where AI players and level generators are evolved and evaluated through mutual adaptation. Additionally, the authors proposed a new metric for evaluating levels that employs a compression algorithm to automatically assess the visual complexity of each level.

Recently, Khalifa *et al.* introduced the PCG Benchmark to facilitate systematic evaluation of generative algorithms across a diverse set of PCG problems [5]. They conducted experiments with three algorithms, namely a random generator, an evolution strategy, and a GA, on the proposed benchmark. The findings demonstrated that the GA generally outperformed the other algorithms. In addition, the authors noted that some problems were easier to solve than others, and generating large levels posed a challenge for all compared algorithms.

Despite the prevalence of evolutionary approaches in single-task PCG, EMTO methods have received little attention in this domain. In particular, MFEA has been extensively studied for continuous and combinatorial optimization problems [24], such as cloud server, neural network, and industrial engineering, yet its application to PCG remains largely unexplored. One reason may be the heterogeneity of PCG representations and evaluation metrics, which complicates the construction of a unified search space required for multitask optimization. Furthermore, most PCG studies focus on individual game problems, whereas MFEA is inherently designed for simultaneous optimization of multiple tasks. With the introduction of the PCG Benchmark, which consolidates diverse PCG problems under standardized evalua-

tion protocols, applying multitask evolutionary approaches has become increasingly feasible. Therefore, this thesis presents the first investigation of employing MFEA within the PCG Benchmark.

2.4 Knowledge Transfer in Procedural Content Generation

Beyond search-based approaches, knowledge transfer across different PCG problems has also been explored in machine learning studies. For example, Adam *et al.* [25] surveyed Procedural Content Generation via Machine Learning (PCGML), providing an overview of emerging machine learning approaches to PCG. In their work, the authors defined PCGML as the process of generating game content through machine learning models trained on existing content. In 2023, Anurag *et al.* [26] introduced a survey on Procedural Content Generation through Knowledge Transformation. Inspired by the concept of generating new content by repurposing derived knowledge, this survey categorizes various approaches within this paradigm and proposes a definition and framework for describing them. Transfer Reinforcement Learning (RL) was also investigated by *et al.* in [27]. Their paper addressed scalability challenges in RL for 3D PCG by utilizing a curriculum-based transfer learning approach. In general, these methods focus on reusing prior knowledge across domains through pretraining or sequential adaptation. In contrast, this thesis investigates EMTO through MFEA, where multiple tasks are optimized concurrently within a unified search process, without the need for prior training.

3 Procedural Content Generation Benchmark

3.1 Overview

The PCG Benchmark [5] is a framework released in 2025 to support the performance evaluation of generative algorithms. It consists of twelve distinct PCG problems, each featuring its own representation and control parameters. These problems encompass a variety of games across different genres and objectives, including rule generation, pattern generation, platformers, roguelikes, puzzles, and more. Three evaluation functions are provided within the framework to assess quality, diversity, and controllability criteria of the generated content.

- **Quality** calculates the percentage of input content meeting quality criteria. This constraint usually relates to the game's playability or its specific rule conditions.

- **Controllability** can only be measured if the problem comes with control parameters, often provided by developers, designers, or users. This function computes the percentage of input content that adheres to those specific constraints. The idea behind this criterion is to create a general generator that can adapt to various target parameters, allowing it to generate different types of games without requiring changes to its underlying code.

- **Diversity** measures the diversity of the generated content. A generator is deemed ineffective or low diversity if it produces identical content or only slight variations in every attempt.

Alongside these evaluation functions come three different fitness functions used to assess chromosomes during the evolution process.

- **Quality Fitness (Q):** The fitness of a chromosome is calculated from the quality of its generated content.

$$f(c_i, p_i, \mathbb{C}) = q(c_i) \tag{3.1}$$

where c_i represents the content being evaluated, p_i denotes the control parameter associated, $\mathbb{C}$ is the total population, and $q(c_i)$ is the quality value for input content.

- **Quality then Controllability Fitness (QT):** This fitness function prioritizes maximizing quality first, followed by controllability only if the quality is optimal.

$$f(c_i, p_i, \mathbb{C}) = \begin{cases} \frac{1}{2} q(c_i) & \text{if } q(c_i) < 1 \\ \frac{1}{2}(q(c_i) + t(c_i, p_i)) & \text{if } q(c_i) = 1 \end{cases} \tag{3.2}$$

where c_i is the content being evaluated, p_i is the control parameter associated, $\mathbb{C}$ is the total population, $q(c_i)$ is the quality value for input content, and $t(c_i, p_i)$ indicates the controllability value for the input content corresponding to the control parameter.

- **Quality, Controllability, then Population Diversity Fitness (QTD):** This fitness function initially aims to maximize quality as in the previous function. Once an optimal solution regarding quality is achieved, it then focuses on enhancing controllability. Finally, if the solution is optimal in terms of both

quality and controllability, it seeks to optimize population diversity.

$$f(c_i, p_i, \mathbb{C}) = \begin{cases} \frac{1}{3} q(c_i) \\ \frac{1}{3}\big(q(c_i) + t(c_i, p_i)\big) \\ \frac{1}{3}\big(q(c_i) + t(c_i, p_i) + d(c_i, \mathbb{C})\big) \end{cases} \tag{3.3}$$

where

1. $q(c_i) < 1$ and $t(c_i, p_i) < 1$,
2. $q(c_i) = 1$ and $t(c_i, p_i) < 1$,
3. $q(c_i) = 1$ and $t(c_i, p_i) = 1$.

where c_i is the content being evaluated, p_i refers to the control parameter associated, $\mathbb{C}$ indicates the total population, $q(c_i)$ is the quality value for input content, $t(c_i, p_i)$ is the controllability value for the input content corresponding to the input control parameter, and $d(c_i, \mathbb{C})$ measures the uniqueness of c_i within $\mathbb{C}$.

These functions are designed to capture different aspects of the generator's capabilities. While the feasibility of the solution is the primary focus, controllability and diversity are also taken into account and evaluated across the entire population, which provides further insight into the generator's search behavior.

Additionally, the PCG Benchmark introduces two types of possibility spaces, namely *content space* and *control space*. These spaces define the search space for each problem, and they typically differ from one another. This design provides the framework with the flexibility required to support diverse forms of content across different games. Possibility space [28] refers to the complete set of all game content that could theoretically exist within a game's rules and representation. This includes every level layout, map, rule configuration, or game artifact that can be described

using the game's components and constraints. In general, one generator can explore only a subset of this space, producing content according to its design and algorithms, while the possibility space represents the broader range of all potential content that could exist in the game.

3.2 Procedural Content Generation Problems

There are a total of twelve PCG problems inside the benchmark. Each problem has its own representation, configurable control parameters, quality, diversity, and controllability criteria. A brief overview of each problem is provided below.

Arcade Rules

Arcade Rules focuses on developing rule sets for 2D arcade games. The solution is represented as a dictionary of integer values specifying game properties, such as initial coordinates, movement rules, collision rules, and win conditions. Seven quality criteria guarantee that the generated games are playable, winnable, and losable by different types of agents. A binary grid with varying sizes, representing the level layout, serves as the control parameter. The benchmark provides three instances of this problem, using 7×7, 15×7, and 15×15 layouts, as shown in Table 3.1.

Table 3.1: All instances of Arcade Rules problem

Instances	arcade-v0	arcade-wide-v0	arcade-large-v0
Width x height	7x7	15x7	15x15

Binary

Binary requires generating fully connected 2D mazes composed of empty and solid tiles. A primary quality constraint is that the longest path between any two empty

tiles must exceed a threshold equal to the sum of the maze's width and height. Three instances of binary mazes of different sizes are included in the benchmark, generating 14x14, 28x14, and 28x28 mazes, as shown in Table 3.2.

Table 3.2: All instances of Binary problem

Instances	binary-v0	binary-wide-v0	binary-large-v0
Width x height	14x14	28x14	28x28

Building

Building focuses on constructing 3D Lego-like structures using four block types (1×1, 1×3, 3×1, and 3×3). Solutions must satisfy constraints on total block usage, spatial dimensions, and minimum height. Four control parameters regulate the relative proportions of the block types. The benchmark provides four instances with varying structure sizes and block counts ranging from 40 to 360 (See Table 3.3).

Table 3.3: All instances of Building problem

Instances	building-v0	building-large-v0	building-tall-v0	building-huge-v0
Width	7	11	7	11
Height	12	12	24	24
Length	7	11	7	11
Blocks	40	180	80	360

Dangerous Dave

Dangerous Dave focuses on generating playable levels for a simplified platform game, where the player's missions include avoiding spikes, collecting diamonds, and reaching the exit. Levels must satisfy constraints related to tile distribution, solvability by AI agents, required jumps, and diamond reachability. The problem involves five

control parameters, including start and exit coordinates, as well as the number of diamonds. Three instances differ primarily in map size and minimum jump requirements (See Table 3.4).

Table 3.4: All instances of Dangerous Dave problem

Instances	ddave-v0	ddave-complex-v0	ddave-large-v0
Width x height	11x7	11x7	17x11
Jumps	2	6	10

Elimination

Elimination generates sequences of letters for a word game. The objective is to produce sequences containing at least one short and one long valid word while preventing longer words from appearing. The problem has five quality constraints related to word frequency categories, and a control parameter limits consecutive valid word segments. Table 3.5 shows three instances of this problem, each presenting a progressively increasing level of difficulty.

Table 3.5: All instances of Elimination problem

Instances	elimination-v0	elimination-easy-v0	elimination-hard-v0
Letters	8	6	10
Short percentage	0.5	0.2	0.8
Long percentage	0.7	0.4	0.9

Isaac

Isaac requires generating fully connected dungeon layouts for a simplified roguelike game. Levels must include specific room types, namely a starting room, a boss room,

a treasure room, and a shop room, and satisfy connectivity and distance constraints. The number of rooms acts as the control parameter. Three instances are provided with progressively increasing map sizes, which indicate the total number of active rooms (See Table 3.6).

Table 3.6: All instances of Isaac problem

Instances	isaac-v0	isaac-medium-v0	isaac-large-v0
Width x height	4x4	6x6	8x8
Map size	6	12	24

Lode Runner

Lode Runner focuses on producing playable puzzle platformer levels. Generated levels must contain a minimum number of gold and enemies, and at the same time meet seven quality constraints, including reachability and the number of tiles constraints. Control parameters include quantities of ladder and rope tiles. Table 3.7 depicts three instances of this problem, which share the same map size but differ in the number of minimum gold and enemies.

Table 3.7: All instances of Lode Runner problem

Instances	loderunner-v0	loderunner-gold-v0	loderunner-enemies-v0
Width	32	32	32
Height	22	22	22
Gold	6	18	6
Enemies	3	3	12

MiniDungeons

MiniDungeons aims to produce solvable dungeon levels where the player must survive enemies throughout the level. Quality constraints determine the minimum number of tiles required for each type in the level, ensure the dungeon is connected, and require that some monsters be defeated along the shortest route to the exit. Control parameters define the start and exit locations as well as treasure targets. The first two instances of this problem have the same map size but differ in the enemy counts, while the last instance features a larger map size and a greater number of enemies (See Table 3.8).

Table 3.8: All instances of MiniDungeons problem

Instances	mdungeons-v0	mdungeons-large-v0	mdungeons-enemies-v0
Width x height	8x12	16x24	8x12
Enemies	8	16	16

Super Mario Bros

Super Mario Bros (SMB) represents its levels as sequences of vertical slices extracted from the original game. Generated levels must be playable by an A* agent and adhere to structural constraints, including flat areas, unbroken pipes, and floating enemies. Three control parameters regulate the number of enemies, coins, and required jumps performed by the AI agent. SMB instances vary only by the number of slices required to create a level, with the smallest requiring 50 slices and the largest requiring 150 slices (See Table 3.9).

Table 3.9: All instances of SMB problem

Instances	smb-v0	smb-medium-v0	smb-small-v0
Width	150	100	50

Sokoban

Sokoban, which originates from Japan, requires generating solvable block-pushing puzzles. Levels must contain required tile types and be solvable within a minimum of 10 moves. The control parameter specifies the target number of crates. Each instance has its own map size and difficulty, and is provided with predefined solver configurations, as illustrated in Table 3.10.

Table 3.10: All instances of Sokoban problem

Instances	sokoban-v0	sokoban-complex-v0	sokoban-large-v0
Width x height	5x5	5x5	8x8
Difficulty	1	4	3
Solver	5000	20000	10000

Talakat

Talakat generates bullet patterns for a shoot-em-up game. Quality constraints evaluate properties of the patterns, such as spawner limits, bullet density, and spatial distribution. The control parameter defines the temporal distribution of bullets within a one second of gameplay, based on a 1D array. The map size remains consistent across all instances. Each instance varies in terms of spawner complexity, coverage, and the number of bullets (See Table 3.11).

Table 3.11: All instances of Talakat problem

Instances	talakat-v0	talakat-simple-v0	talakat-complex-v0
Width x height	200x300	200x300	200x300
Spawner complexity	10	5	20
Max health	150	150	150
Min bullets	50	15	60
Coverage	0.95	0.75	0.95

Zelda

Zelda involves generating mazes containing a key, a door, and enemies. The objective is for the player to retrieve the key and reach the exit while avoiding enemies. Levels must be connected, with a required number of tiles of each type, and a minimum solution length required. Control parameters specify minimum distances between the start, key, and door locations. The benchmark consists of two instances with different sizes and a varying number of enemies, along with one instance that features a large number of enemies (See Table 3.12).

Table 3.12: All instances of Zelda problem

Instances	zelda-v0	zelda-enemies-v0	zelda-large-v0
Width x height	11x7	11x7	18x12
Enemies	3	12	8

4 Multifactorial Evolutionary Algorithm-Based for Procedural Content Generation

This section introduces the multitask approach for solving the PCG Benchmark in more detail. MFEA is implemented to generate content for a set of K PCG problems simultaneously. Each problem, or task, is defined as:

$$P^i = (c_r^i, p_r^i, Q^i, T^i, D^i), i = 1...K$$

where c_r^i and p_r^i denote the content and control spaces, and Q^i, T^i, and D^i are quality, controllability, and diversity functions, respectively.

4.0.1 Construction of Unified Search Spaces

To facilitate knowledge exchange among different tasks, traditional MFEA creates a shared environment by integrating all search spaces into a single unified representation. However, PCG problems in this benchmark often exhibit heterogeneous representations and various search space structures. Furthermore, each PCG task consists of two distinct search spaces, i.e., a content space that specifies the structure of the generated artifact, and a control space that governs generation parameters

under game-specific constraints. These two spaces differ fundamentally in both semantics and structure and therefore cannot be meaningfully unified into a single search space.

To tackle this challenge, we propose a USS construction method specifically designed for PCG Benchmarks as follows. Given a set of PCG problems $\{P^i\}_{i=1}^K$ as input, the framework first analyzes the representational compatibility of their search spaces, ensuring that only tasks with compatible semantic interpretations are unified into a shared search space. Rather than requiring identical representations, compatibility is defined by preserving the minimal structural invariants necessary for valid decoding and meaningful genetic transfer, as follows.

1. Integer-valued spaces are considered compatible regardless of their ranges. The unified bounds are defined as:

$$l = \min_i l_i, \quad u = \max_i u_i$$

   where $[l_i, u_i]$ is the range of task i.

2. Real-valued spaces are following the same approach as integer-valued spaces.

3. Array-based spaces can be unified despite differing dimensions by adopting the maximum dimension across tasks. The unified array shape is computed as:

$$\mathbf{d} = (\max_i d_{i,1},\ \max_i d_{i,2},\ \ldots,\ \max_i d_{i,m})$$

   where $d_{i,j}$ is the length along dimension j for task i, and m is the maximum number of dimensions among all tasks.

4. In contrast, dictionary-based spaces require consistent key sets, as mismatched keys would invalidate the semantic interpretation of generated content. While each key k retains its original meaning, it is worth noting that its corresponding

value now is the unified subspace across tasks, constructed following the same unification rules as described above. Nested dictionaries are recursively unified using the same rules

Based on these criteria, problems that satisfy these compatibility conditions are grouped (See Algorithm 5), while incompatible tasks are optimized separately. In this work, task compatibility is primarily determined by the content search space, since content representations must be structurally aligned to enable valid genetic exchange. If the corresponding control spaces are also compatible, those tasks are referred to as *fully compatible*. Otherwise, control parameters are evolved independently.

For each group of fully compatible tasks, two independent USS are constructed, namely content USS and control USS. Then, a single population is evolved to solve multiple similar PCG tasks within one group simultaneously, where each task adds a distinct factor that influences the evolutionary process.

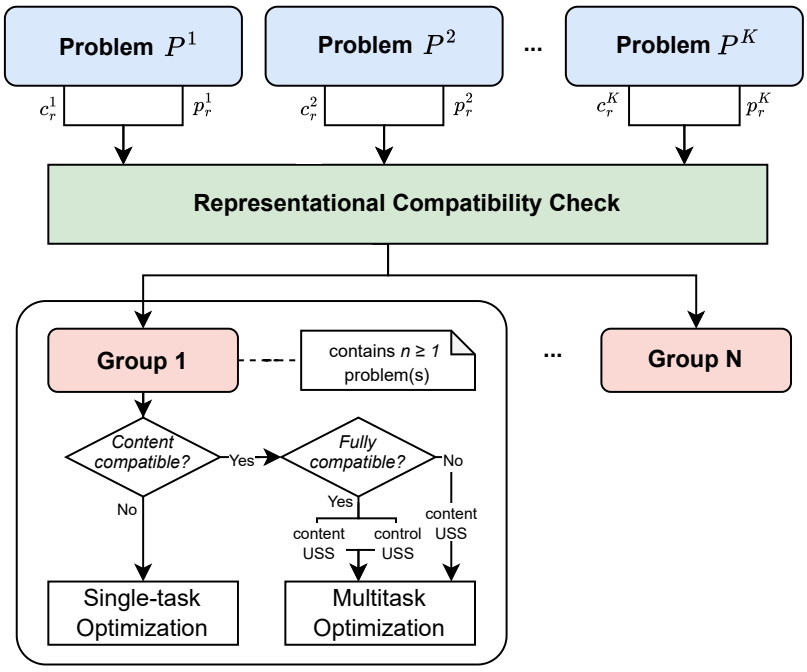

Figure 4.1: Overview of the proposed framework for unified search space construction.

Algorithm 5 Representational Compatibility Check and USS Construction

Input: K problems $\{P^1, \ldots, P^K\}$, each with content space c_r and control space p_r.
Output: N groups of compatible problems $\{G_1, \ldots, G_N\}$.

```
Initialize Groups ← ∅
for each problem P^i do
    Assigned ← false
    for each existing group G_n do
        Let P^n be a representative problem from G_n
        if P^i is representationally compatible with P^n then
            Add P^i to G_n
            Update G_n unified search space to encompass P^i
            Assigned ← true
            break
        end if
    end for
    if not Assigned then
        Create new group G_new = {P^i}
        Add G_new to Groups
    end if
end for
return Groups
```

This design enables the construction of USS by ensuring representational compatibility across tasks, while also reducing the risk of negative transfer arising from structurally incompatible representations. The overall structure of the proposed approach for USS construction is illustrated in Figure 4.1.

4.0.2 Unified Chromosome Representation and Decoding

Let U^c and U^p denote the unified content and control search spaces constructed for a group of fully compatible PCG problems, respectively. Each individual in the population is represented as:

$$I = (I^c, I^p), \quad I^c \in U^c,\ I^p \in U^p,$$

where I^c is the content vector and I^p is the control parameter vector. When a chromosome is initialized, these vectors are sampled from their respective USS, which

Algorithm 6 Decoding a Chromosome into a Task Solution

Input: A chromosome $I = (I^c, I^p)$; $P^j = (c_r^j, p_r^j)$
Output: A solution $I^{(j)} = (I^{c(j)}, I^{p(j)})$ for P^j

```
// Decode the content segment
I^{c(j)} ← Decode(I^c, c_r^j)
// Decode the control segment
I^{p(j)} ← Decode(I^p, p_r^j)
// Combine segments
I^{(j)} ← (I^{c(j)}, I^{p(j)})
return I^{(j)}
```

defines both their representational structure and value ranges. While the chromosome's representational structure remains fixed throughout evolution, its contents evolve under the action of evolutionary operators.

For a task P^j, its solution, denoted as $I^{(j)}$, is obtained by decoding a unified representation $I = (I^c, I^p)$ into its original search spaces (c_r^j, p_r^j). Decoding is applied independently to the content and control segments of the chromosome, as summarized in Algorithm 6, using the same mapping rules. For integer-valued spaces, decoding maps unified chromosome values into the valid range via modulo-based projection. Binary spaces are treated as a special case of integer spaces, where values are projected onto $\{0, 1\}$. Real-valued spaces are mapped linearly to the task-specific bounds, preserving the continuous structure of the segment. For array-based spaces, the decoding process first adapts the unified representation to the shape of the considered task by truncating each dimension to the required size. After that, each element in the resulting array is decoded according to the underlying value space of the array. Finally, for dictionary-based spaces, decoding is applied independently to each key. Each key maintains its semantic meaning, and its associated value is decoded recursively based on the corresponding subspace defined by the considered task.

4.0.3 Evolutionary Operator

Since chromosomes are encoded using varying structures, the generator adopts the uniform crossover and mutation operators employed in [5]. Uniform crossover exchanges individual elements between two parents based on a fixed probability. Figure 4.2 demonstrates an example of the adopted uniform crossover operator. The process begins with the initialization of two offspring, $O1$ and $O2$, which are generated as deep copies of parents $P1$ and $P2$, respectively. Genetic recombination is then performed through a stochastic swap mechanism, in which each element of $O1$ is exchanged with the corresponding element from $P2$ based on a predefined probability. Simultaneously, $O2$ undergoes an identical exchange with $P1$.

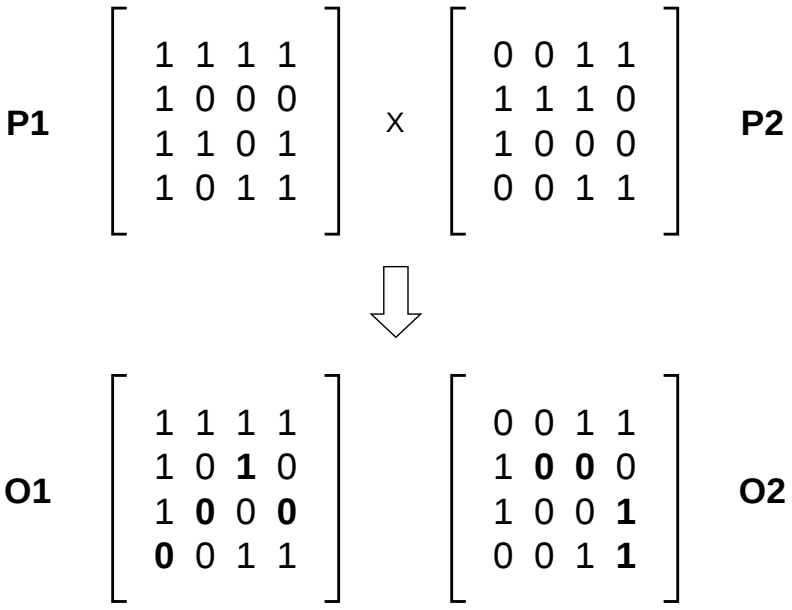

Figure 4.2: An example of the adopted Uniform crossover

Mutation, on the other hand, is applied to each element of the chromosome also with a small probability, where mutated elements are replaced by newly sampled values from a unified search space. An example of this mutation is shown in Figure 4.3, where the offspring O is generated by replacing one element from its parent P.

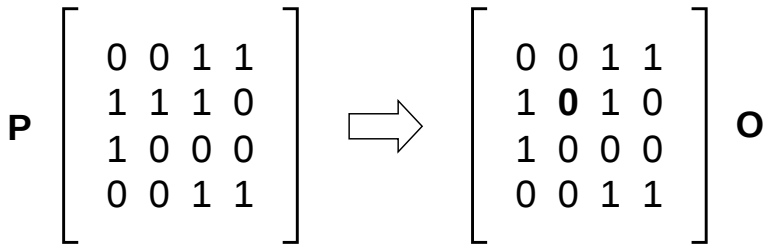

Figure 4.3: An example of the adopted mutation approach

These operators ensure that all offspring remain valid while preserving their structure. Note that the operators only perform on both content and control parts when tasks are fully compatible; otherwise, the control parameter stays intact. Parent selection and assortative mating adhere to the standard MFEA procedure.

5 Experiments and Results

5.1 Experimental Setups

This section assesses the effectiveness of the approach using the PCG Benchmark [5]. As mentioned above, this benchmark includes 12 PCG problems, each containing 3 to 4 instances with varying sizes and constraints. Noting that in [5], the evaluation is conducted using a single instance per PCG problem. This thesis, in contrast, considers all instances for each problem, as they can form a multitasking environment suitable for the proposed approach.

To evaluate the performance of the MFEA solver within the proposed framework, this thesis adopts the single-task GA from [5] as a baseline, given its experimentally confirmed efficiency on the benchmark. The same parameters as the original paper are utilized to create a similar experimental environment, thereby maximizing the performance of the comparison algorithm. For the employed MFEA, the *rmp* parameter configuration was selected as 0.2, which indicated stable and competitive performance on the benchmark. This selection is also consistent with commonly used values in the MFEA literature.

In this experiment, each algorithm was simulated 10 times on the same computer (Intel Core i7 - 2.10 GHz, 32 GB RAM). To ensure the results align with baseline standards, this thesis adopted the evaluation budget established in [5]. This budget utilizes 20000 evaluations, with a population size of 100 individuals evolving over 200

generations. For GA, these evaluations are dedicated entirely to a single problem instance. For MFEA, the evaluations are shared across K problems. Although this setup presents a challenge for MFEA, resulting in a 66% - 75% reduction in the evaluation budget per task compared to the GA baseline, it allows for testing whether inter-task transfer in MFEA can offset this limitation.

Regarding experimental criteria, this thesis focuses on the mean value of fitness results and the number of feasible, controlled, and unique solutions achieved. Besides, the Wilcoxon rank-sum test and Vargha-Delaney A_{12} (See subsections 5.1.1, 5.1.2) are utilized to further assess the performance difference between the compared algorithm. Finally, the source codes were implemented in Python. [1]

5.1.1 Wilcoxon rank-sum test

The Wilcoxon rank-sum test, proposed by Frank Wilcoxon [29], is a non-parametric hypothesis test used to determine whether two independent, or unpaired, samples originate from the same distribution. It is mathematically equivalent to the Mann–Whitney U test, developed by Henry Mann and Donald R. Whitney [30], and the two are often collectively referred to as the Wilcoxon–Mann–Whitney test.

Given two independent samples $X = x_1, x_2, \ldots, x_n$ and $Y = y_1, y_2, \ldots, y_m$, the Wilcoxon rank-sum test first combines them into a common list. The combined data are then sorted in ascending order and assigned ranks $1, 2, \ldots, n + m$, which indicates their magnitude. If two or more samples have equal magnitude, average ranks are assigned. The rank sum value of X is then calculated as:

$$W_X = \sum_{i=1}^{n} rank(x_i)$$

Next, the hypothesis pair is specified, including:

[1] The source code and experimental setup used in this thesis can be provided upon reasonable request via email: longnh.mso@gmail.com

- Null hypothesis ($H0$): The means of the two sample groups X and Y are the same, or in other words, the data in X and Y are samples from continuous distributions with equal medians.
- Alternative hypothesis ($H1$): The means of the two sample groups X and Y are different, or in other words, X and Y have unequal medians.

Under this null hypothesis ($H0$), the expectation and variance of W are, respectively:

$$\mu_W = \frac{n(n+m+1)}{2}, \quad \sigma_W = \sqrt{\frac{nm(n+m+1)}{12}}$$

For sufficiently large sample sizes, the distribution of W_X can be approximated by a normal distribution. The standardized test statistic is computed as:

$$Z = \frac{W_X - \mu_W}{\sigma_W}$$

In practice, a continuity correction may be applied when computing the Z-value. The resulting Z-score is then used to obtain the corresponding p-value from the standard normal distribution. Finally, the decision is made by comparing the p-value with a predefined significance level α (e.g., $\alpha = 0.05$). If $p < \alpha$, the null hypothesis $H0$ is rejected, indicating a statistically significant difference between the two samples. Otherwise, there is insufficient evidence to conclude a difference between them.

In this study, the Wilcoxon rank-sum test is employed to compare the performance of MFEA and GA across multiple independent runs. The comparison is non-trivial, as GA optimizes each task independently, whereas MFEA simultaneously optimizes multiple tasks within a shared population, resulting in different evaluation dynamics and stochastic behaviors. Since the resulting performance metrics may not follow a normal distribution and the number of runs is relatively limited,

a non-parametric test is more appropriate than parametric alternatives. Therefore, the Wilcoxon rank-sum test provides a statistically sound method to assess whether the observed differences in solution quality between the two algorithms are significant rather than due to random variation.

5.1.2 Vargha-Delaney A_{12} effect size

Vargha-Delaney A_{12} effect size [31] is considered a robust test when assessing randomized algorithms. Unlike hypothesis tests such as the Wilcoxon rank-sum test, which only indicate whether a difference is statistically significant, the Vargha-Delaney A_{12} measure quantifies the magnitude of that difference.

Given two independent samples $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_m\}$, the A_{12} statistic is defined as:

$$A_{12} = P(x > y) + \frac{1}{2}P(x = y)$$

where $P(x > y)$ denotes the probability that a randomly chosen value $x \in X$ is greater than a randomly chosen value $y \in Y$, while $P(x = y)$ represents the probability that the two values are equal. In this context, the probability is estimated by comparing all possible pairs (x_i, y_j), where $x_i \in X$ and $y_j \in Y$. The value of A_{12} ranges from 0 to 1. A value of 0.5 indicates no difference between the two samples. Values greater than 0.5 suggest that sample X tends to produce larger values than Y, while values less than 0.5 indicate the opposite.

In this study, the Vargha-Delaney A_{12} statistic is used alongside the Wilcoxon rank-sum test to provide a more comprehensive comparison between MFEA and GA. While the Wilcoxon test determines whether the observed differences are statistically significant, the A_{12} measure quantifies the strength and direction of those differences.

5.2 Experimental Scenarios

Two experimental scenarios were designed for the used multitasking algorithm, i.e, MFEA, in order to investigate its performance and the knowledge transfer between tasks under different multitasking conditions, as follows.

- **Scenario 1:** Fully compatible multitasking environment: All instances belonging to the same PCG problem are optimized simultaneously. These instances share the same content and control representations but differ in problem size or control parameters, which may allow potential knowledge transfer across instances. The results obtained are further compared against GA.

- **Scenario 2:** Content-compatible multitasking environment: Instances from different PCG problems are optimized together when their content search spaces are compatible and of comparable size. This scenario is designed to evaluate whether meaningful knowledge transfer can occur across various games. Specifically, this study consider grid-based level generation problems, i.e., *Binary*, *Dangerous Dave*, *MiniDungeons*, *Sokoban*, and *Zelda*, where content is represented as two-dimensional arrays of discrete tiles encoding navigational structure, obstacles, and interactive entities. Although originating from different games, these problems share some common design abstractions, such as reachability, obstacle placement, and player-goal interaction, making them suitable candidates for multitask knowledge transfer. To control for differences in problem scale, the selected instances are further divided into subgroups according to their grid sizes, allowing the effects of multitasking to be evaluated under comparable levels of problem complexity. This design can mitigate the risk of negative transfer caused by extreme dimensional imbalance. Also noting that only grid-based problems are considered, as other problems in the benchmark have heterogeneous representations and belong to different genres (e.g.,

generating rules, 3D building blocks, bullet patterns) that are incompatible or meaningless to solve together.

Table 5.1: Multitask instance groupings for the experimental scenarios

Scenario	**Task Bundle**
Fully compatible	arcade-v0_arcade-wide-v0_arcade-large-v0; binary-v0_binary-wide-v0_binary-large-v0; building-v0_building-large-v0_building-tall-v0_building-huge-v0; ddave-v0_ddave-complex-v0_ddave-large-v0; elimination-v0_elimination-easy-v0_elimination-hard-v0; isaac-v0_isaac-medium-v0_isaac-large-v0; loderunner-v0_loderunner-gold-v0_loderunner-enemies-v0; mdungeons-v0_mdungeons-enemies-v0_mdungeons-large-v0; smb-v0_smb-small-v0_smb-medium-v0; sokoban-v0_sokoban-complex-v0_sokoban-large-v0; talakat-v0_talakat-simple-v0_talakat-complex-v0; zelda-v0_zelda-enemies-v0_zelda-large-v0
Content-compatible	ddave-v0_mdungeons-v0_sokoban-v0_zelda-v0; binary-v0_ddave-large-v0_zelda-large-v0

Table 5.1 summarizes the multitask groupings used in the two experimental

scenarios. Each row corresponds to a bundle of PCG Benchmark instances that are solved together within a single multitasking run. Instance identifiers within a bundle are separated by underscores and indicate different variants of the same problem or content-compatible problems, depending on the scenario.

5.3 Experimental Results

5.3.1 Performance comparison on Fully compatible tasks

Figure 5.1 illustrates the improvement in maximum Quality fitness across generations for the two algorithms. We observe that both algorithms demonstrate highly efficient search capabilities in certain problems, i.e., *Arcade Rules*, *Binary*, and *Talakat*. In these environments, maximum or near maximum quality fitness is achieved before generation 50 in the evolutionary process. Some other problems, such as *Isaac*, *Lode Runner*, and *Zelda*, witnessed competitive convergence trends between the two algorithms. In these cases, GA often shows faster initial improvement, while MFEA gradually catches up by generation 100, eventually resulting in nearly identical fitness levels. This is likely attributable to positive transfer, in which MFEA successfully leverages useful genetic material shared across related tasks.

In contrast, there are clear distinctions in more difficult environments. In *Super Mario Bros*, both algorithms struggle to meet quality constraints for most instances. However, GA achieves a significantly higher and faster fitness growth of 317% improvement for smb-small-v0. This advantage may be related to the smaller level size, which can simplify single-task optimization. A similar pattern can be seen in *Building*, with the wide shaded areas indicating high variance between the 10 runs for both algorithms. One possible explanation for MFEA's low performance here is the increased complexity of the unified search space, which comprises four instances with differing level sizes. *Dangerous Dave* is one of the most volatile problems. The

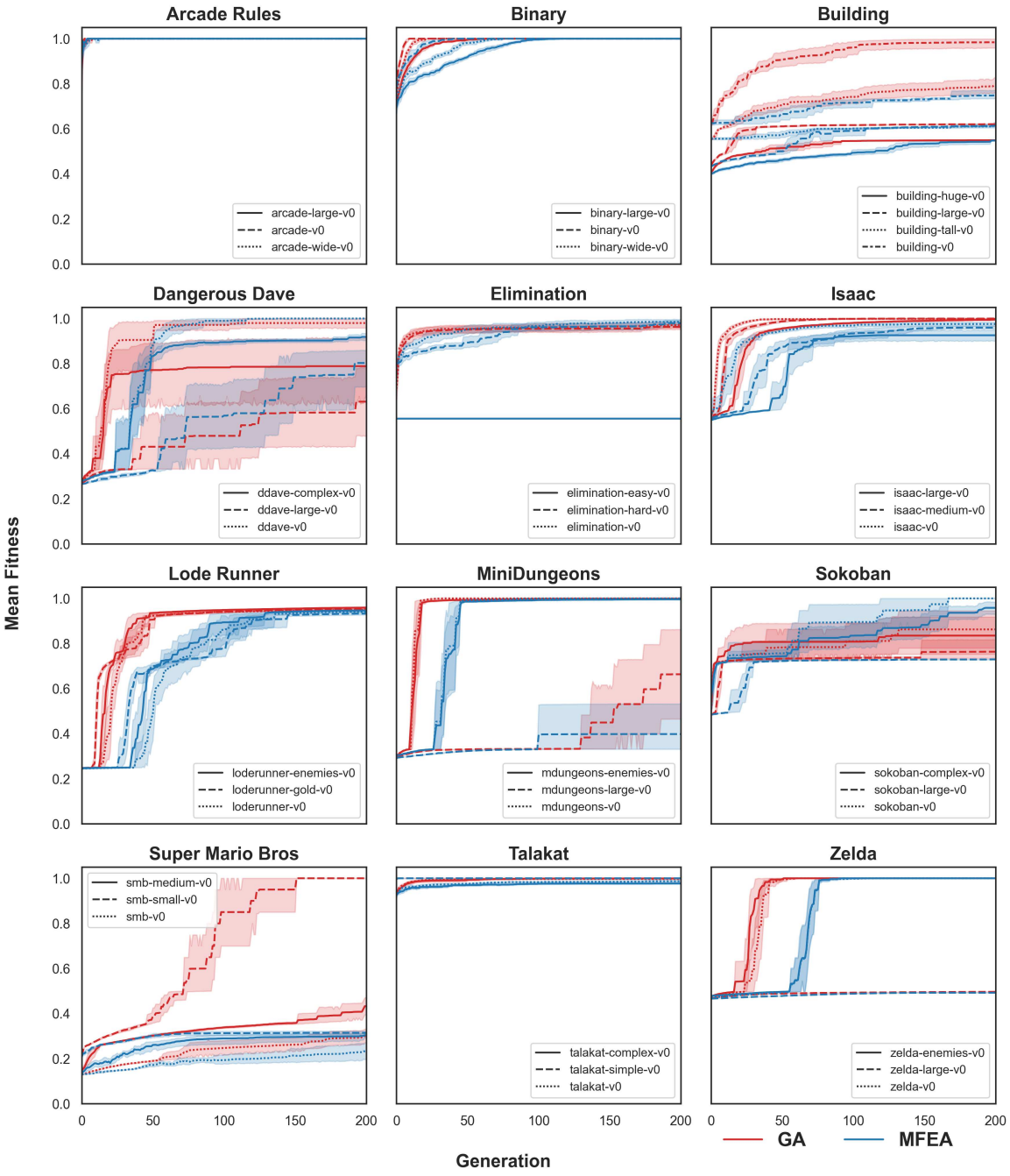

Figure 5.1: Evolutionary progress of the maximum Quality fitness of GA and MFEA. Results are averaged from 10 runs, with 95% confidence intervals as the shaded area.

large shaded areas for both methods indicate that the search is sensitive to initial conditions, with some runs performing much better than others. Nevertheless, this problem also provides clear evidence of positive transfer in MFEA, as demonstrated by improvement percentages exceeding 200% across all instances. Overall, MFEA tends to exhibit narrower shaded regions corresponding to the 95% confidence intervals compared to GA, suggesting more consistent performance across independent runs in several environments.

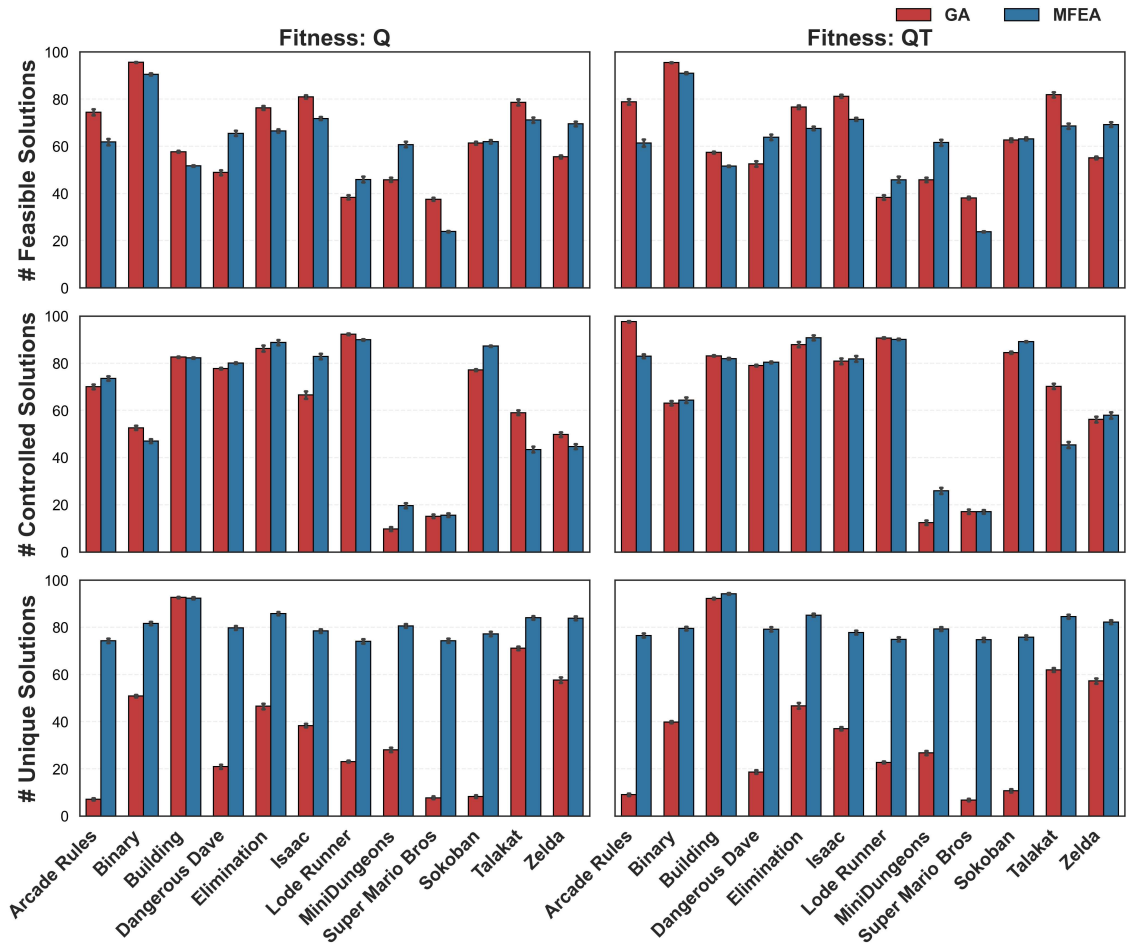

Figure 5.2: Number of solutions c_i in the 200^{th} generation that are feasible ($q(c_i) = 1$) and controlled ($t(c_i, p_i) = 1$) compared to the final population $\mathbb{C}$. Results are averaged from 10 runs across all instances per task, with 95% confidence intervals shown as error bars.

Figures 5.2 and 5.3 depict the average number of feasible individuals ($q(c_i) = 1$), unique individuals ($d(c_i, \mathbb{C}) = 1$) with $\mathbb{C}$ representing the final population in the

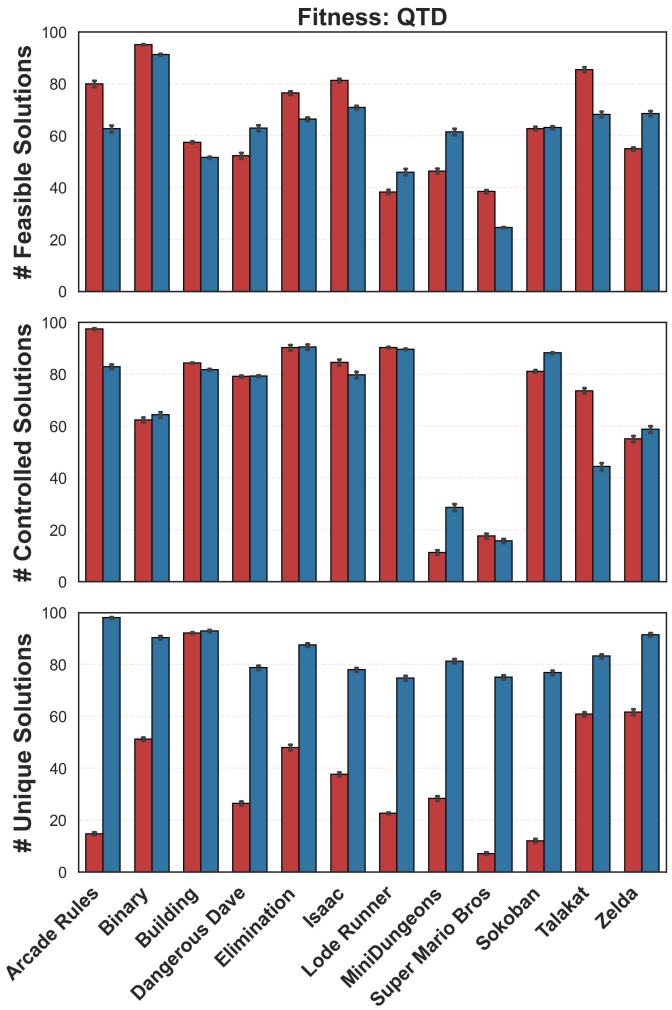

Figure 5.3: Number of solutions c_i in the 200^{th} generation that are unique ($d(c_i, \mathbb{C}) = 1$) compared to the final population $\mathbb{C}$. Results are averaged from 10 runs across all instances per task, with 95% confidence intervals shown as error bars.

same run), and controlled individuals that fulfill all controllability constraints for their paired control parameters p_i, $(t(c_i, p_i) = 1)$, across all instances of each task. In these figures, the error bars represent 95% confidence intervals derived from the standard error of the mean, which provide a visual indication of the algorithmic stability and performance variance between the GA and MFEA across the benchmark problems. Regarding the number of feasible solutions (first row of Figures 5.2 and 5.3), GA proves to have a slight advantage in problems such as *Arcade Rules*, *Binary*, *Isaac*, and *Talakat*, regardless of the fitness objective. However, MFEA remains highly competitive and actually outperforms GA in *Dangerous Dave* and *MiniDungeons*. Both algorithms show similar performance in generating individuals that meet controllability constraints, often exceeding 50 solutions out of 100 in most tasks (second row of Figures 5.2 and 5.3). Notable exceptions include *Super Mario Bros* and *MiniDungeons*, where both methods struggle. In *Super Mario Bros*, GA manages to produce slightly more controlled individuals than MFEA. Conversely, MFEA consistently achieved at least twice the number of controlled solutions compared to GA across all fitness targets in *MiniDungeons*. The most significant difference appears in the number of unique individuals (bottom row of Figures 5.2 and 5.3), where MFEA demonstrates its primary strength. In almost every environment, MFEA produces a significantly higher number of unique solutions compared to GA. Most notably, under the QTD fitness scheme (Figure 5.3), MFEA achieves nearly 100% uniqueness for *Arcade Rules*, *Binary*, *Building*, and *Zelda*, and maintains a significant advantage in more challenging games (*MiniDungeons*, *Super Mario Bros*, and *Sokoban*). This suggests that MFEA is effective at preserving population diversity, thus reducing the risk of premature convergence.

Finally, the Wilcoxon rank-sum test and Vargha-Delaney A_{12} measure were conducted to further assess the performance differences between MFEA and GA. The results are summarized in the win–tie–loss table (Table 5.2) and visualized in the

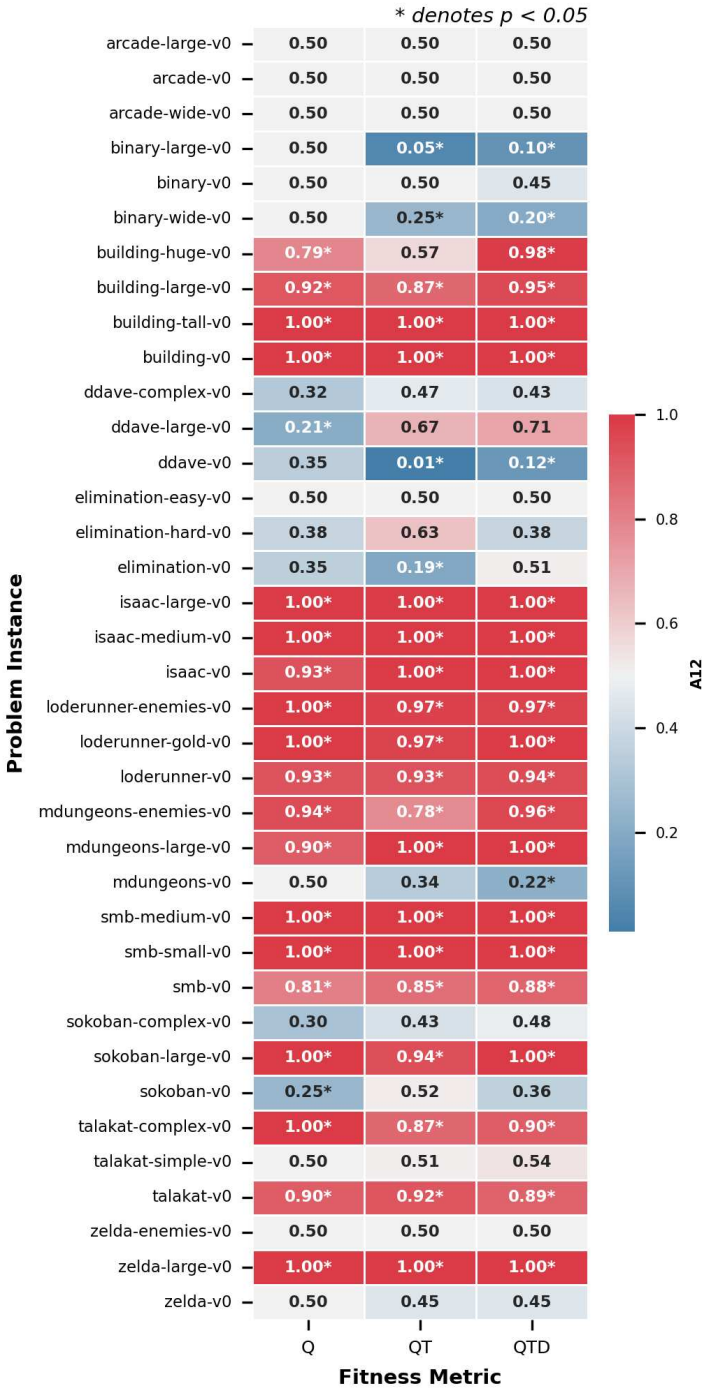

Figure 5.4: Statistical comparison of fitness performance via A_{12} effect sizes and Wilcoxon rank-sum tests. Red cells ($A_{12} > 0.5$) indicate that GA outperformed MFEA, while blue cells ($A_{12} < 0.5$) indicate the opposite. Statistically significant differences, determined by the Wilcoxon rank-sum test at $p < 0.05$, are marked with an asterisk (*).

heatmap shown in Figure 5.4. In Table 5.2, the positive decision (+) means that MFEA outperforms GA with statistical significance, whereas the negative decision (−) indicates the opposite. The tied decision (≈) denotes that no statistically significant performance difference was detected between the two algorithms. The results show that in 14 to 16 out of 37 instances, there is no statistically significant difference between MFEA and GA. In other words, in approximately 40% of the tasks, MFEA achieves the same quality as GA. As seen in the heatmap, although GA wins 18 to 19 instances, these wins are concentrated in particular problems, such as *Isaac*, *Lode Runner*, and *Super Mario Bros.* When viewed alongside population uniqueness (Figure 5.2 and 5.3), these fitness advantages appear to be associated with GA's tendency to prematurely converge on a single type of solution, leading to reduced content diversity. In contrast, MFEA maintains a significantly more diverse population across all tasks, regardless of whether it wins, loses, or ties.

Table 5.2: The performance of GA and MFEA for 10 independent runs with Wilcoxon rank-sum test ($p = 0.05$)

Fitness	**MFEA Wins (+)**	**Tied (≈)**	**GA Wins (−)**
Q	2	16	19
QT	4	15	18
QTD	4	14	19

Overall, for Scenario 1, which consists of fully compatible tasks, the results demonstrate that MFEA performs competitively with GA, even with a significant reduction (66% or 75% depending on problem) in evaluation budget. It achieves comparable or sometimes better solution quality while consistently maintaining higher population diversity, which shows its effectiveness as a strategy for PCG, where content variety plays a critical role. The observed convergence behavior across several problems suggests that positive knowledge transfer may occur, allowing MFEA to

leverage shared information among related environments. This indicates that fully compatible PCG tasks constitute a suitable setting for EMTO.

5.3.2 Evaluate multitask learning on Content-compatible tasks

Figure 5.5 illustrates the progression of maximum Q fitness over generations for two content-compatible environment bundles optimized using MFEA. In the first bundle of small instances (right plot of Figure 5.5), we can observe a clear sequence of trends suggesting that information is transferred from simpler to more complex tasks. Specifically, ddave-v0 is the first to reach a good quality point at 0.9 around generation 60. Shortly after it stabilizes, mdungeons-v0 experiences a sharp improvement between generations 65 and 75, rising from around 0.5 to 0.9. This increase may indicate that useful information discovered in one task may have been successfully transferred to another. Similarly, zelda-v0 shows little progress initially but experiences a sudden rise in fitness after generation 75, further supporting the presence of knowledge transfer.

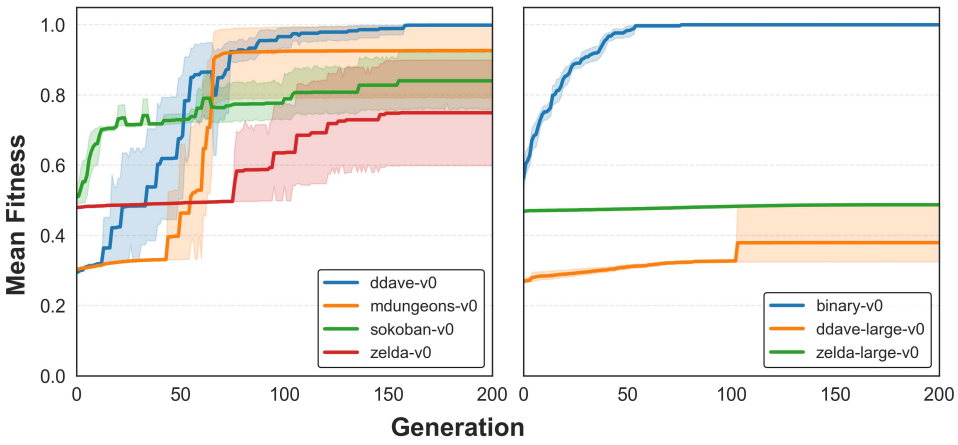

Figure 5.5: Evolutionary progress of the maximum Quality fitness of MFEA in Scenario 2. Results are averaged from 10 runs, with 95% confidence intervals as the shaded area.

In contrast, the second bundle (left plot of Figure 5.5), which contains three medium-sized instances, demonstrates a different learning behavior. binary-v0 con-

verges rapidly, reaching its maximum fitness extremely before generation 50. While ddave-large-v0 stays unchanged at approximately 0.32 for the first half of the evolutionary process, it undergoes a sudden increase to around 0.38 at generation 100. This delayed improvement suggests that knowledge extracted from the optimized binary-v0 may eventually be transferred to the larger task through successful genetic exchange. On the other hand, zelda-large-v0 remains almost flat for the entire 200 generations, which indicates that the knowledge required for this task is too distant from the other tasks in its bundle.

In comparison to the learning behavior of MFEA in Scenario 1, Scenario 2 exhibits clear signs of negative transfer, characterized by prolonged stagnation periods and slower fitness growth in most cases. Specifically, the late improvements in some instances suggest that while knowledge transfer is still possible between different games, the lack of task compatibility creates a transfer barrier that can only be overcome after significant independent evolution.

5.3.3 Representatives of generated content

This subsection presents examples of the best generated content that optimizes the Q fitness function for each PCG problem within the benchmark. Content marked with a red background indicates that it did not meet the quality constraints. To visually distinguish between varying levels of performance, the value v of the optimized function is mapped to a color gradient where lighter shades represent higher fitness ($1 > v > 0.75$) and darker red hues indicate lower fitness tiers ($0.25 \geq v \geq 0$) (See Figure 5.6). Additional content that optimizes the QT, as well as that which optimizes QTD, can be found in Appendix A.

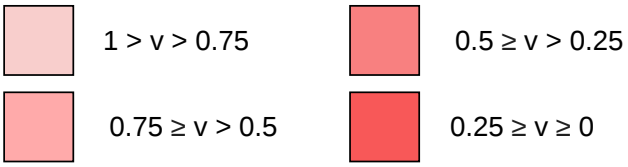

Figure 5.6: Color scale representing the ranges of level fitness values (v)

Representatives of generated content from Scenario 1

Figures 5.7 - 5.19 showcase some of the best content generated by GA and MFEA while optimizing the Q function across 12 problems in the PCG Benchmark according to Scenario 1 (See Section 5.2). For GA, the final level for each instance is determined by running each instance individually. In contrast, the resulting levels for MFEA are achieved by optimizing all instances simultaneously.

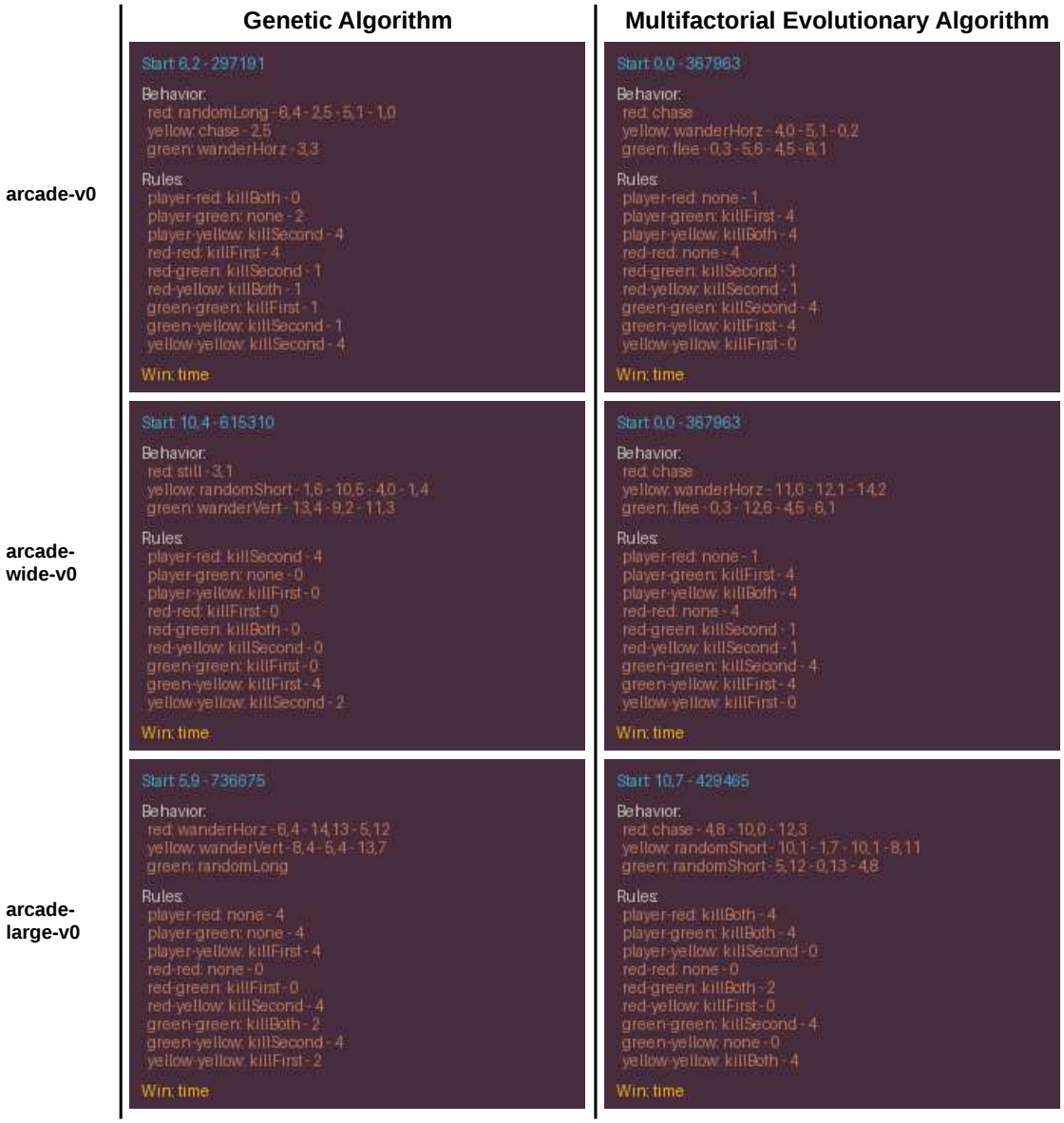

Figure 5.7: Representatives of the best content generated for two compared generators optimizing the Q function for Arcade Rules.

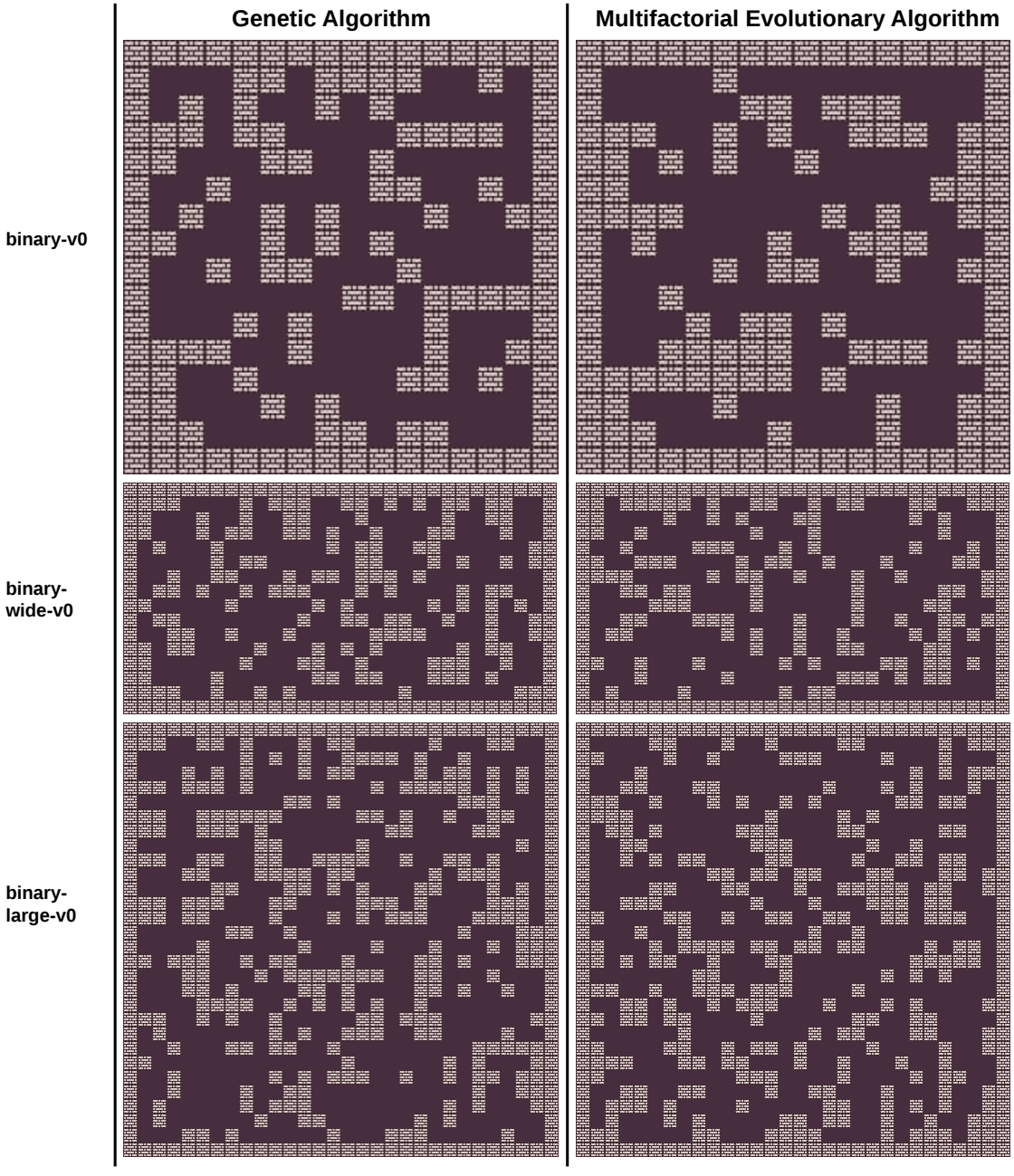

Figure 5.8: Representatives of the best content generated for two compared generators optimizing the Q function for Binary.

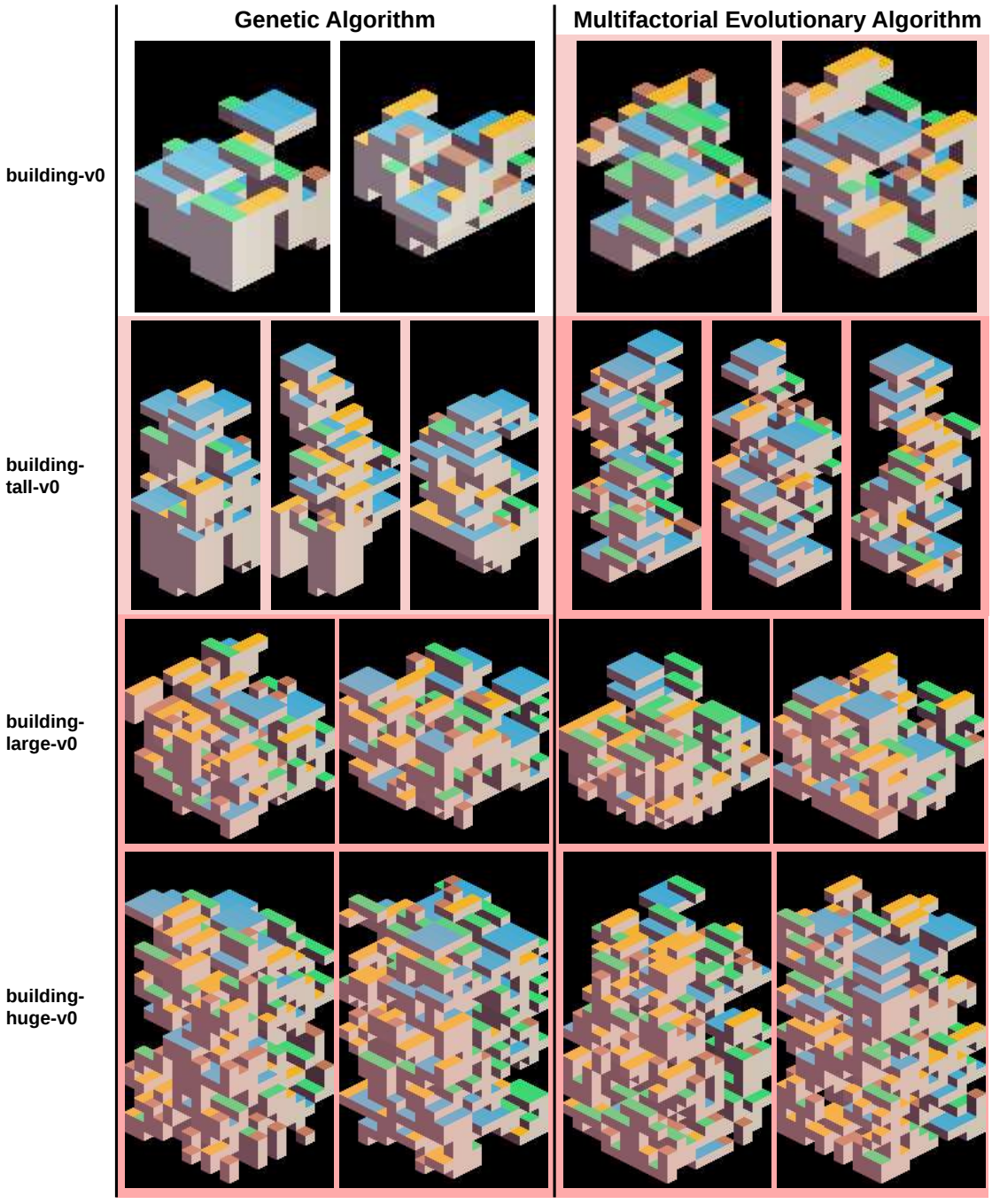

Figure 5.9: Representatives of the best content generated for two compared generators optimizing the Q function for Building. Content marked with a red background indicates that it did not meet the Q constraints.

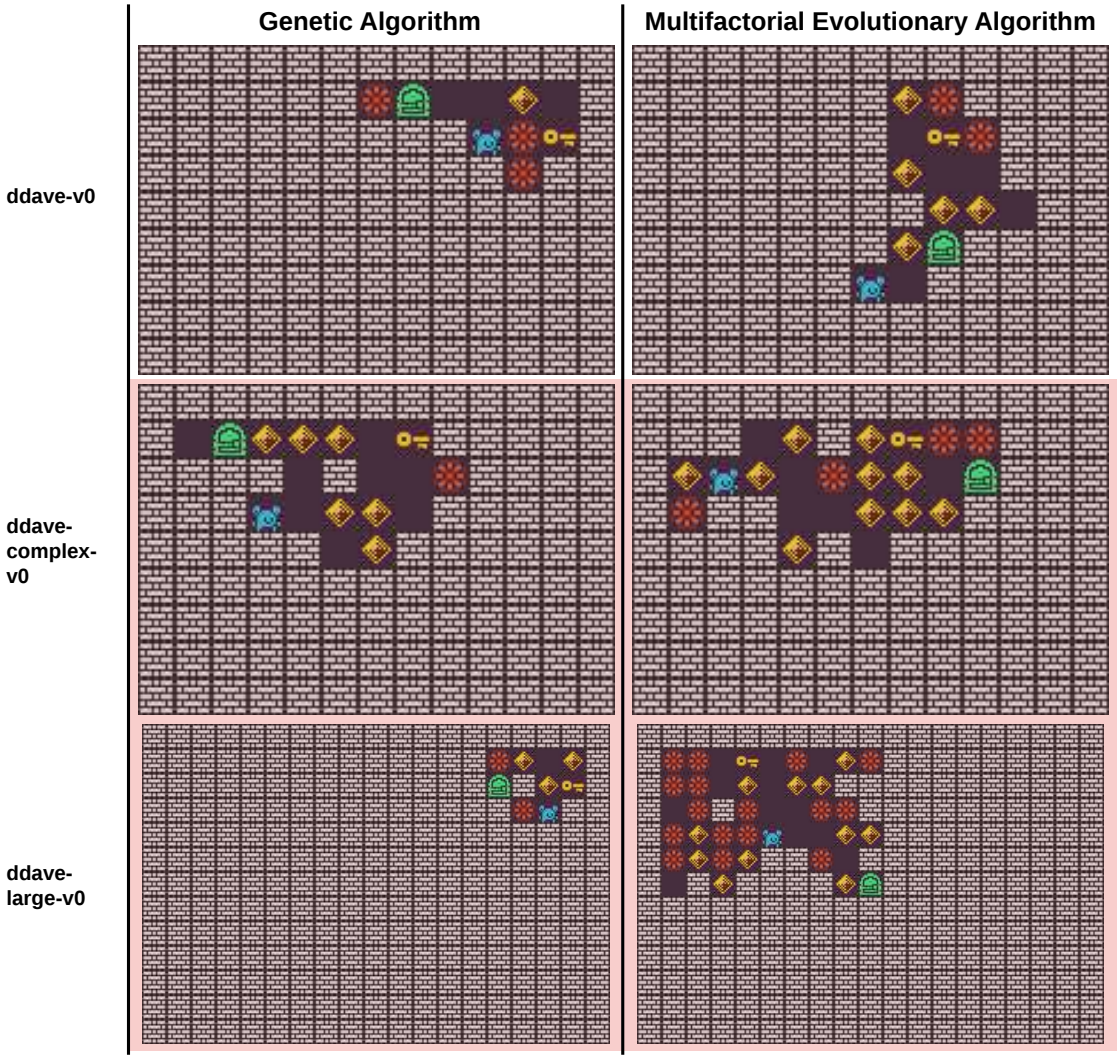

Figure 5.10: Representatives of the best content generated for two compared generators optimizing the Q function for Dangerous Dave. Content marked with a red background indicates that it did not meet the Q constraints.

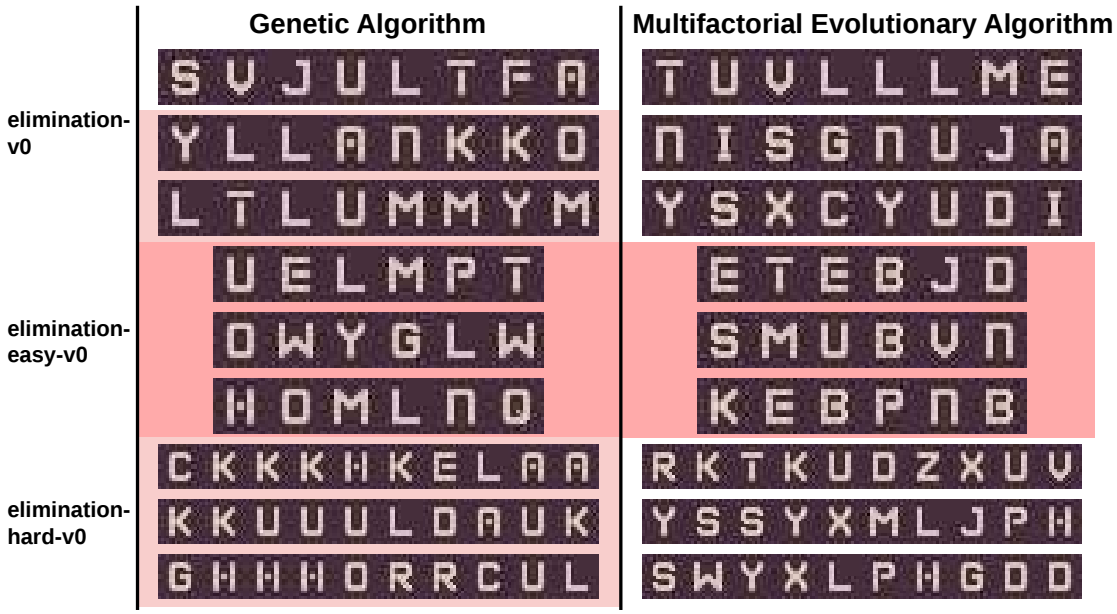

Figure 5.11: Representatives of the best content generated for two compared generators optimizing the Q function for Elimination. Content marked with a red background indicates that it did not meet the Q constraints.

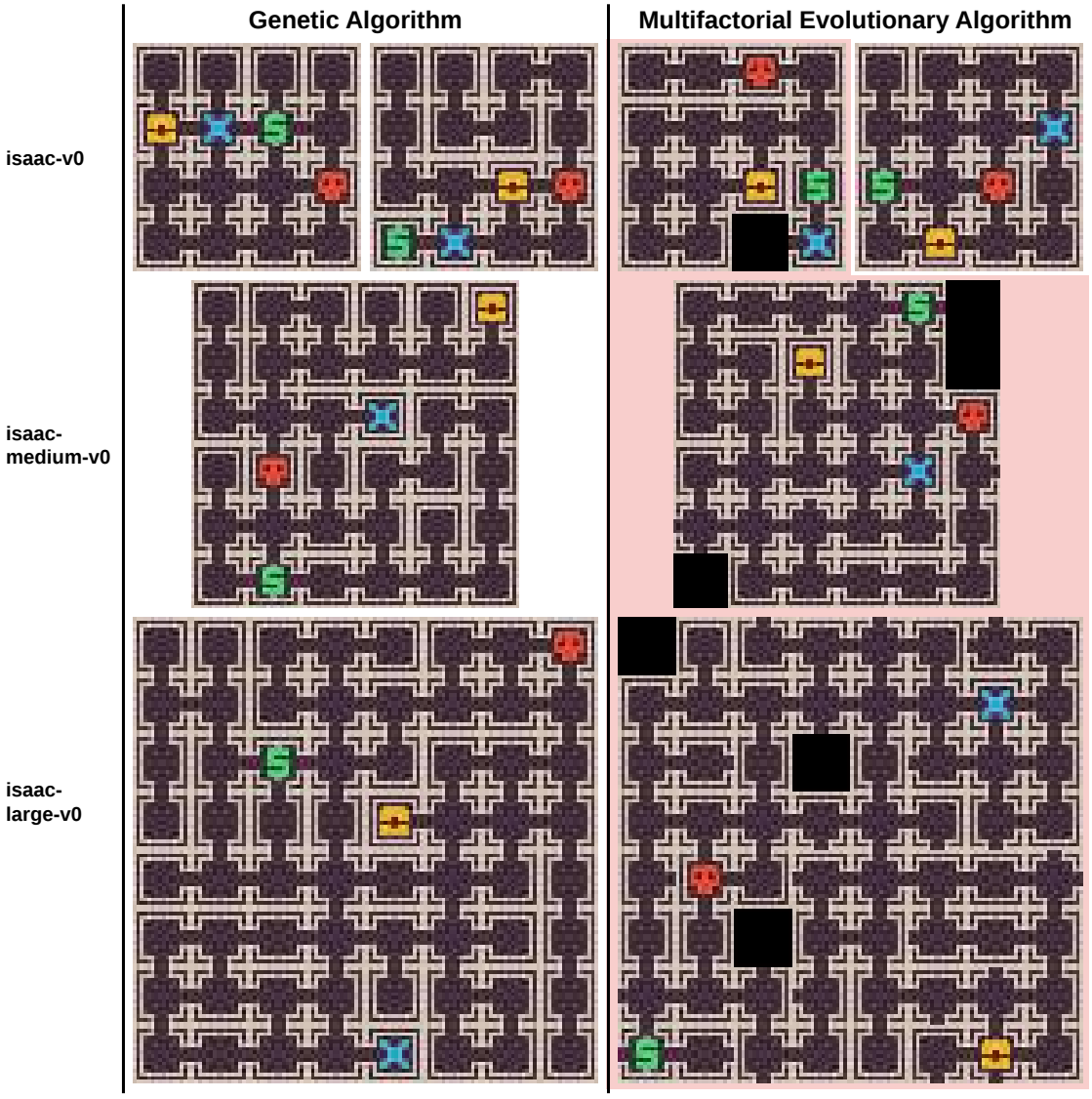

Figure 5.12: Representatives of the best content generated for two compared generators optimizing the Q function for Isaac. Content marked with a red background indicates that it did not meet the Q constraints.

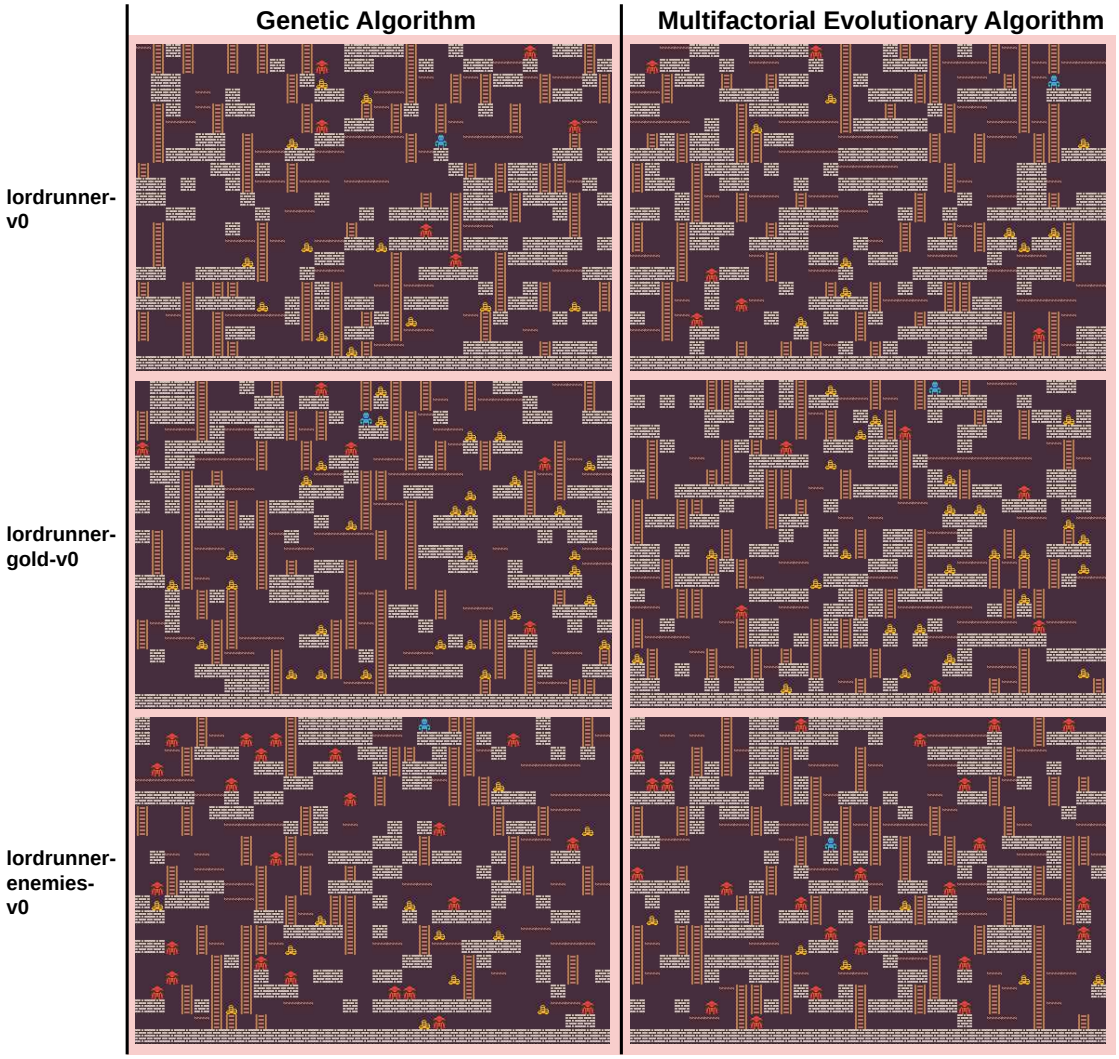

Figure 5.13: Representatives of the best content generated for two compared generators optimizing the Q function for Lode Runner. Content marked with a red background indicates that it did not meet the Q constraints.

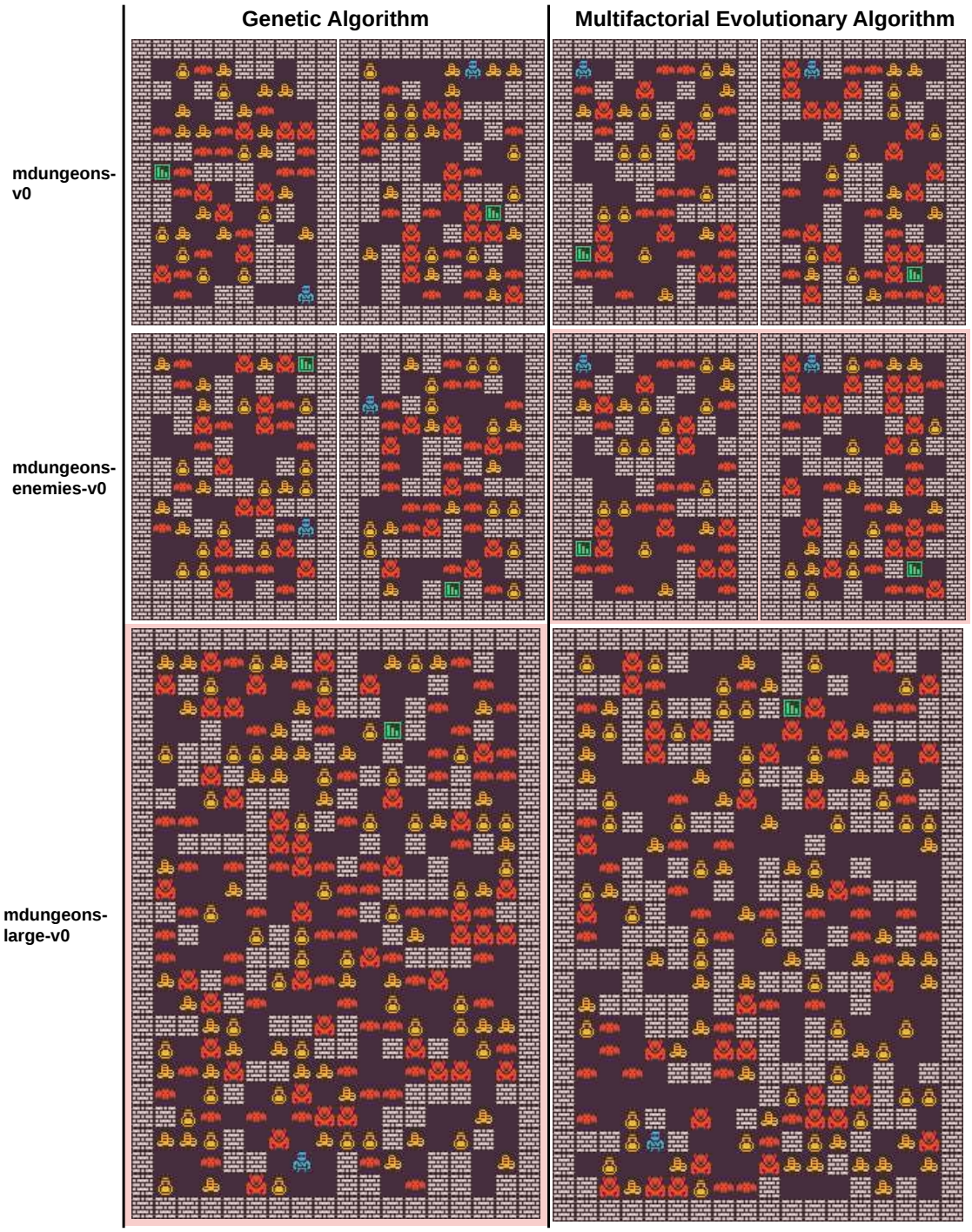

Figure 5.14: Representatives of the best content generated for two compared generators optimizing the Q function for MiniDungeons. Content marked with a red background indicates that it did not meet the Q constraints.

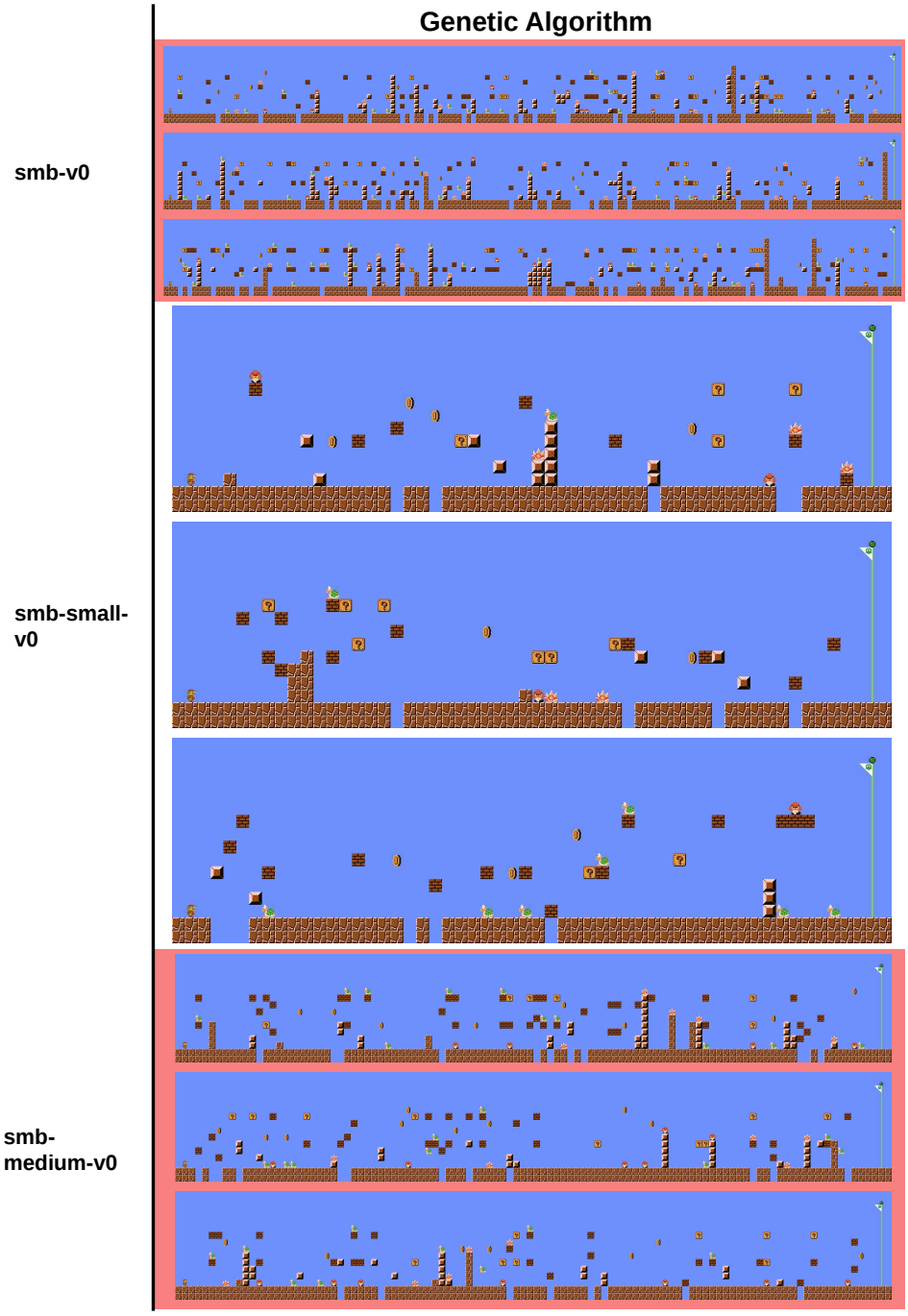

Figure 5.15: Representatives of the best content generated of GA optimizing the Q function for SMB. Content marked with a red background indicates that it did not meet the Q constraints.

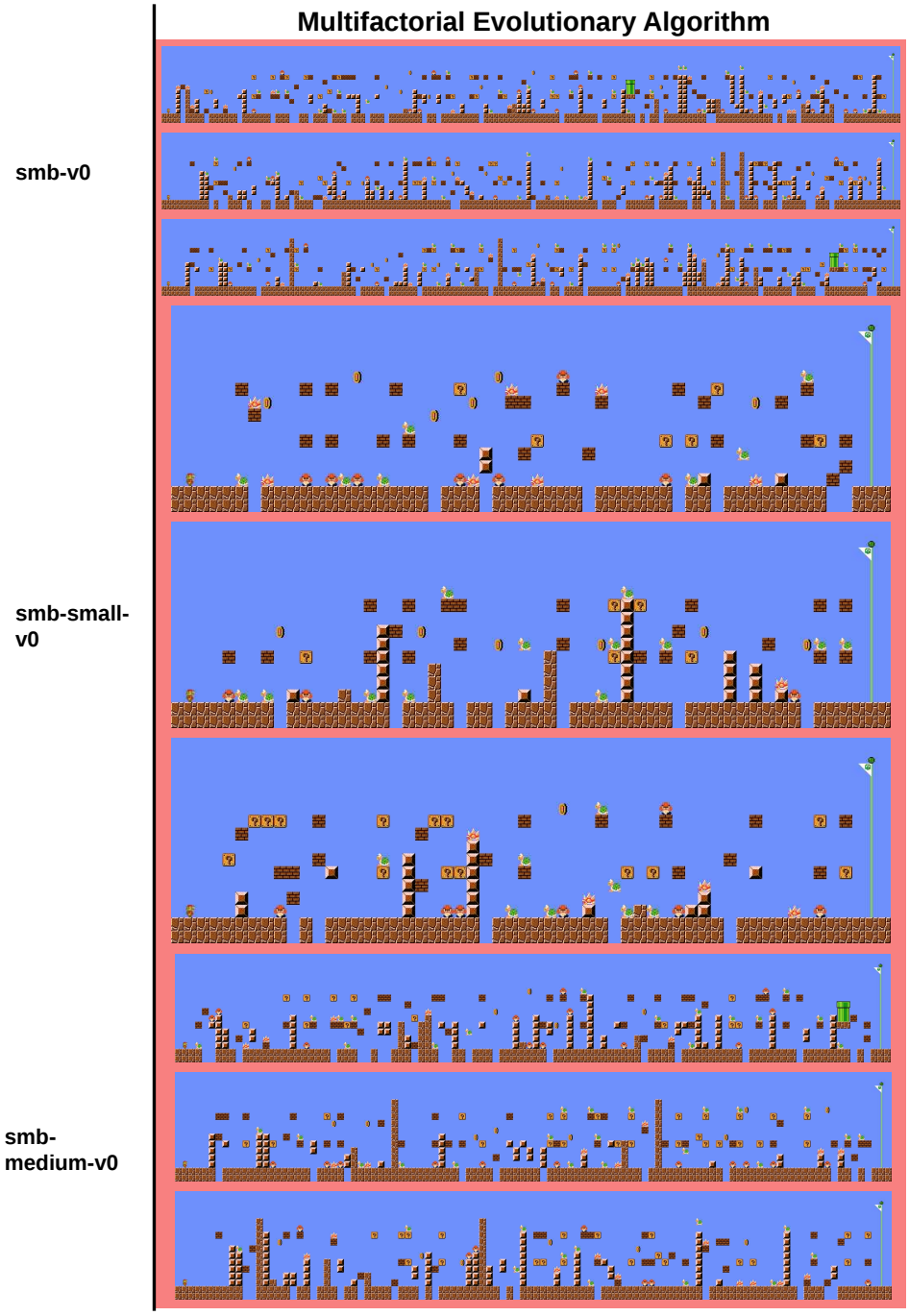

Figure 5.16: Representatives of the best content generated of MFEA optimizing the Q function for SMB. Content marked with a red background indicates that it did not meet the Q constraints.

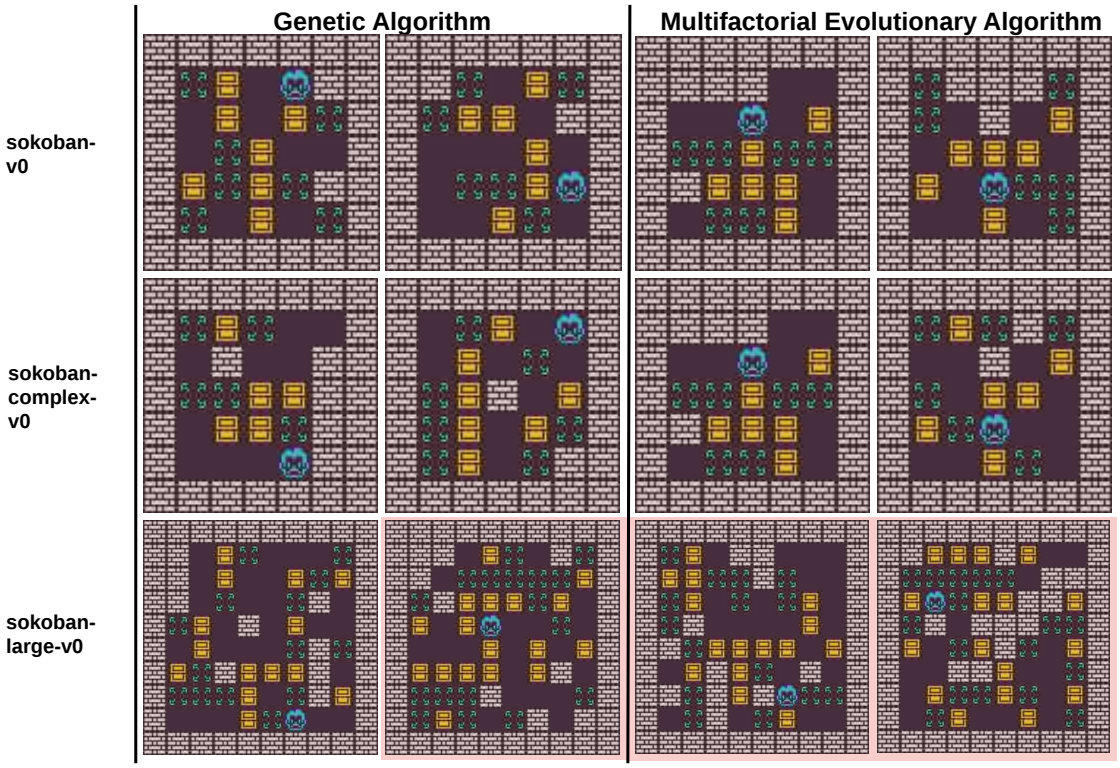

Figure 5.17: Representatives of the best content generated for two compared generators optimizing the Q function for Sokoban. Content marked with a red background indicates that it did not meet the Q constraints.

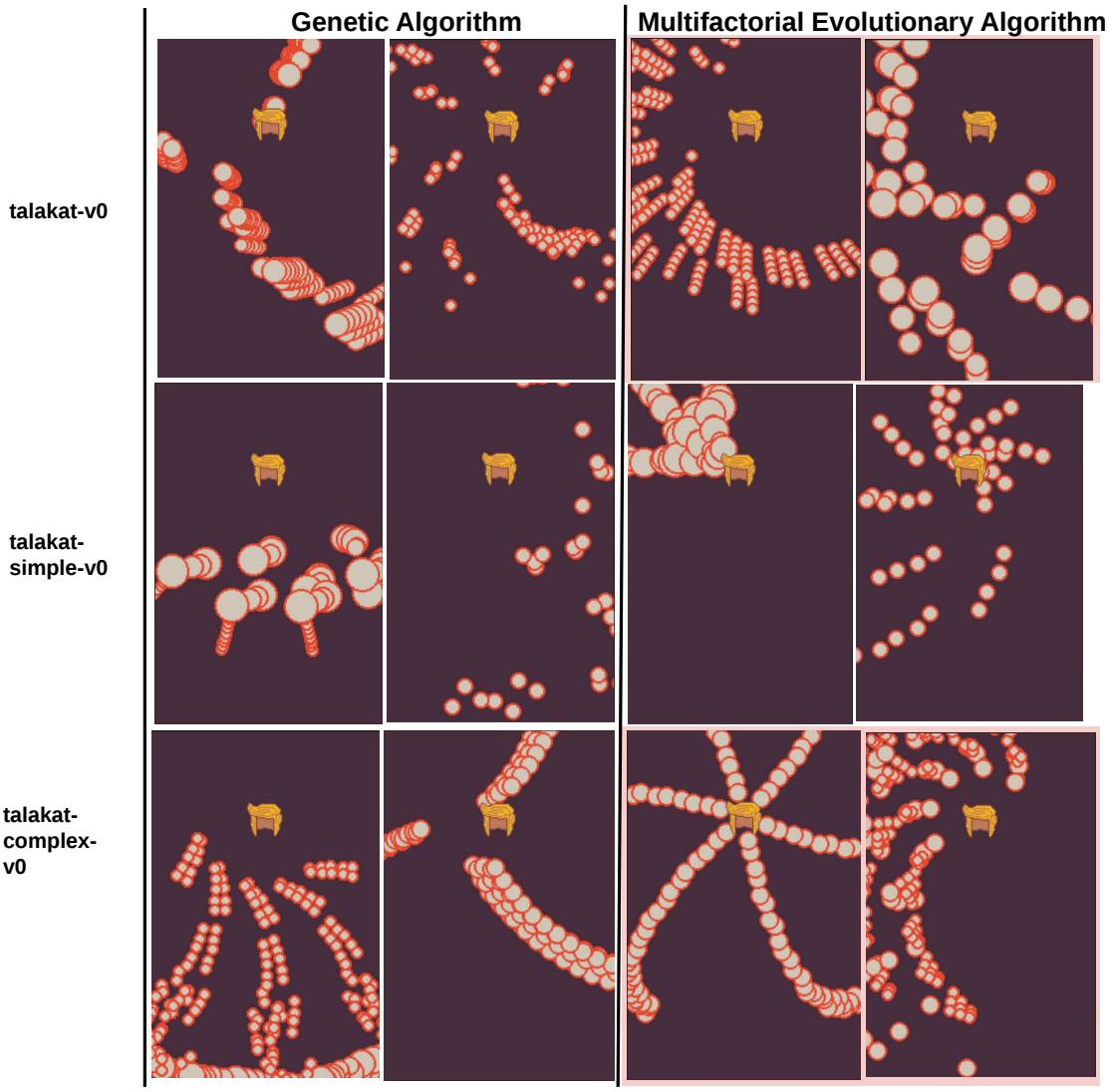

Figure 5.18: Representatives of the best content generated for two compared generators optimizing the Q function for Talakat. Content marked with a red background indicates that it did not meet the Q constraints.

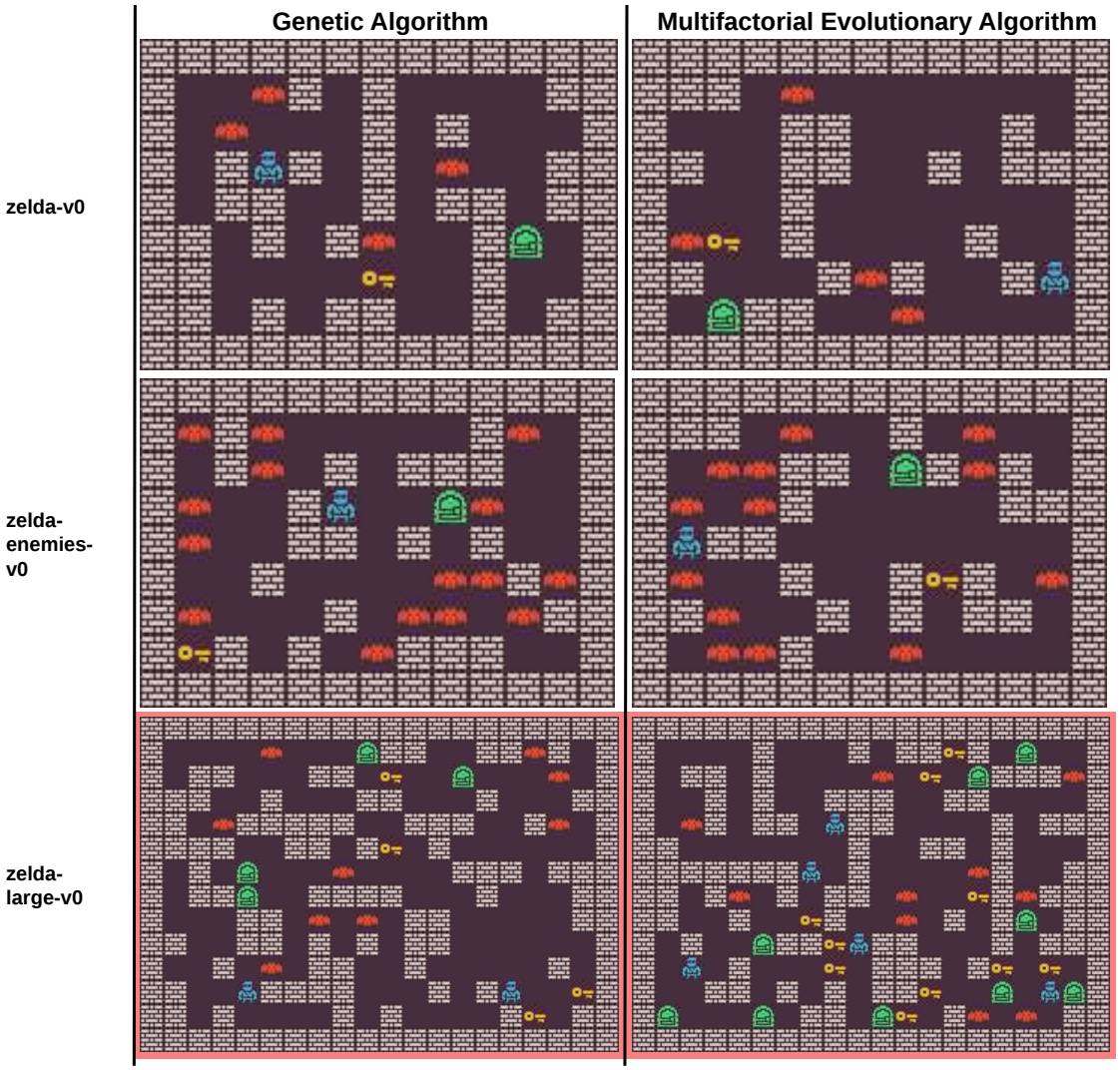

Figure 5.19: Representatives of the best content generated for two compared generators optimizing the Q function for Zelda. Content marked with a red background indicates that it did not meet the Q constraints.

Representatives of generated content from Scenario 2

The best content produced by MFEA, optimizing the Q function under Scenario 2 (See Section 5.2), is illustrated in Figures 5.20 and 5.21.

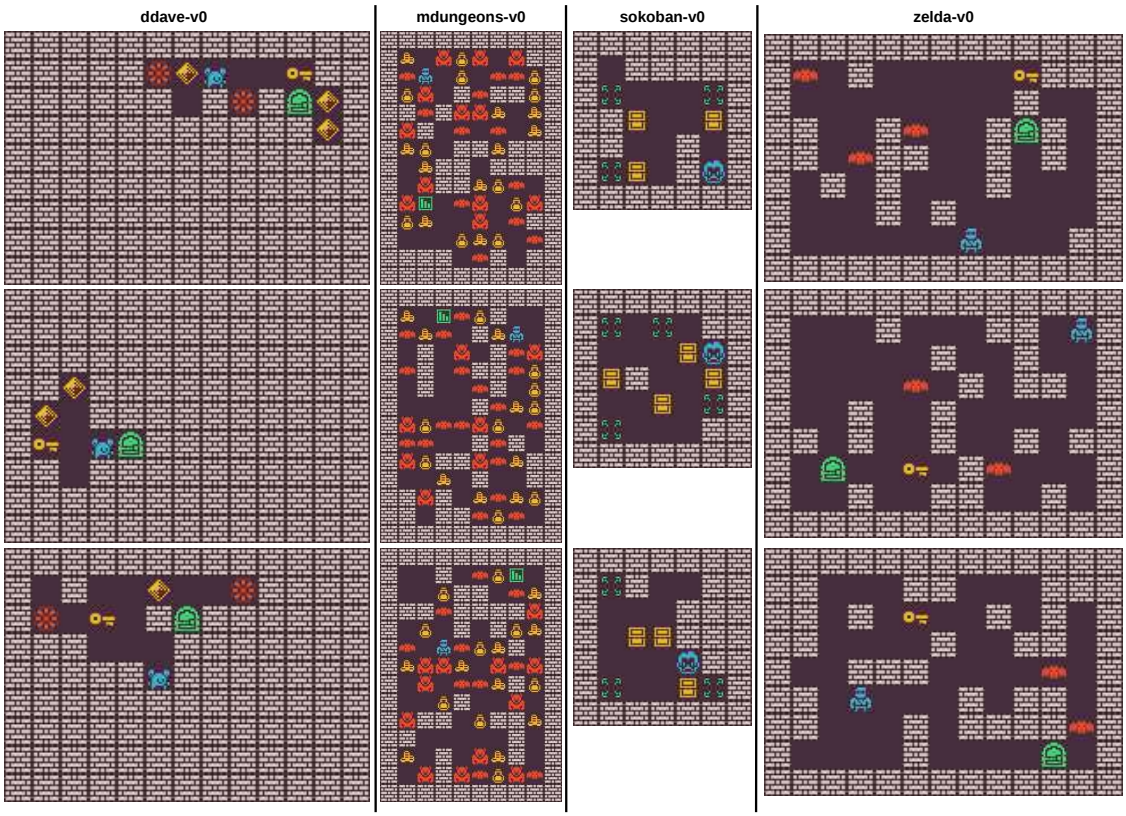

Figure 5.20: The best generated content by MFEA optimizing the Q function for small-sized instances of four different problems simultaneously.

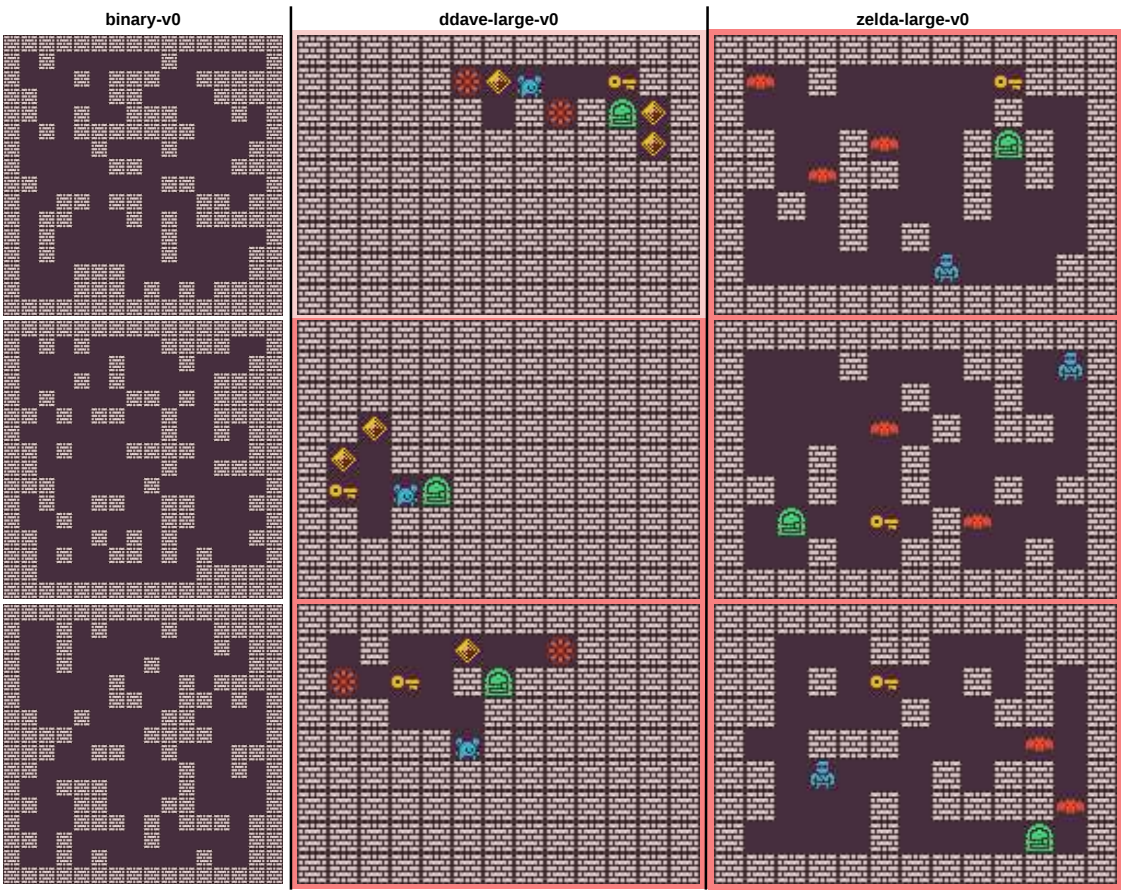

Figure 5.21: The best generated content by MFEA optimizing the Q function for medium-sized instances of three different problems simultaneously. Content marked with a red background indicates that it did not meet the Q constraints.

6 Conclusion

This thesis explored the application of Evolutionary Multitask Optimization (EMTO) within the Procedural Content Generation (PCG) Benchmark. An Multifactorial Evolutionary Algorithm (MFEA)-based framework was proposed to systematically group PCG problems according to the compatibility of their content and control search spaces. To assess the effectiveness of this framework as well as MFEA, experiments were conducted across several multitask scenarios and compared with a single-task Genetic Algorithm (GA) baseline provided in the PCG Benchmark.

This section revisits the research questions presented in the introduction of this thesis. The **RS1** investigates how heterogeneous search spaces of different games can be unified into a common representation. The proposed method for task grouping and Unified Search Space (USS) construction demonstrated that different PCG tasks can be encoded within a shared search space by standardizing their content and control representations, thus enabling the application of multitask optimization. At the same time, task-specific characteristics are preserved through the decoding process, ensuring that solutions remain meaningful.

The **RS2** examined examined whether MFEA can improve previous PCG generators through knowledge transfer, and how to mitigate negative transfer when tasks are incompatible. The proposed framework follows a common technique for minimizing harmful interactions in multitask optimization, which is aligning similar tasks for knowledge transfer. The experimental results show that knowledge trans-

fer is most effective when tasks are closely related, particularly in fully compatible scenarios where instances originate from the same game. Moreover, the number of tasks is another factor that impacts the performance of MFEA, as it increases the complexity of the multitasking environment. This can be seen in the case of *Building*, where the increasing number of tasks leads to lower performance of MFEA. On the other hand, when tasks are less similar, the benefits of transfer become limited and may even lead to negative transfer, which was observed in content-compatible scenarios where tasks originate from different games. These findings highlight the importance of task compatibility and emphasize the need for careful task grouping to maximize the effectiveness of knowledge transfer.

The **RS3** focused on evaluating the benefits and limitations of MFEA in PCG. By comparing multitask optimization with a single-task GA baseline, this study assessed performance across multiple criteria, including solution quality, controllability, and diversity. The experimental results indicate that MFEA is capable of addressing multiple PCG tasks simultaneously while achieving performance comparable to the baseline, even when under significantly reduced evaluation budgets. Additionally, MFEA demonstrated a strong ability to maintain solution diversity, which is a crucial aspect in PCG. In general, the evaluation shows that EMTO approaches provide a promising balance between efficiency and diversity.

6.1 Discussion and Limitations

These findings suggest that EMTO represents a promising research direction for PCG, particularly when the relationships between tasks are carefully considered. By allowing multiple related tasks to be optimized simultaneously, EMTO can potentially improve computational efficiency, which is especially valuable in PCG settings where different content elements share structural or functional similarities.

From a practical perspective, this multitasking approach is particularly rele-

vant for games that contain multiple related content components. Many modern games feature large worlds composed of numerous levels, missions, or environments that are generated under similar design constraints. Likewise, some games present collections of different minigames within a single platform, with each minigame corresponding to a specific generation task. This is especially common in platformers, puzzle, and dungeon-based games, but the same idea can also extend to 3D environments, such as open-world games, where multiple regions or structures must adhere to similar design requirements. This trend is also reflected in the growing popularity of games that integrate multiple gameplay experiences within a single ecosystem. For example, Mario Party [32] has attracted a considerable number of players through its diverse set of minigames, while Roblox [33] serves as a platform hosting a large variety of user-created games and gameplay experiences. In these games, players can transition between different activities while continuously experiencing new content. Additionally, many games combine procedurally generated elements with fixed handcrafted content, where multitask optimization can be used to support the generation of reusable content components while preserving manually designed core content. From a PCG perspective, such environments naturally involve generating a wide variety of related content types, making them well-suited for multitask optimization approaches such as EMTO. Consequently, the ability to generate multiple forms of content within a unified framework could play an important role in supporting large-scale gaming platforms with rich content in the future.

Despite the promising results, this thesis contains several limitations. First, the experiments were conducted exclusively on the PCG Benchmark. Although the benchmark provides a diverse set of PCG tasks, further evaluation on other PCG domains would be necessary to assess the generality of the proposed framework. Second, the approach used in this thesis focused on applying multitask optimization to solve all instances within a single problem simultaneously, without explicitly ad-

dressing the issue of negative transfer. Since multitask environments often contain complex tasks with complex relationships, many EMTO approaches are only applied to a limited number of tasks to minimize the risk of negative transfer. Furthermore, as the number of tasks increases beyond three, the problem setting approaches is referred to as Many-task Optimization (MaTO). Although one experimental bundle in this study exhibit characteristics of such settings, addressing MaTO is beyond the scope of this thesis, as the primary objective is to evaluate the feasibility of applying MFEA to PCG. In that case, MFEA may suffer from increasingly severe negative transfer as the number of tasks grows, particularly when the tasks are not strongly related. This effect was evident in the *Building* problem, where the increasing number of instances reduced the number of evaluations allocated to each instance, which consequently led to a decline in performance. Finally, the experimental setup in this thesis posed a considerable challenge for MFEA, as multiple instances were optimized simultaneously under a reduced evaluation budget compared to the GA baseline. Despite this constraint, MFEA was still able to achieve competitive performance relative to GA.

6.2 Future Work

Future work may explore several directions to further improve the effectiveness of multitask optimization for PCG.

The first direction is on enhancing the methods used in the proposed framework. One approach is developing adaptive task grouping strategies, where tasks are dynamically grouped based on their similarity or transferability during the optimization process. Such technique could help reduce negative transfer and improve the efficiency of knowledge sharing between compatible tasks. Additionally, future studies may investigate the performance of the proposed framework under larger evaluation budgets, which could provide further insights into the scalability of mul-

titask optimization. It would also be beneficial to evaluate alternative multitask evolutionary solvers, which are equipped with adaptive knowledge transfer to further improve the generated content of PCG problems. Finally, extending the proposed framework to explicitly address MaTO settings remains an important direction for future work.

The second direction focuses on practical applications of EMTO in real game development. While this thesis evaluates the framework on the PCG Benchmark, future work should investigate its applicability to practical content generation problems, particularly in 3D open-world games. Since these environments often require multiple interconnected generation tasks, they provide a strong use case for multitask optimization.

References

[1] Mojang Studios. "Minecraft", Microsoft Corporation. [Online]. Available: `https://www.minecraft.net/`

[2] Hello Games. "No Man's Sky".

[3] M. Hendrikx, S. Meijer, J. Van Der Velden, and A. Iosup, "Procedural content generation for games: A survey", *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 9, no. 1, pp. 1–22, 2013.

[4] J. Togelius, E. Kastbjerg, D. Schedl, and G. N. Yannakakis, "What is procedural content generation? mario on the borderline", in *Proceedings of the 2nd international workshop on procedural content generation in games*, 2011, pp. 1–6.

[5] A. Khalifa, R. Gallotta, M. Barthet, A. Liapis, J. Togelius, and G. N. Yannakakis, "The procedural content generation benchmark: An open-source testbed for generative challenges in games", in *Proceedings of the 20th International Conference on the Foundations of Digital Games*, 2025, pp. 1–12.

[6] T. Short and T. Adams, *Procedural generation in game design*. CRC Press, 2017.

[7] A. Gupta, Y.-S. Ong, and L. Feng, "Multifactorial evolution: Toward evolutionary multitasking", *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 3, pp. 343–357, 2015.

[8] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey", *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 172–186, 2011.

[9] E.-G. Talbi, *Metaheuristics: from design to implementation*. John Wiley & Sons, 2009.

[10] A. E. Eiben and J. E. Smith, *Introduction to evolutionary computing*. Springer, 2015.

[11] A. P. Engelbrecht et al., *Computational intelligence: an introduction*. Wiley Online Library, 2007, vol. 2.

[12] A. Brabazon, M. O'Neill, and S. McGarraghy, *Natural computing algorithms*. Springer, 2015, vol. 554.

[13] J. H. Holland, "Genetic algorithms", *Scientific american*, vol. 267, no. 1, pp. 66–73, 1992.

[14] B. H. T. Thanh, T. B. Thang, N. H. Long, et al., "Ensemble multifactorial evolution with biased skill-factor inheritance for many-task optimization", *IEEE Transactions on Evolutionary Computation*, vol. 27, no. 6, pp. 1735–1749, 2022.

[15] Y. Chen, J. Zhong, L. Feng, and J. Zhang, "An adaptive archive-based evolutionary framework for many-task optimization", *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 4, no. 3, pp. 369–384, 2019.

[16] K. K. Bali, Y.-S. Ong, A. Gupta, and P. S. Tan, "Multifactorial evolutionary algorithm with online transfer parameter estimation: Mfea-ii", *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 1, pp. 69–83, 2019.

[17] T. B. Thang, T. C. Dao, N. H. Long, and H. T. T. Binh, "Parameter adaptation in multifactorial evolutionary algorithm for many-task optimization", *Memetic Computing*, vol. 13, no. 4, pp. 433–446, 2021.

[18] J. Tang, Y. Chen, Z. Deng, Y. Xiang, and C. P. Joy, "A group-based approach to improve multifactorial evolutionary algorithm.", in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2018, pp. 3870–3876.

[19] X. Zheng, A. K. Qin, M. Gong, and D. Zhou, "Self-regulated evolutionary multitask optimization", *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 1, pp. 16–28, 2019.

[20] R. Nintendo, "Super mario bros", *Game [NES].(13 September 1985). Nintendo, Kyoto, Japan*, 1985.

[21] S. Dahlskog and J. Togelius, "Patterns as objectives for level generation", in *Design Patterns in Games (DPG), Chania, Crete, Greece (2013)*, 2013.

[22] V. Volz, J. Schrum, J. Liu, S. M. Lucas, A. Smith, and S. Risi, "Evolving mario levels in the latent space of a deep convolutional generative adversarial network", in *Proceedings of the genetic and evolutionary computation conference*, 2018, pp. 221–228.

[23] J. Flimmel, J. Gemrot, and V. Černỳ, "Coevolution of AI and level generators for Super Mario game", in *2021 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2021, pp. 2093–2100.

[24] G. Luo and X. Ma, "A survey on evolutionary multitask optimization", *Journal of Artificial Intelligence and Technology*, vol. 5, pp. 304–313, 2025.

[25] A. Summerville et al., "Procedural content generation via machine learning (pcgml)", *IEEE Transactions on Games*, vol. 10, no. 3, pp. 257–270, 2018.

[26] A. Sarkar, M. Guzdial, S. Snodgrass, A. Summerville, T. Machado, and G. Smith, "Procedural content generation via knowledge transformation (pcg-kt)", *IEEE Transactions on Games*, vol. 16, no. 1, pp. 36–50, 2023.

[27] M. Müller-Brockhausen, A. Khalifa, and M. Preuss, "Scalable procedural content generation via transfer reinforcement learning", in *International Conference on Data Science and Artificial Intelligence*, Springer, 2024, pp. 109–123.

[28] M. Cook. "Tutorial: Generative & possibility space". [Online]. Available: `https://www.possibilityspace.org/tutorial-generative-possibility-space/`

[29] F. Wilcoxon, "Individual comparisons by ranking methods", *Biometrics bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[30] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other", *The annals of mathematical statistics*, pp. 50–60, 1947.

[31] A. Vargha and H. D. Delaney, "A critique and improvement of the CL common language effect size statistics of McGraw and Wong", *Journal of Educational and Behavioral Statistics*, vol. 25, no. 2, pp. 101–132, 2000.

[32] Nintendo. "Mario party: Star rush", Super Mario Wiki. [Online]. Available: `https://www.mariowiki.com/Mario%20Party:%20Star%20Rush`

[33] Roblox Corporation. "Roblox". [Online]. Available: `https://developer.roblox.com/`

Appendix A Additional Generated Levels

This appendix presents additional examples of the generated levels produced during the experiments. A complete collection of generated levels is also available online through the following GitHub repository: `https://github.com/longnh176040/MFEA_PCG`

A.1 Additional levels from Scenario 1

This section shows examples of the best generated content that optimizes the QT and QTD fitness functions in Scenario 1 for each PCG problem within the benchmark. Content marked with a red background indicates that it did not meet the QT or QTD constraints.

Figure A.1: Representatives of the best content generated for two compared generators optimizing the QT and QTD functions for Arcade Rule.

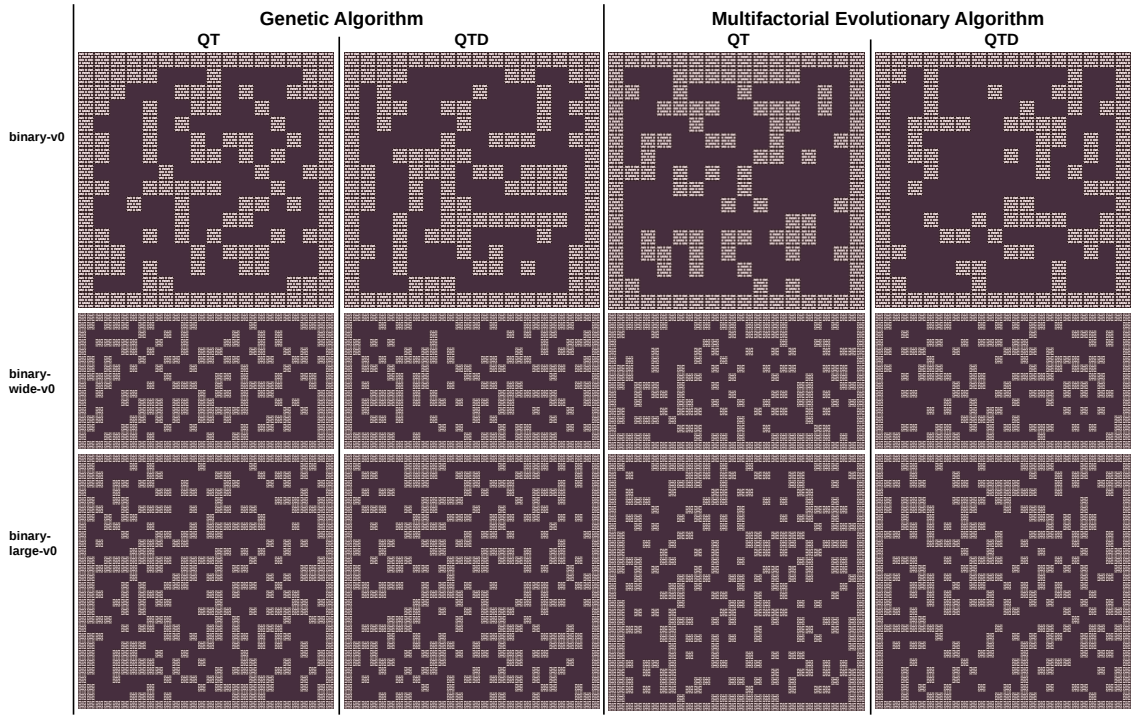

Figure A.2: Representatives of the best content generated for two compared generators optimizing the QT and QTD functions for Binary.

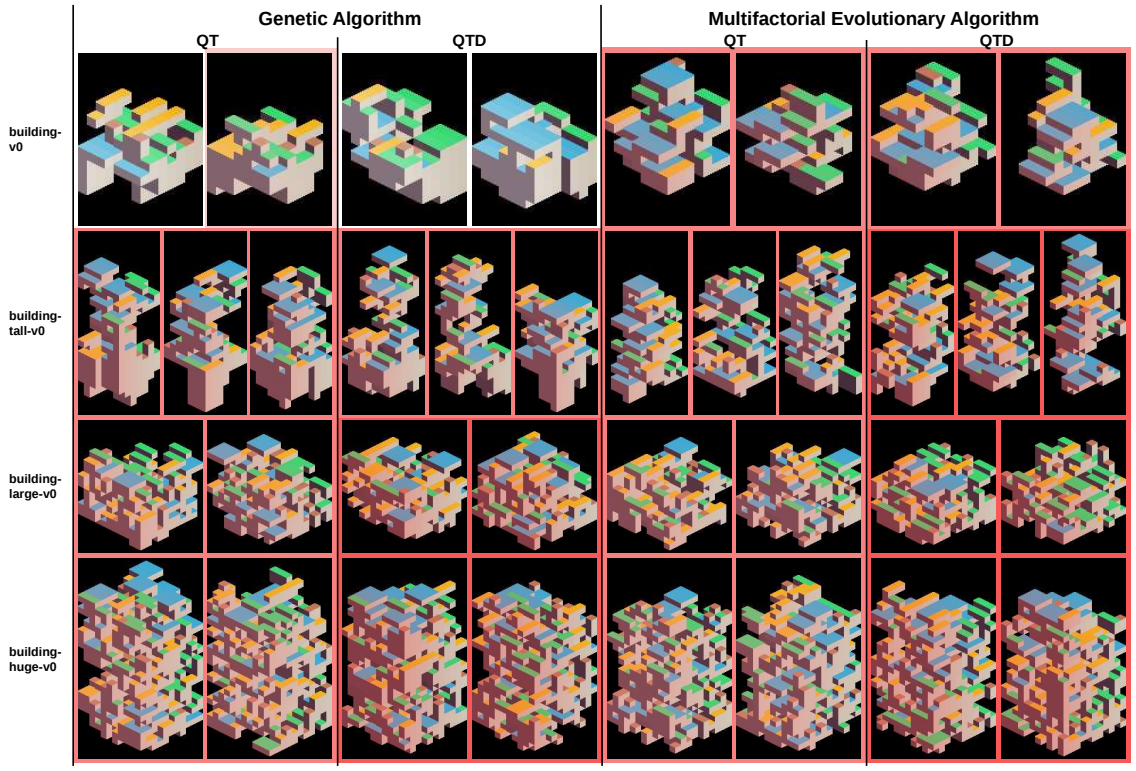

Figure A.3: Representatives of the best content generated for two compared generators optimizing the QT and QTD functions for Building. Content marked with a red background indicates that it did not meet the QT or QTD constraints.

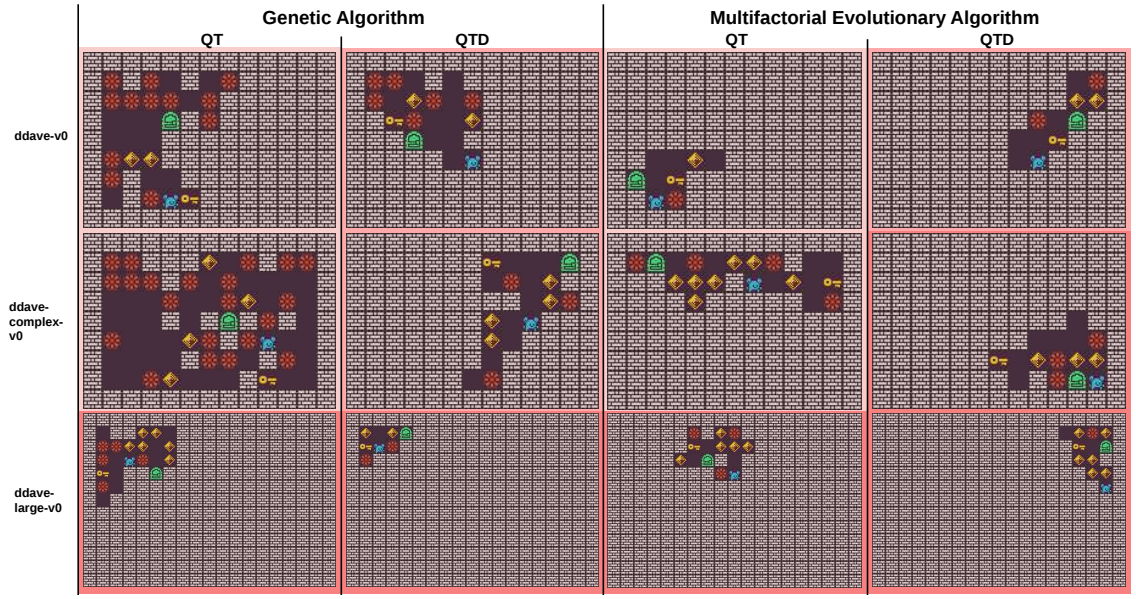

Figure A.4: Representatives of the best content generated for two compared generators optimizing the QT and QTD functions for Dangerous Dave. Content marked with a red background indicates that it did not meet the QT or QTD constraints.

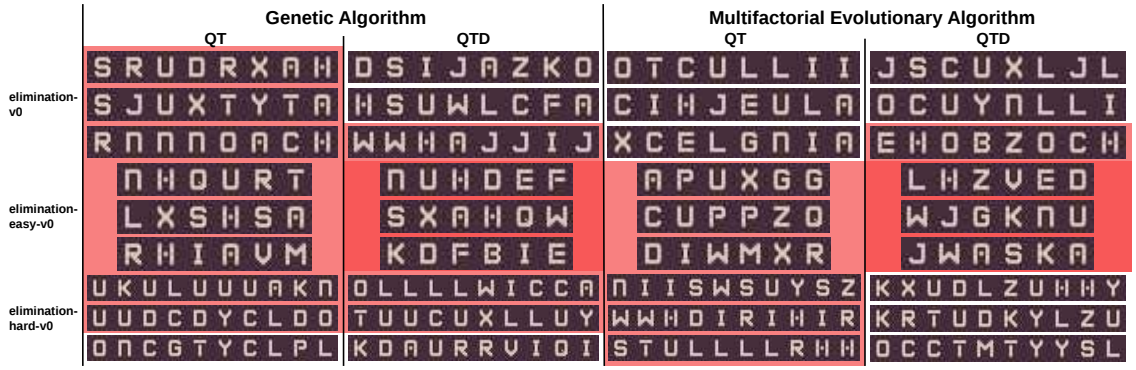

Figure A.5: Representatives of the best content generated for two compared generators optimizing the QT and QTD functions for Elimination. Content marked with a red background indicates that it did not meet the QT or QTD constraints.

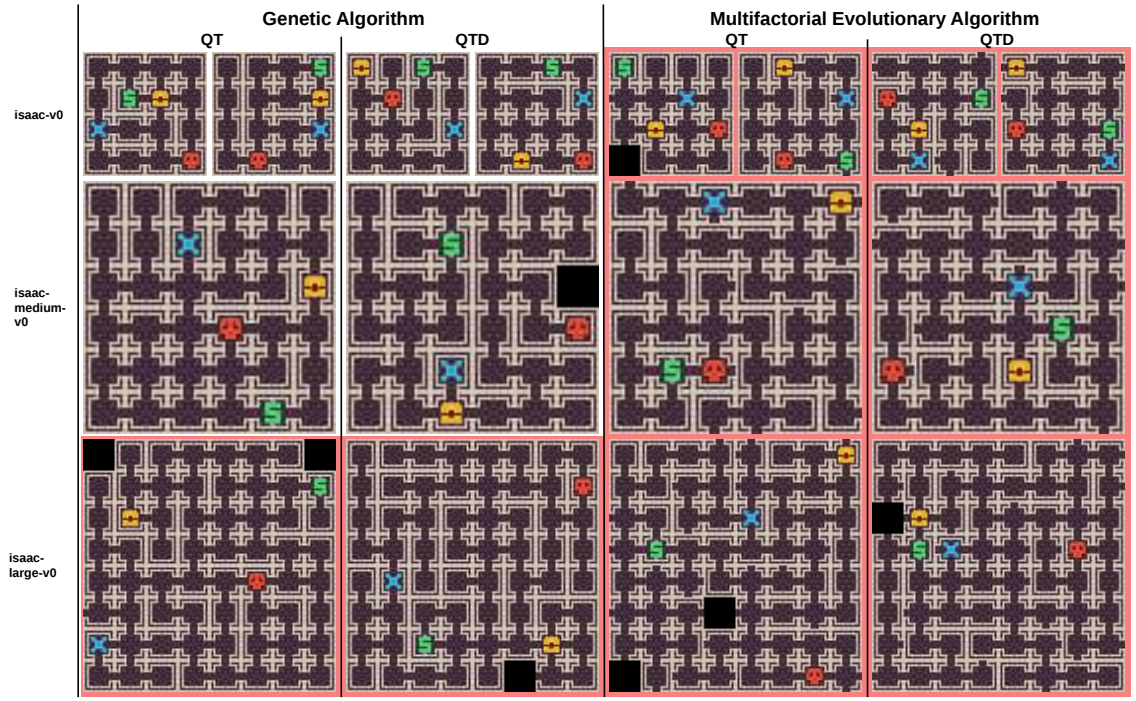

Figure A.6: Representatives of the best content generated for two compared generators optimizing the QT and QTD functions for Isaac. Content marked with a red background indicates that it did not meet the QT or QTD constraints.

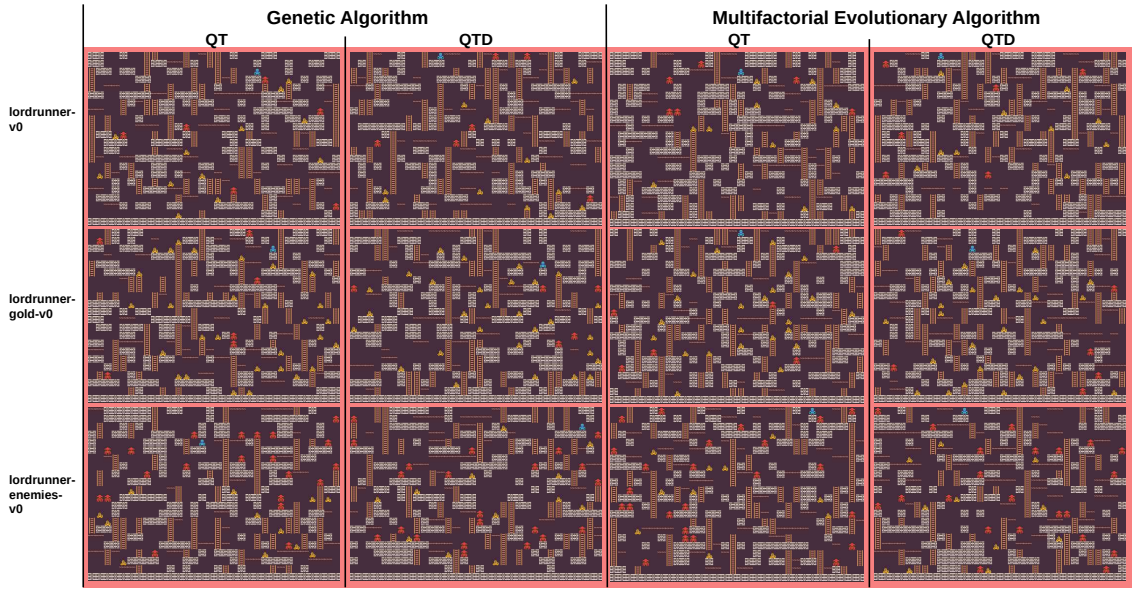

Figure A.7: Representatives of the best content generated for two compared generators optimizing the QT and QTD functions for Lord Runner. Content marked with a red background indicates that it did not meet the QT or QTD constraints.

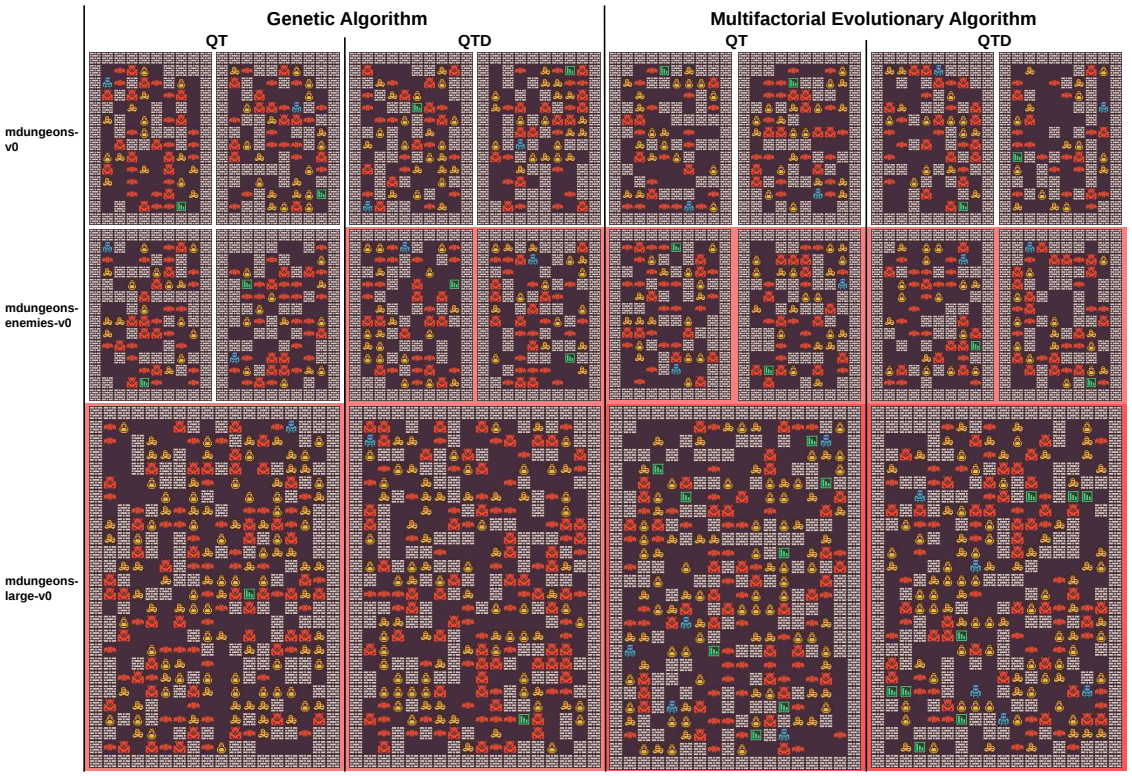

Figure A.8: Representatives of the best content generated for two compared generators optimizing the QT and QTD functions for MiniDungeons. Content marked with a red background indicates that it did not meet the QT or QTD constraints.

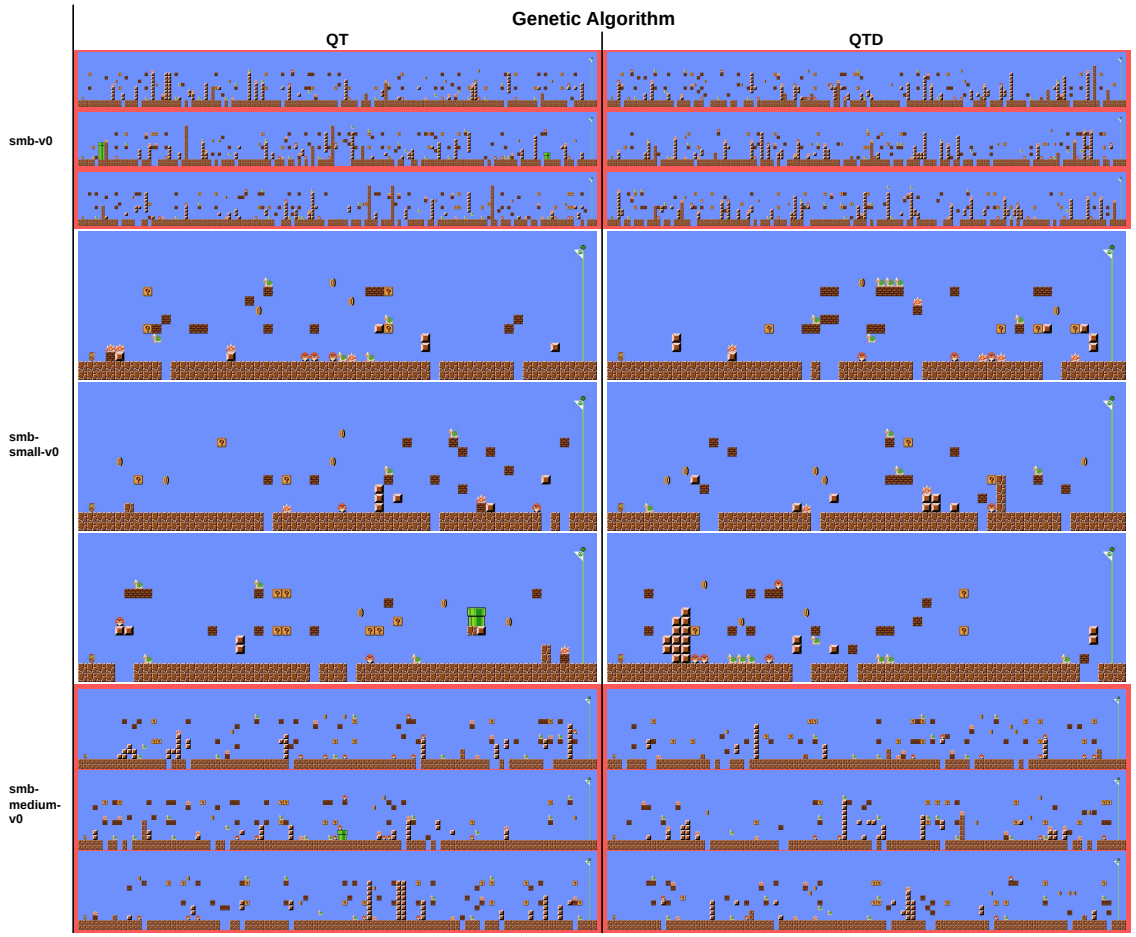

Figure A.9: Representatives of the best content generated for GA optimizing the QT and QTD functions for SMB. Content marked with a red background indicates that it did not meet the QT or QTD constraints.

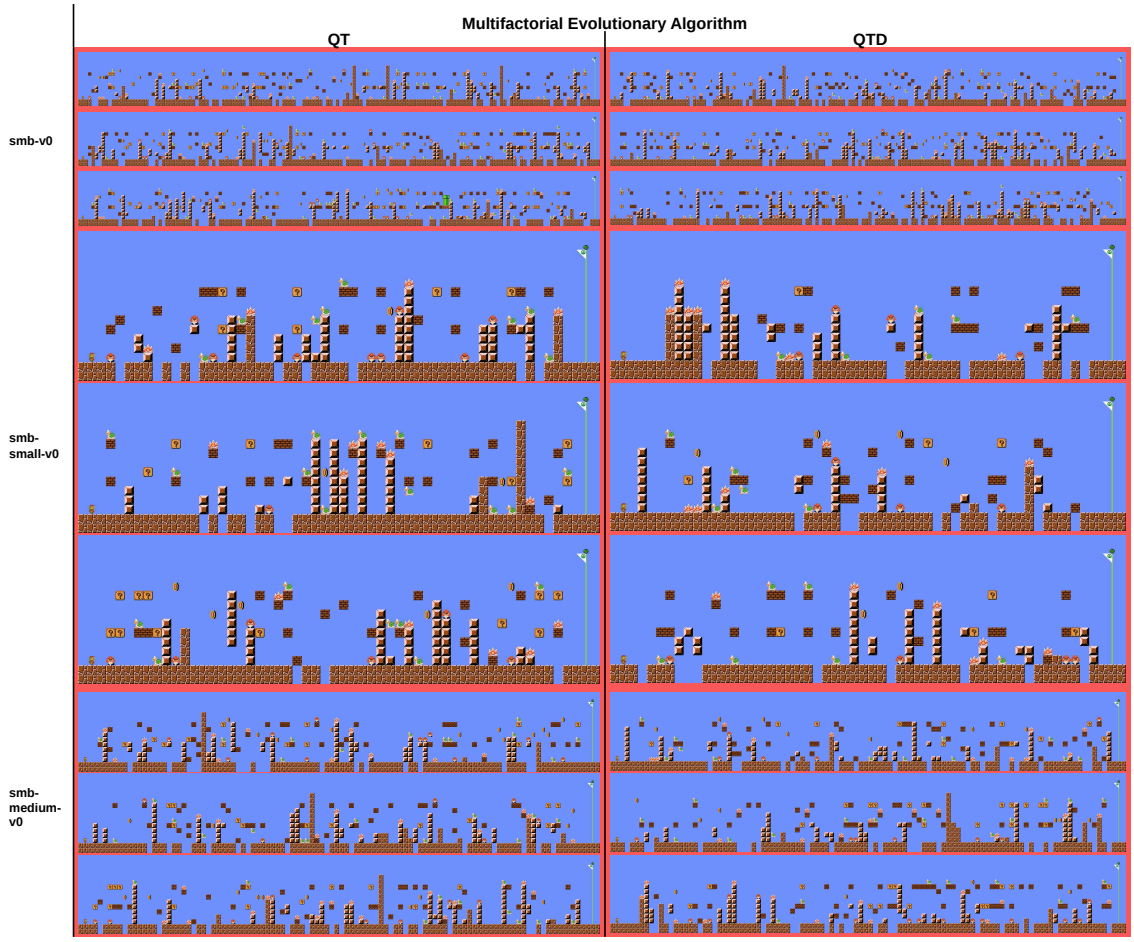

Figure A.10: Representatives of the best content generated for MFEA optimizing the QT and QTD functions for SMB. Content marked with a red background indicates that it did not meet the QT or QTD constraints.

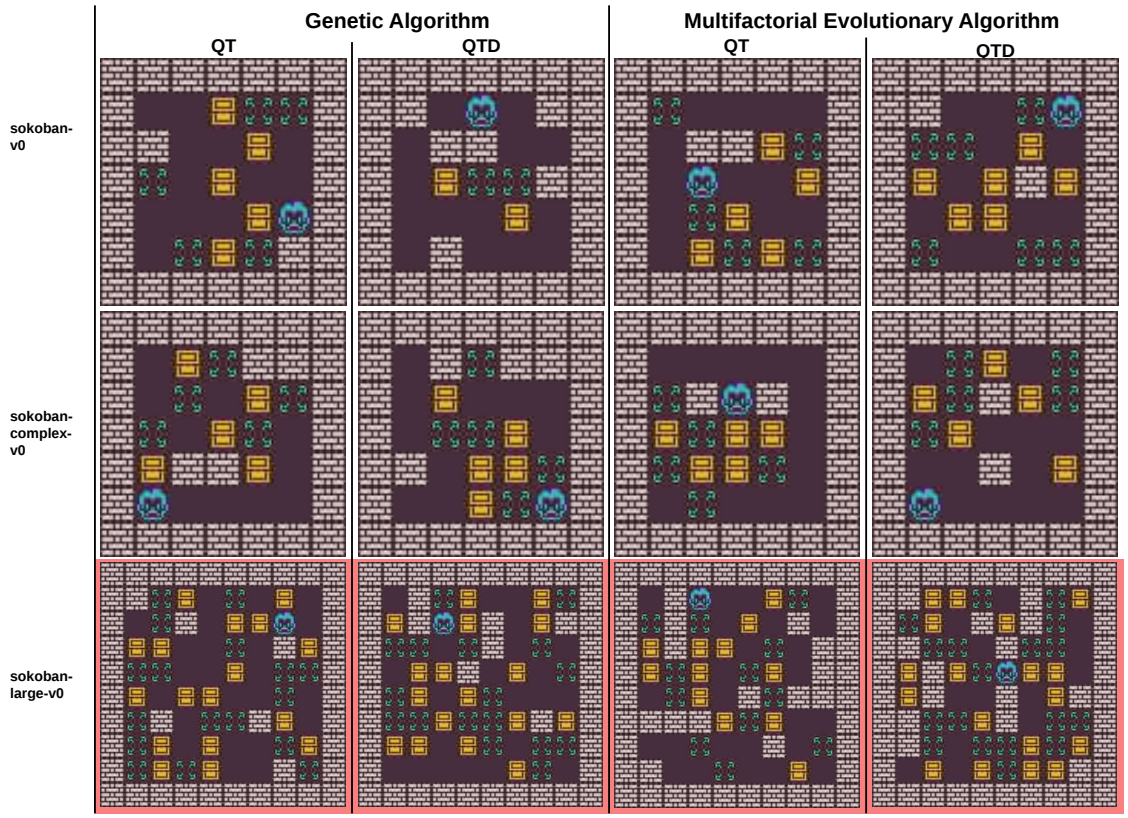

Figure A.11: Representatives of the best content generated for two compared generators optimizing the QT and QTD functions for Sokoban. Content marked with a red background indicates that it did not meet the QT or QTD constraints.

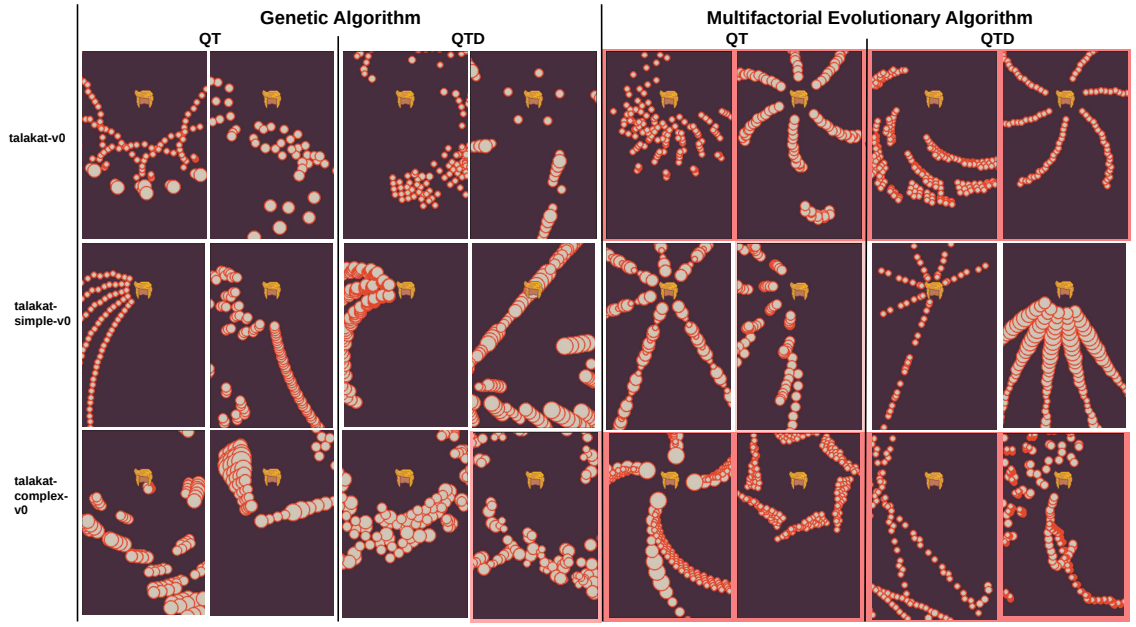

Figure A.12: Representatives of the best content generated for two compared generators optimizing the QT and QTD functions for Talakat. Content marked with a red background indicates that it did not meet the QT or QTD constraints.

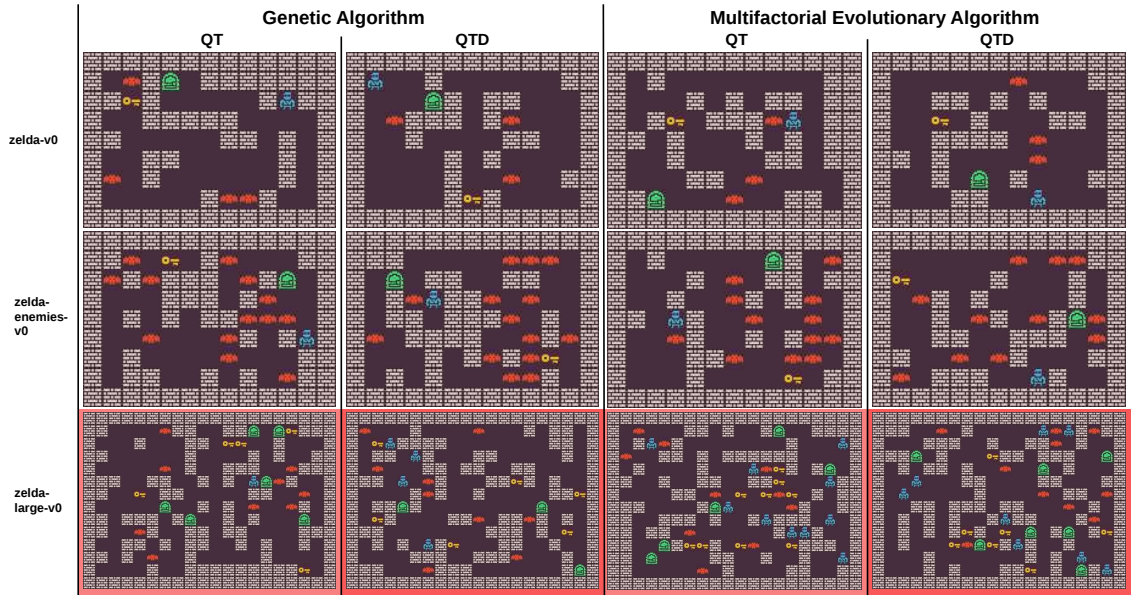

Figure A.13: Representatives of the best content generated for two compared generators optimizing the QT and QTD functions for Zelda. Content marked with a red background indicates that it did not meet the QT or QTD constraints.

A.2 Additional levels from Scenario 2

This section shows examples of the best generated content that optimizes the QT and QTD fitness functions in Scenario 2 using MFEA. Content marked with a red background indicates that it did not meet the QT or QTD constraints.

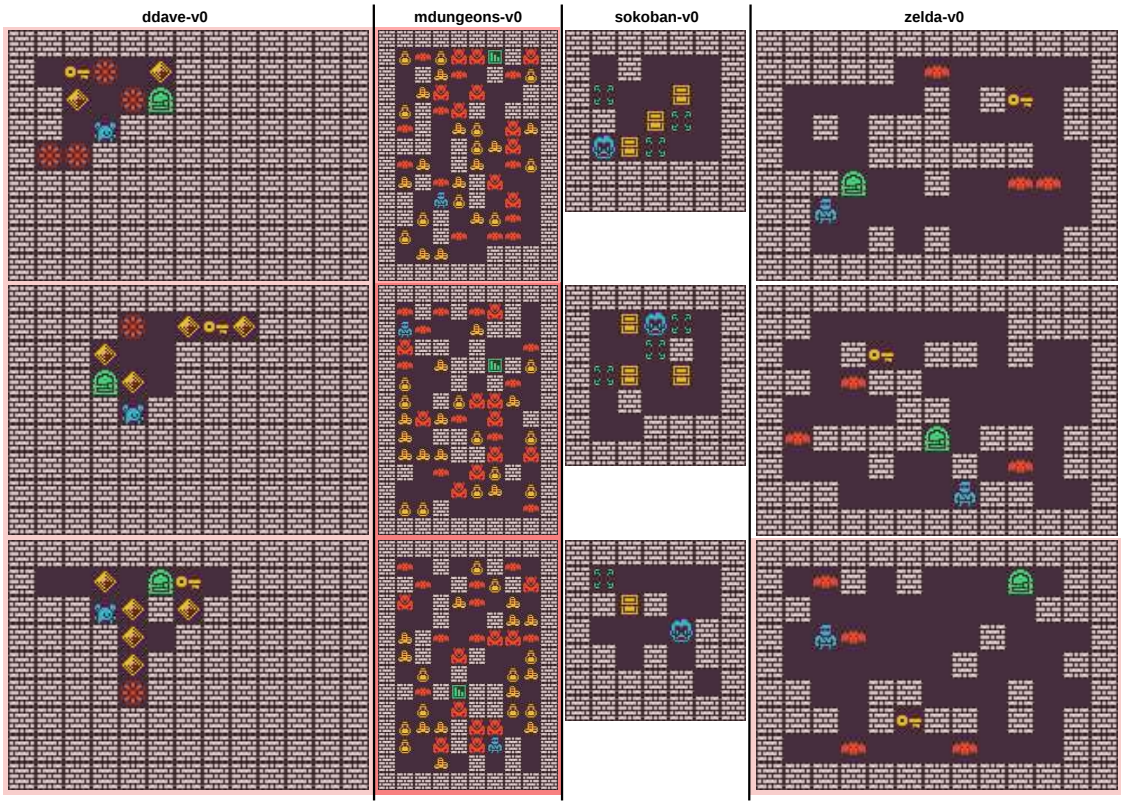

Figure A.14: The best generated content by MFEA optimizing the QT function for small-sized instances of four different problems simultaneously. Content marked with a red background indicates that it did not meet the QTconstraints.

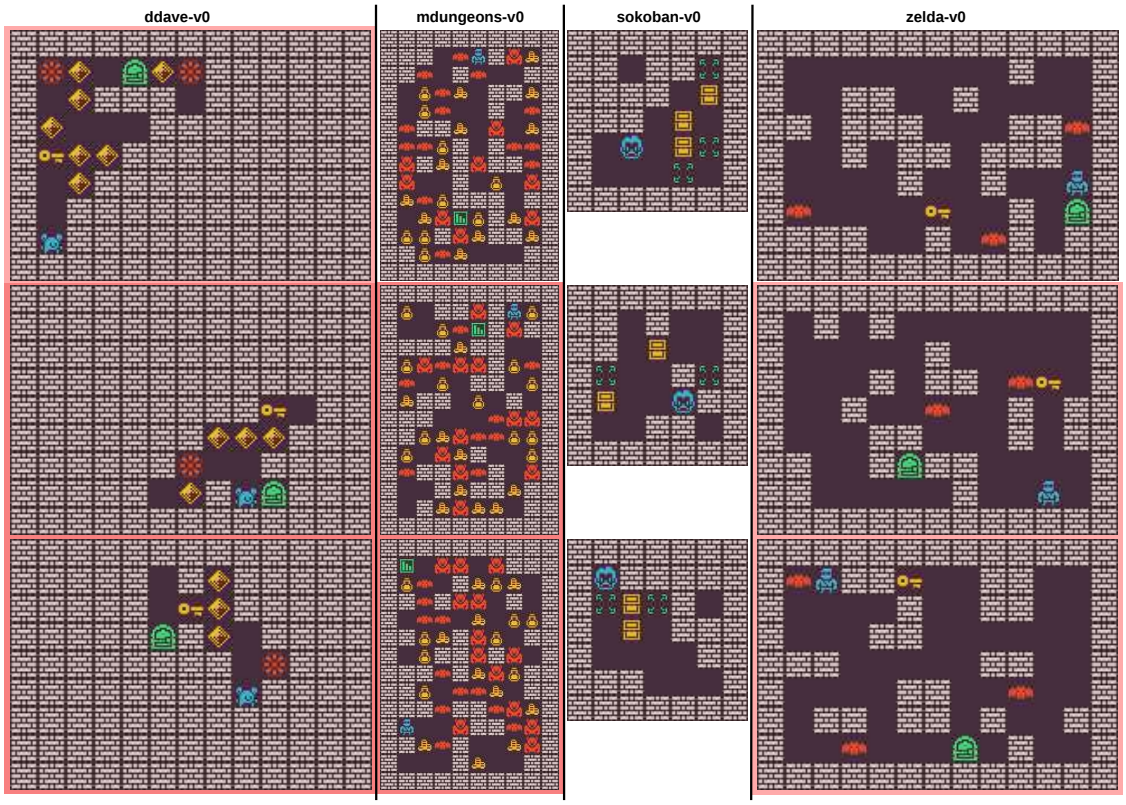

Figure A.15: The best generated content by MFEA optimizing the QTD function for small-sized instances of four different problems simultaneously. Content marked with a red background indicates that it did not meet the QTDconstraints.

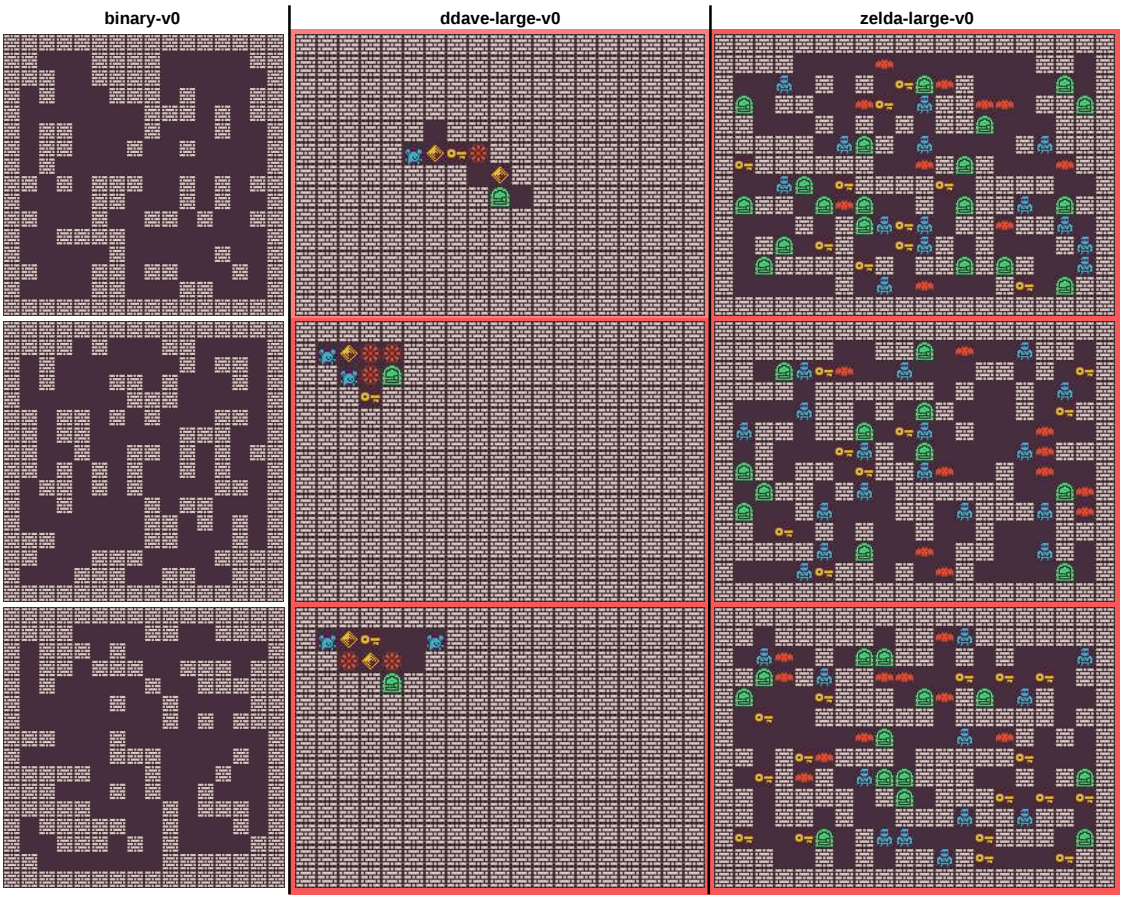

Figure A.16: The best generated content by MFEA optimizing the QT function for medium-sized instances of four different problems simultaneously. Content marked with a red background indicates that it did not meet the QTconstraints.

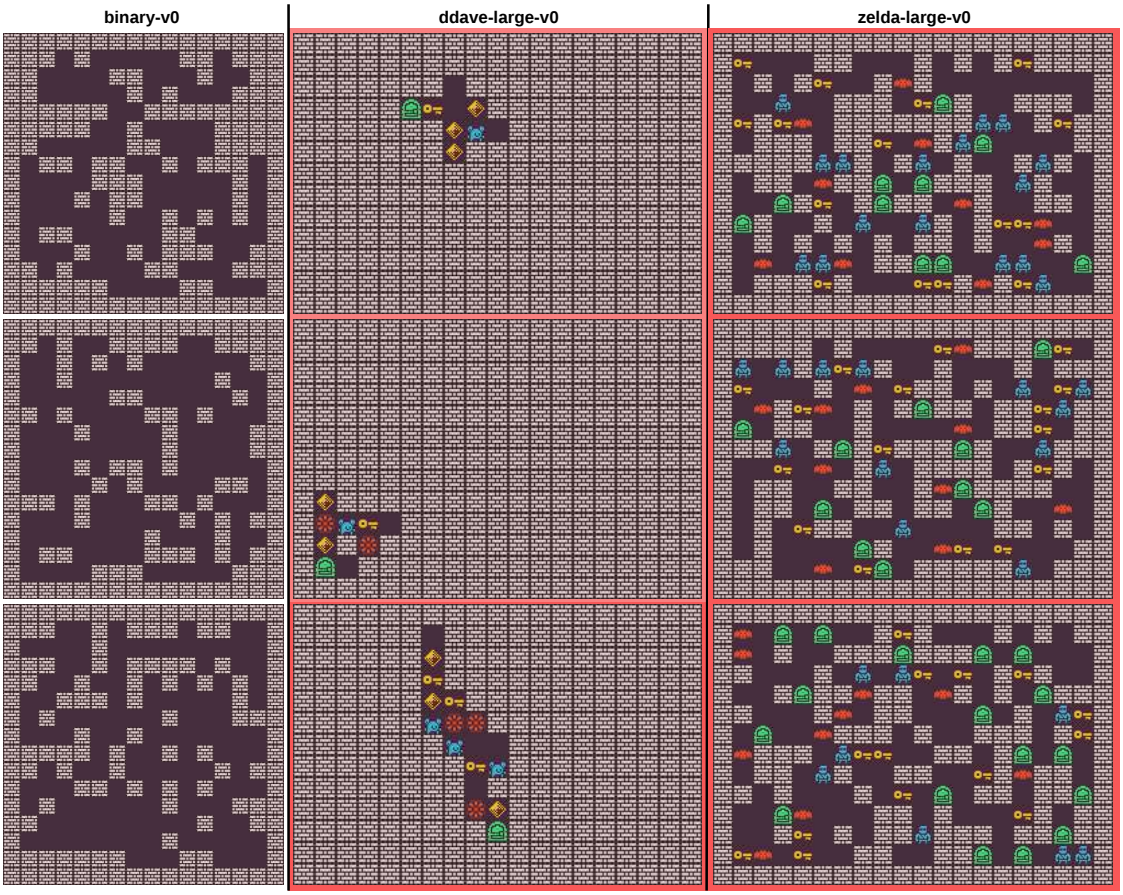

Figure A.17: The best generated content by MFEA optimizing the QTD function for medium-sized instances of four different problems simultaneously. Content marked with a red background indicates that it did not meet the QTDconstraints.

Appendix B Use of AI tools

During the writing of this thesis, free versions of OpenAI ChatGPT and Grammarly were used to support writing and language refinement. These tools were primarily employed to improve grammar, enhance clarity, and strengthen the academic tone of the text. All suggestions from AI were carefully reviewed, revised, and refined to ensure that the final content accurately reflected my own understanding, interpretation, and writing style.

In addition, the free version of ChatGPT was used to support brainstorming during the research process. It was particularly helpful for explaining complex calculations, equations, and mathematical reasoning, which improved my understanding of these concepts and supported their clearer presentation in the thesis. AI tools were used solely as supportive aids and did not replace independent research, critical analysis, or the original contributions of this work.