Exhaustive Backtracking in Hierarchical Wave Function Collapse for Procedural Music Generation

Pál Patrik Varga[1]
Supervisor: Dr.ir. Rafael Bidarra[1]
[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2024

Name of the student: Pál Patrik Varga
Final project course: CSE3000 Research Project
Thesis committee: Dr.ir. Rafael Bidarra, Dr. Joana de Pinho Gonçalves

Exhaustive Backtracking in Hierarchical Wave Function Collapse for Procedural Music Generation

Pál Patrik Varga
VargaPalPatrik@student.tudelft.nl
Delft University of Technology
Delft, The Netherlands

ABSTRACT

Wave Function Collapse (WFC) is among the most well-known and beloved procedural content generation algorithms. WFC solvers can be understood as a specific type of constraint solver, and as such, backtracking is an integral part of exploring the space of possible solutions. Recent advancements were inspired by this algorithm to model procedural music generation, using a hierarchy of many canvases, each with its own semantic domain within music composition (sections, chords, melody). In the model, due to the structural dependencies between canvases, constraints that involve cells from different canvases make exhaustive backtracking a challenge. Existing backtracking methods only consider a single set of variables, but if a specific value on one canvas can affect the entire structure of another canvas, we need more elaborate ways to deal with conflicts.

This paper presents the basic principles that backtracking models for this hierarchy should abide by, and additional guidelines for evaluating them. Two approaches are proposed, and their runtime efficiencies are compared. Depth-first traversal of the canvas hierarchy results in significantly shorter runtimes than its breadth-first counterpart.

CCS CONCEPTS

• **Applied computing** → **Sound and music computing**; • **Theory of computation** → **Constraint and logic programming**.

KEYWORDS

wave function collapse, procedural music generation, constraint programming, backtracking

ACM Reference Format:
Pál Patrik Varga. 2024. Exhaustive Backtracking in Hierarchical Wave Function Collapse for Procedural Music Generation. In *Proceedings of Research Project (CSE3000).* ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Procedural content generation (PCG) algorithms are crucial in many creative industries, such as gaming and digital arts. Wave Function Collapse (WFC) stands out for its ability to generate complex and varied content based on set patterns. Initially popular for creating visual content, WFC is now also being used to automate music composition, handling different musical elements like sections, chords, and melodies [12].

At its core, WFC is a type of constraint solver that manages how different elements relate to each other to ensure the output is coherent. In music, this means the algorithm must satisfy multiple constraints to produce acceptable musical results.

The use of mixed-initiative capabilities in WFC is particularly important because it allows human composers to interact with the algorithm. This interaction enables composers to influence the output by adding their input directly, shaping the music as the algorithm generates it.

However, the complexity of managing many constraints can lead to conflicts. This is where backtracking is essential. Backtracking is a technique used to explore and find solutions by reverting decisions to resolve conflicts. It needs to iterate through possible solutions until it finds one that complies with every constraint. It should only detect a conflict (stemming from the combination of the specific constraints) if truly no solution is possible.

Implementing backtracking in a hierarchical structure, such as the one used in WFC for music generation, presents a challenge. The primary difficulty lies in the interconnectedness of its levels. Each level, representing different musical elements, depends on the others in ways that complicate the backtracking process. For instance, a change in the chord level might necessitate alterations in the melody level. This dependency means that identifying and correcting a single conflicting element often requires adjustments across multiple levels. Efficiently managing these multi-level adjustments without excessive backtracking is key to maintaining both the algorithm's performance and the output's compliance to the constraints.

Our contribution to the state of the art is presenting a model that enables backtracking in this hierarchical structure.

2 RELATED WORK

2.1 Backtracking

Backtracking and constraint programming are well-established concepts in computer science, widely recognized for their applications in artificial intelligence, combinatorial optimization, and operations research. These methods have been thoroughly documented and popularized through the work of several key figures in the field. Notably, Donald Knuth provides an extensive overview of combinatorial algorithms including detailed discussions on backtracking algorithms [8]. Similarly, Peter van Beek's contributions [10] offer comprehensive insights into constraint satisfaction problems and

CSE3000, June 22, 2024, Delft

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/XXXXXXX.XXXXXXX

the associated backtracking search algorithms. These contributions have significantly shaped the understanding and application of these concepts in various domains.

It is commonly understood that solvers for constraint satisfaction problems should not detect a conflict if a solution is possible. Backtracking is the most widespread method for ensuring this [2].

2.2 Wave Function Collapse and Hierarchical WFC

Wave Function Collapse is a PCG algorithm inspired by a concept from quantum mechanics, where a wave function – representing a superposition of multiple different states – collapses into a definite state [4]. In the context of PCG, WFC uses this analogy to generate structured and diverse content. The algorithm operates by considering a superposition of possible states (or values) for each cell in a grid (which can represent e.g. pixels, tiles, or voxels). Iteratively, a cell gets collapsed to a single state, leading to a propagation of cell state changes based on rules derived from an input sample. Ultimately, if no rule conflict takes place, the algorithm outputs coherent and visually appealing content, that somehow resembles the input sample.

The original WFC algorithm, as conceptualized by Maxim Gumin [4], has inspired considerable PCG research. Karth and Smith [6] have examined the algorithm and the power that lies within it, supported by multiple in-the-wild use cases of WFC. Kim et al. [7] have demonstrated WFC's capabilities of solving constraint problems in non-grid-shaped settings.

Much of the work on WFC has addressed several of its original limitations, and extended it in a variety of directions, particularly regarding its accessibility to non-technical users. Recent advancements have focused on introducing mixed-initiative elements into WFC [9]. These enhancements allow for better user interaction and more control, enabling users to directly influence the generation process. A mixed-initiative approach not only makes WFC more intuitive for users but also expands its utility in creative domains such as game level design and graphic arts. By allowing for interactive steering of the algorithm, manual editing, and manipulation of generation parameters, these developments have opened new avenues for creative expression and design flexibility in procedural content generation.

As detailed by Karth and Smith [5], WFC not only exemplifies the successful application of complex concepts like constraint solving and machine learning in PCG, but also highlights the significant potential of such algorithms beyond traditional academic research. This is evidenced by its diverse adaptations and extensions in various fields, ranging from game development to creative design, demonstrating a versatile and practical approach to algorithmic problem-solving and content creation.

The integration of a hierarchical tileset into the WFC algorithm has been shown to yield significant improvements in procedural content generation. Alaka and Bidarra [1] introduced Hierarchical WFC (HWFC), an extension to WFC featuring *meta-tiles*, i.e. intermediate elements (e.g. forest) which can stand for a group of possible tiles with a specific meaning (e.g. bush, pine, oak, grass, etc.). Their work shows how the introduction of a hierarchy of tiles can improve interactivity and control in the design process.

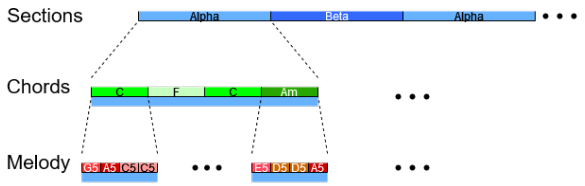

Figure 1: Basic HWFC model for procedural music generation, proposed by Varga and Bidarra [12].

Similarly, Beukman et al. [3] reinforced this, proposing a different hierarchical approach to enhance the diversity and control of level generation.

2.3 Music generation inspired by WFC

Varga and Bidarra [12] introduced a model for procedural music generation, based on Wave Function Collapse. Instead of a single canvas, this model features three levels (sections, chords and melody), with each cell on the upper levels being associated with an entire canvas of cells on the level below (see Figure 1). The level below is formed from the concatenation of all of these canvases, with each one belonging to a single cell on a higher level, but containing multiple cells. For example, after filling a canvas of chords, a melody canvas will be generated for each of the chord cells, and the final melody will be the results of all of the melody canvases combined.

Constraints can be specified by the composer, either by directly filtering the domain, or by drawing up certain relations that neighboring cells should obey. Some parameters of some constraints may depend on values chosen for the cells above in the hierarchy. So-called prototypes can also be defined, and when these are chosen for a value of a given cell, the canvases underneath this cell in the hierarchy may get new constraints, the ones specified with the prototype.

This model does not use any backtracking, and to our knowledge, no model exists for backtracking over such hierarchically arranged variables, in any constraint solver setting. The goal of this paper is to present such a backtracking model for the music generation model, in a way that the takeaways are more widely applicable in constraint solvers with structural hierarchy between their variables.

3 EVALUATION CRITERIA FOR VARIOUS BACKTRACKING ALGORITHMS

We identify the following aspects of backtracking algorithms as a basis for evaluating and comparing them:

- **Completeness:** By far most importantly, the algorithm should consider the entire solution space. In other words, the algorithm should only be unsuccessful if truly no solution is possible with the specified constraints.
- **Runtime efficiency:** An important characteristic of a solver in general is its runtime. Backtracking is a significant portion of this, maybe some backtracking method we find will be more efficient than others.
- **Soft constraint satisfaction:** The model allows for specifying soft constraints. These do not remove anything from the

domains, they just mark some values on some cells as more or less desirable. Maybe different methods of backtracking satisfy these constraints to varying degrees.
- **Conflict localization:** As a composer using an editor based on this model, after running into a conflict, a very useful piece of information would be the location and the constraints at the source of this conflict.

Unfortunately, properly evaluating each of these properties for our proposed algorithms seemed too ambitious for the scope of this study. In the following section, we go to great lengths to ensure that all of the algorithms we propose completely exhaust the domains of the variables. Subsection 5.2 details our experiment for comparing the runtime efficiency of the algorithms, and Section 6 contains the results. Evaluating the last two aspects fell out of scope, however, the pruning technique detailed in Subsection 7.2 may help greatly in localizing conflicts.

4 BACKTRACKING OVER MULTIPLE LEVELS

Crucially, the original implementation of WFC does not utilize any backtracking at all: upon encountering a conflict, it starts all over again. After a certain number of failed attempts, it gives up completely [4]. This approach works for the original domain of 2-dimensional tilemaps because:

- There are typically very few values each cell can take (2-5 in the illustrative examples).
- Because of this, there are relatively few adjacency constraints.
- All of the adjacency constraints are derived from an example (a video game level or an image), so there is guaranteed to be some valid arrangement, at least for the canvas size of the input.

Thus, ultimately, conflicts are unlikely enough not to disturb the generator. However, for the hierarchical model used for music generation, these points no longer apply:

- There are many chords and notes, and exponentially more possible neighbor-combinations.
- In alignment with this, most meaningful constraints exclude a larger proportion of the domain for the neighbors when used in propagation.
- A defining idea behind the model is that the composer can define their own constraints. If they are composing with some specific idea in mind, it is very possible for some of the constraints to conflict in cases that the composer is not even thinking about.

In the music generation model, there are two main kinds of constraints: **(a)** ones that are defined in terms of parameters inherited from higher levels (like the chords belonging to a key, or the melody starting on the root note of the underlying chord), and **(b)** ones that are defined in terms of the cell's relationship to neighboring cells (like each new chord being different from the last one, or the melody consisting of small steps between neighboring notes). It is the interplay between these different types of constraints – specifically when the neighbor type constraints reach over canvas boundaries – that makes backtracking non-trivial. Although this feature of constraints reaching over canvas boundaries is not an inherent part of the model (nor does it go against it in any way), it is a useful feature which we need to deal with when modelling backtracking.

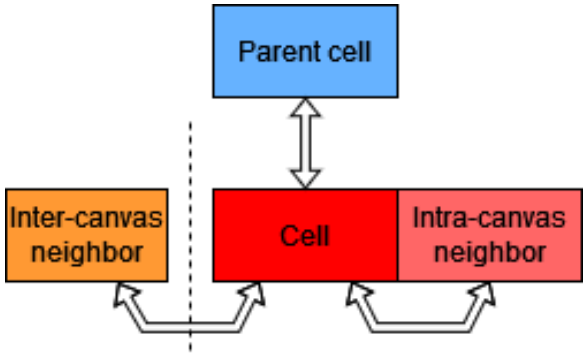

Figure 2: The three types of constraints acting on a cell, based on its position relative to the cells used to define the constraints. Note that the constraints for inter-canvas neighbors and intra-canvas neighbors are in a sense the same type of constraint, but differentiating between them is beneficial for modelling backtracking.

To illustrate the utility of constraints reaching over canvas boundaries, consider a constraint that specifies that neighboring notes in the melody should be close together. This is one of the most common constraints to add to the melody, since it helps melodies feel coherent. But if this constraint only applies within single canvases, this coherence can be broken with every chord and section change, which – though it might be the composer's intent in some cases – is something that could be undesirable in many cases. In a composer-friendly system, the composer should be free to choose whether they want such a constraint to reach over canvas boundaries, allowing to specify behaviour for either end of the canvas separately.

A breakdown of the different kinds of constraints based on the hierarchical relationship between the cells that the constraint links can be seen in Figure 2. Keep in mind that constraints for inter-canvas neighbors and intra-canvas neighbors might be the same constraint, but for the purposes of modelling backtracking, it is useful to make a distinction: conflicts between intra-canvas constraints can be resolved by immediate backtracking within the canvas, and this will not change the state of any other canvas. However, conflicts including inter-canvas neighbors can be less straightforward to resolve, especially if there are also constraints at play regarding higher values. These constraints make it so that decisions made in a canvas can affect the domain of a cell on a different canvas, similarly to the parent type relationship; in one case, the tiles are on the same hierarchical level, and in the other, on different levels.

Moving forward, we are making two big assumptions about our solver algorithm:

- The collapsing process traverses the hierarchy canvas-by-canvas. That is, it fully collapses a given canvas before making any decisions regarding cells on other canvases.
- The canvases on a given level of the hierarchy are visited in a strict left-to-right order. This means that in making a

> decision on a given canvas, we can assume that all canvases on the same level and to the left of it are completely collapsed. Note however, that it does not necessarily mean that within a canvas, cells will also be collapsed left-to-right. Furthermore (for clarifying terminology in the rest of the paper), in the specific case of a music generator, this left-to-right ordering essentially means a temporal ordering, so saying "the canvas to the left" is equivalent to saying "the previous canvas".

While approaches that do not obey these assumptions could exist, the sheer amount of possible traversal algorithms without them would extend the scope of this topic outside of what is possible to explore within this study. We would need to compare heuristics for choosing out of all available canvases and cells to collapse, all of which could have significant trade-offs regarding the evaluation criteria mentioned in Section 3. Nevertheless, we are convinced that even with the restricted domain of the problem with this assumption, meaningful research can be done to model useful backtracking algorithms.

4.1 Using a decision stack

In constraint programming, the domains of the variables are explored by making decisions: each decision is associated with a cell and a value that it is collapsed to, and it also contains some data for restoring the state before the decision was made. When a conflict is found, the latest decision is reverted, the state is restored to the one before the decision was made, and the value is removed from the domain of the variable.

Just like in a classical constraint problem with no hierarchical layout to complicate constraints, it is important to realize that any decision the algorithm makes to collapse a cell only applies **in the context of all prior decisions**. If decision x is reverted, the value associated with it will be removed from the domain of the respective cell, but only until the decision before decision x is reverted as well. This is important to mention, since it is deceptively easy to choose a backtracking model that seems to satisfy this, utilizing some property of the hierarchical structure for "optimization", while it actually falls short of properly exhausting all of the domains.

As an example, consider the following backtracking algorithm, based on the observations made in Figure 2, and illustrated in Figure 3. The idea is that if the current canvas is unsolvable, either there are fundamental conflicts between the constraints in the canvas, or the conflict is caused by a value on inter-canvas neighbor (of one of the edge cells) or a parent cell. This algorithm prioritizes fixing conflicts with inter-canvas neighbors first.

Since we are assuming that canvases are visited left-to-right on a given level, there cannot be any decisions made on the canvas to the right of the current one, so the only necessary backtracking is on the left. The algorithm backtracks and regenerates the previous canvas, until there is a different value on its final cell than there was before. This may lead to different constraints applying to the first cell of the current canvas, which may produce a solution. If the previous canvas cannot produce any more different values for its final cell, the level above has to be regenerated, as that is the only remaining place for a conflict to originate (other than it being an inherent conflict within the constraints of the current canvas).

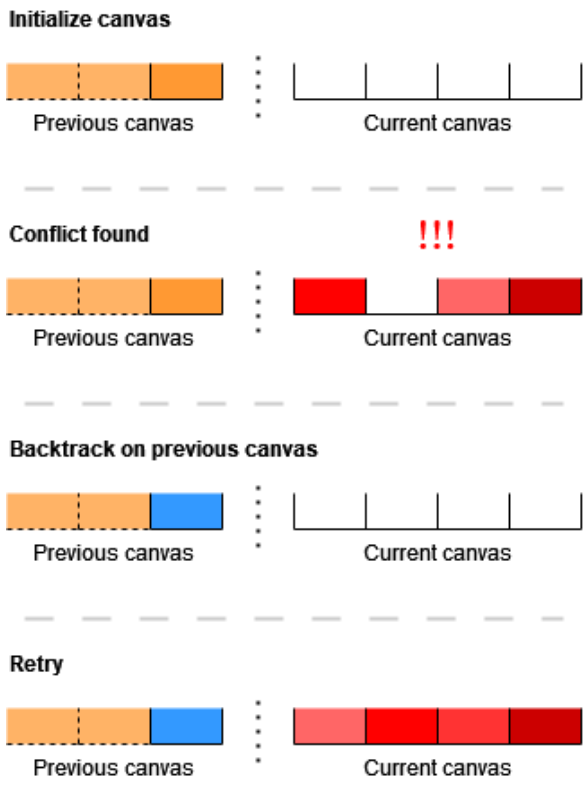

Figure 3: Basic idea for a simplistic backtracking algorithm, which is not exhaustive.

The algorithm above misses that the original domains of the previous canvas might be restricted by decisions made in the canvas before it. In other words, if decisions in canvas *1* influence the domains – and as such, the possible decisions – of canvas *2*, and the decisions in canvas *2* influence the domains of canvas *3*, then that means that the decisions in canvas *1* (as well as any other canvases before it) also need to be taken into consideration when determining whether something in canvas *3* is truly a conflict.

This example teaches us that all decisions from across the hierarchy should be considered on a single stack of decisions, and when backtracking, it is always the last decision that should be reverted.

4.2 Proposed algorithms

The assumptions and observations above lead to this: we need to traverse the tree of canvases in an order such that each hierarchical level on its own is traversed left-to-right. For every canvas visited, we collapse all of its cells, collecting all decisions on one common stack. Upon finding a conflict, the top decision on the stack is reverted, and a different decision is made.

This leads us to two basic traversal orders (also seen in Figure 4), which we are going to evaluate and compare based on their runtime efficiency:

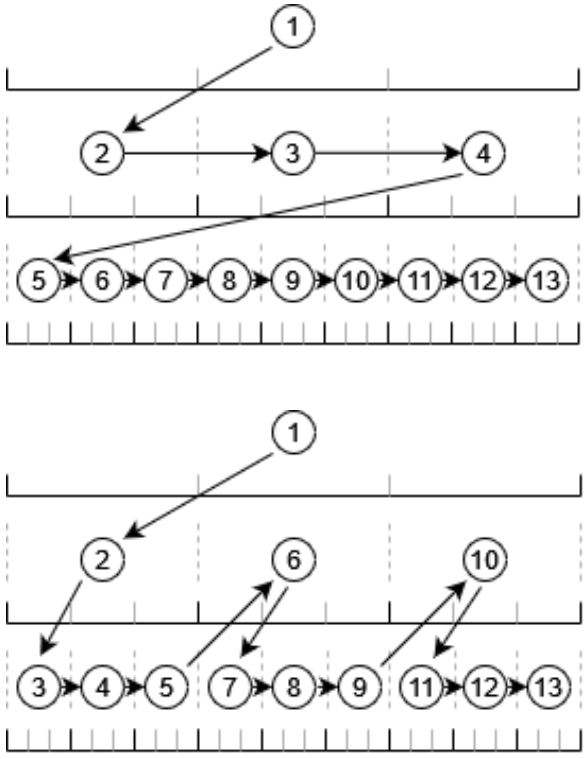

Figure 4: Breadth-first (top) and depth-first (bottom) traversals of the same simple hierarchy of canvases.

- **Breadth-first:** all sections are generated, then all chords, then all melodies. Canvases on all levels are visited from left to right.
- **Depth-first:** the canvas of sections is filled, then the first canvas of chords, then all the melody canvases under that chord canvas, then the next chord canvas, and so on.

5 METHODS

We implemented both traversal models, then ran both on the same set of constraints in order to determine whether there was a significant difference in their runtimes.

5.1 Implementation

We implemented both breadth-first and depth-first traversal of the hierarchy, based on the original repository for Varga and Bidarra's model [11]. During the implementation stage, we encountered several unexpected difficulties. Some of these included:

- With choosing different values on higher levels possibly changing the sizes of the canvases and the number of nodes in the hierarchy, saving and restoring states for decisions is not trivial. Decisions being reverted in one part of the hierarchy might have effects in a completely different area.
- Due to the recursive nature of propagation, a given issue may arise at a point where the callstack is quite deep. This makes reasoning about where the problem is in the code exceedingly difficult, also because many of the calls come from the same line of the same file, but the crucial error often happens one time, in one place.

Though there was no catch-all strategy for dealing with these issues, some techniques helped tremendously with finding the sources of issues:

- While the model uses a random number generator for making decisions, it is all based on a seed system. By fixing the seed of a setup that produced an issue, we could consistently recreate the series of events, and follow the execution of the code multiple times in order to pinpoint where it went wrong.
- For any bug found, we tried reducing the input "problem" (i.e. the extent of the constraints, the sizes of the canvases etc.) to be as small as possible. This made it easier to keep track of the states of all the canvases and also led to smaller decision stacks to inspect.

5.2 The experiment

We set up the following set of constraints:

(1) Every chord and note should be in the key of C major.
(2) Every melody canvas should start on the root note and end on the fifth note of its corresponding chord.
(3) Every melody step – including those on chord and section boundaries – should be a minor or major second (so a step of one or two semitones).
(4) Every chord needs to be different from its neighbors – including those over section boundaries.
(5) The melody should stay in the range between C4 and C6.
(6) Two sections should be generated, each with two chords.
(7) Each of the two sections generated has two options to collapse to: *Section1* has four melody notes per chord, *Section2* has five.

Constraints 1-3 mean the following when combined with constraint 7: since getting from the root of a chord to its fifth requires three steps when descending on the scale and four steps when ascending on the scale, each chord in *Section1* will have melody segments **descending** step-wise from the root to the fifth, and each chord in *Section2* will have melody segments **ascending** step-wise from the root to the fifth.

Because of constraint 5, some combinations of sections will not lead to a solution. Consider the example of the section canvas collapsing to two consecutive instances of *Section2*. Even though each section "by itself" is solvable, the constantly descending melody would drive the generator out of the allowed melody range. In this case, a constraint phrased purely in terms of the melody level has a considerable effect on the domains of the section level.

This all means that conflicts stemming from an early decision may only arise relatively late in the generation process, which – in a considerable fraction of cases – leads to excessive backtracking. This makes this set of constraints a good candidate for comparing the backtracking efficiency of the two approaches.

We ran our implementations of the breadth-first and the depth-first approach on this set of constraints 1000 times each and measured the runtime for both.

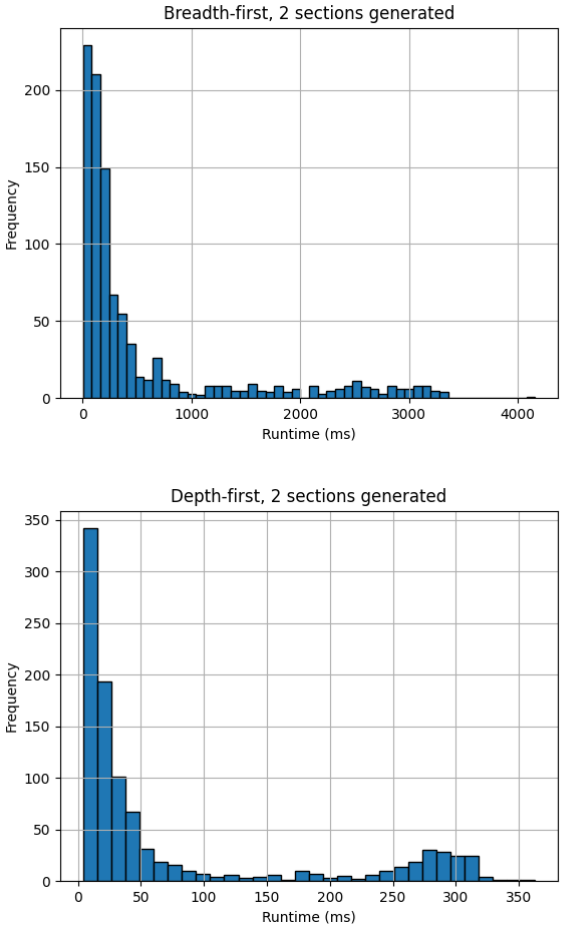

Figure 5: Runtimes of the breadth-first and depth-first models in the experiment detailed in Subsection 5.2.

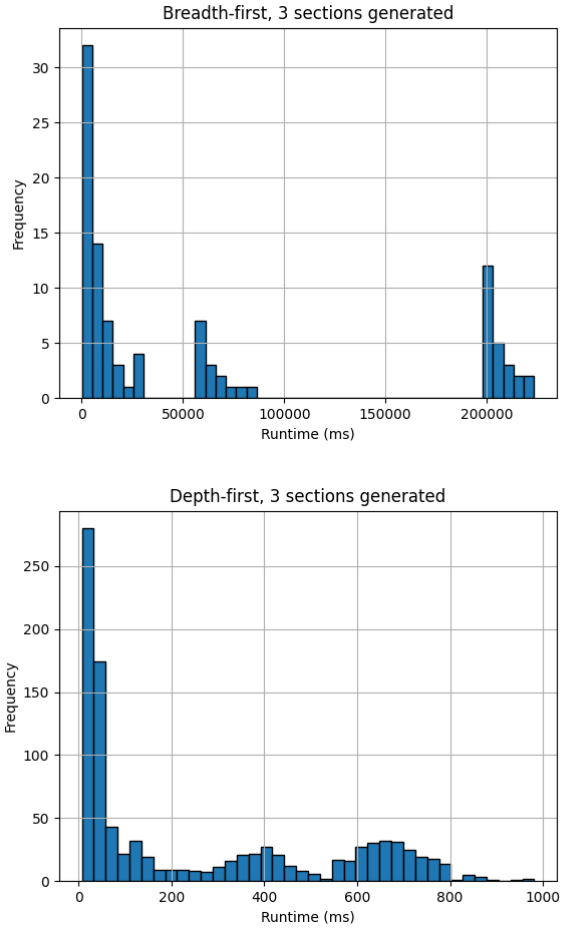

Figure 6: Runtimes of the breadth-first and depth-first models in the experiment with the larger problem.

6 FINDINGS

The average runtime of the breadth-first model was found to be 565.1 milliseconds, with a 95% confidence interval of [513.1, 617.0] milliseconds. **For the depth-first model, the mean runtime was 74.97 milliseconds**, with a 95% confidence interval of [67.78, 80.16] milliseconds. More details can be seen in the histograms in Figure 5.

Due to the remarkable difference between the runtimes between the two models, we conducted an additional experiment, with the only difference being that there were three sections generated instead of two. This setup provided us with some additional insight into how differently the different models scale when presented with a larger problem. Because of the surprisingly poor performance of the breadth-first model on this problem, we conducted the breadth-first side of this experiment with a reduced number of samples, it was only run 100 times instead of 1000. However, we could keep the sample size at 1000 for the depth-first model.

The average runtime of the breadth-first model for the larger problem was found to be 63567 milliseconds, with a 95% confidence interval of [47291, 78943] milliseconds. For the depth-first model, the mean runtime on the larger problem was 256.9 milliseconds, with a 95% confidence interval of [239.9, 273.8] milliseconds. More details can be seen in the histograms in Figure 6.

7 DISCUSSION

The depth-first model was significantly faster than the breadth-first one. Our explanation for this is that breadth-first wastes a lot of time exhausting the domains of variables which have nothing to do with the conflict. For example, consider the case where the conflict is found under one of the early chords, on the melody level, and the problem lies in the choice of the chord above. Before breadth-first can revert this faulty decision, it has to exhaust the domains of all following chord canvases first, which will not contribute to resolving the conflict at all. Depth-first combats this much more directly.

Note that the second experiment is merely meant as a demonstration of how much worse the runtime of the breadth-first method scales when compared to depth-first. The sample size 100 is clearly

too small to get a precise value for the expected runtime of this setup, which is also apparent in how wide the confidence interval is. However, we think that it is still very clear that the depth-first model dominates the breadth-first one, with a much more significant time difference than in the case of the smaller problem. Considering the mean runtimes, in the case of the original experiment, depth-first was about 7.5 times faster, while for the larger problem, this factor grew to almost 250.

Notice the three disconnected groups of bins in the breadth-first version of the experiment with the larger problem. Our explanation for this is the following: in the larger problem, it is more likely that two instances of *Section2* will end up next to each other, which – as discussed earlier, due to the range of the melody – will necessarily lead to a conflict. However, breadth-first will exhaust the domains of all of the chords before reverting any decision concerning sections. Our theory is that the fastest group of bins belongs to the cases that the assignment of sections was valid for the first try, the middle group belongs to the cases where the last section decision (of three) had to be reverted to find a solution, and the slowest group is where the second section decision was faulty.

7.1 Limitations

It is important to explore the ways in which this paper's findings are limited. Even though the concept is proven, there are questions mentioned earlier in this paper which would be great topics for further research:

- Evaluating the two approaches proposed in this paper based on the latter two aspects proposed in Section 3. For soft constraint satisfaction, a system could be set up which evaluates how well a given output corresponds to some soft constraints. Afterwards, we could generate many pieces with those soft constraints and see if there is a difference between their "scores".
- The two big assumptions made in Section 4 are in no way imperative to backtracking in general, they were simply made to scale down the vast number of possible backtracking approaches, and to ease implementation. Other approaches – violating one or both of these assumptions – could be proposed and evaluated.
- The experiment based on our implementation is sadly not a guarantee for the theoretical, ideal runtime efficiencies of the algorithms. Further optimization of the various implementations could potentially lead to different results. Nonetheless, we believe that the general finding of the depth-first method fixing conflicts quicker than the breadth-first method will still apply.

7.2 Additional pruning: Isolation before backtracking

In this subsection, we propose a pruning extension to the breadth-first backtracking model, which has the potential of counteracting some of the shortcomings of the algorithm.

Notice that, in the breadth-first approach, even though it is possible that the past decisions on the same level do not play a role in an arising conflict, they will still be reverted first. To combat this, we can introduce the following optimization (also seen in Figure 7).

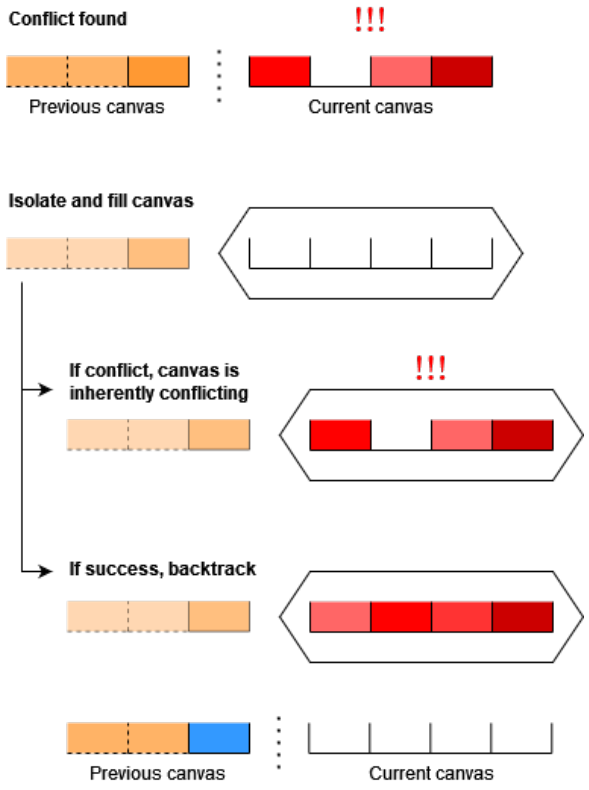

Figure 7: An improvement to the breadth-first backtracking model, avoiding unnecessary backtracking on the same level.

Upon finding that the current canvas has a conflict, instead of backtracking over the previous canvases on the same level, we first create a new canvas with the same parameters and constraints as the current one, but we do not relate it to the neighboring canvases of the original canvas. With this, we essentially remove all the constraints between inter-canvas neighbor cells: if the conflict is then resolvable, it has to stem from values chosen higher in the hierarchy. So there are two cases:

- **The isolated canvas finds a conflict:** Even with the neighboring canvases ignored, the canvas is unsolvable, so it would be pointless to try and backtrack on the same level. The backtracking process can skip forward to the point where a decision on the level above is reverted.
- **The isolated canvas finds a solution:** There is no inherent conflict within the canvas or coming directly from above, so backtracking on canvases on the current level *might* solve the conflict. Regular backtracking detailed above can commence. In addition, this combination of higher values for the canvas can be stored as *not inherently conflicting*, further optimizing later checks of the same kind.

Whenever this optimization finds an inherent conflict, the speedup caused should be significant, especially if the conflicting canvas has multiple canvases to the left of it on its level. In this case, a significant (and clearly solution-less) portion of the search tree is

trimmed away. However, it does have some computational costs, and if no inherent conflict was found, this provides no speedup at all.

We can see that the degree to which this pruning extension to the algorithm is useful depends on a multitude of factors, most importantly whether the conflict was inherent to the canvas. Other factors include: the number of decisions made previously on the same level, the number of alternative choices at those decision points, as well as the sizes of the branches of the decision tree that were left completely untouched, and the specific implementation.

We suspect that the way this method finds inherently impossible values for certain cells on higher levels could be of tremendous help in localizing conflicts (see criterion 4 in Section 3).

Unfortunately, a proper analysis of the best-case and expected speedup obtained from this extension to the algorithm, as well as the prospects regarding conflict localization is too large an undertaking for this study. It is, however, an excellent topic for future work.

8 CONCLUSION

In this paper, we have explored the challenge of implementing backtracking algorithms within a hierarchical procedural music generation model based on the Wave Function Collapse (WFC) algorithm. We have highlighted the inherent complexities introduced by the hierarchical structure, where constraints between different canvases create dependencies that complicate the backtracking process.

We proposed two primary traversal methods for the backtracking algorithm: breadth-first and depth-first.

The evaluation criteria for our models were based on completeness, runtime efficiency, soft constraint satisfaction, and conflict localization. Although our experiments focused primarily on comparing the runtime efficiency of the two approaches, the additional aspects offer a valuable basis for further research.

Our findings indicate that depth-first traversal is more efficient under the given constraints, for reasons that would also apply for a different set of constraints.

Overall, our research contributes to a deeper understanding of backtracking in this hierarchical model, allowing composers to use many constraints with the guarantee of completely exploring the space of possible solutions.

9 RESPONSIBLE RESEARCH

One of the important advantages of WFC over other methods for procedural content generation is that a significant number of the ethical considerations demanded by many AI systems have simple solutions for this algorithm.

9.1 Transparency and Explainability

Transparency and explainability are crucial in the deployment of AI technologies like WFC, especially in creative domains such as music composition.

For WFC, transparency means that users can understand how the algorithm makes decisions, which patterns it chooses, and how it interprets constraints.

Explainability involves providing insights into why certain choices are made by the algorithm—such as why a particular chord progression or melody was generated—allowing users to trust and effectively interact with the technology. This is particularly important in collaborative environments where artists may wish to tweak algorithmic suggestions.

WFC is a white-box algorithm, meaning that given an input, anyone could generate an output, solely based on the description of the algorithm. This property is preserved in the hierarchical model used for music generation.

9.2 Enhancement vs. Replacement

The mixed-initiative approach of WFC in music generation underscores its role as a tool for enhancing rather than replacing human creativity. In this setup, there is an iterative loop between the user and the algorithm. The process starts with the user defining initial constraints, which the WFC algorithm then uses to generate a composition that adheres to these parameters. Upon reviewing the algorithm's output, the user can draw inspiration and identify aspects they might want to alter, leading them to tweak the input constraints based on their creative judgment and preferences.

This iterative process allows for a deeply interactive and collaborative form of composition, where the algorithm serves as both a source of inspiration and a sophisticated tool that extends the creative capacities of the user. By enabling a creative dialogue between the composer and the algorithm, WFC supports the artistic process, allowing composers to explore new ideas and refine their work, all while ensuring that the technology remains a helpful tool rather than taking over the creative process.

9.3 Use of Data

In the application of WFC for music generation, the algorithm operates uniquely as it does not require any training data or examples to function. Instead, it relies entirely on explicit constraints provided directly by the user. This approach significantly shifts the data use paradigm in AI by eliminating the need for large datasets and the associated concerns of data privacy and copyright infringement. By relying on user-defined constraints, WFC allows composers to maintain full control over the creative output, ensuring that the compositions are both original and closely aligned with the artist's intent.

REFERENCES

[1] Shaad Alaka and Rafael Bidarra. 2023. Hierarchical Semantic Wave Function Collapse. In *Proceedings of the 18th International Conference on the Foundations of Digital Games* (Lisbon, Portugal) *(FDG '23)*. Association for Computing Machinery, New York, NY, USA, Article 68, 10 pages. https://doi.org/10.1145/3582437.3587209

[2] Roman Barták. 2005. Constraint propagation and backtracking-based search, 1st International Summer School on CP. *Acquafredda di Maraeta* (01 2005), 11–15.

[3] Michael Beukman, Branden Ingram, Ireton Liu, and Benjamin Rosman. 2023. Hierarchical WaveFunction Collapse. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 19, 1 (Oct. 2023), 23–33. https://doi.org/10.1609/aiide.v19i1.27498

[4] Maxim Gumin. 2016. Wave Function Collapse Algorithm. https://github.com/mxgmn/WaveFunctionCollapse

[5] Isaac Karth and Adam Smith. 2021. WaveFunctionCollapse: Content Generation via Constraint Solving and Machine Learning. *IEEE Transactions on Games* 14 (2021), 364–376. https://doi.org/10.1109/TG.2021.3076368

[6] Isaac Karth and Adam M. Smith. 2017. WaveFunctionCollapse is Constraint Solving in the Wild. In *Proceedings of the 12th International Conference on the Foundations of Digital Games* (Hyannis, Massachusetts) *(FDG '17)*. Association

for Computing Machinery, New York, NY, USA, Article 68, 10 pages. https://doi.org/10.1145/3102071.3110566

[7] Hwanhee Kim, Seongtaek Lee, Hyundong Lee, Teasung Hahn, and Shinjin Kang. 2019. Automatic generation of game content using a graph-based wave function collapse algorithm. In *2019 IEEE Conference on Games*. IEEE, London, UK, 1–4.

[8] Donald E. Knuth. 2022. *The Art of Computer Programming, Volume 4B: Combinatorial Algorithms, Part 2.* Addison-Wesley Professional, Boston, MA, USA.

[9] Thijmen SL Langendam and Rafael Bidarra. 2022. miWFC - Designer Empowerment through Mixed-Initiative Wave Function Collapse. In *Proceedings of the 17th International Conference on the Foundations of Digital Games* (Athens, Greece) *(FDG '22)*. Association for Computing Machinery, New York, NY, USA, Article 66, 8 pages. https://doi.org/10.1145/3555858.3563266

[10] Peter Van Beek. 2006. Backtracking search algorithms. In *Foundations of artificial intelligence*. Vol. 2. Elsevier, 85–134.

[11] Pál Patrik Varga and Rafael Bidarra. 2023. ProceduraLiszt Repository. https://github.com/ProceduraLisztDevs/proceduraliszt

[12] Pál Patrik Varga and Rafael Bidarra. 2024. Harmony in Hierarchy: Mixed-Initiative Music Composition Inspired by WFC. Submitted for publication.