Enhancing Wave Function Collapse with Design-level Constraints

Arunpreet Sandhu
Computer Science
University of California, Davis
Davis, CA, USA
asisandhu@ucdavis.edu

Zeyuan Chen
Computer Science
University of California, Davis
Davis, CA, USA
zeychen@ucdavis.edu

Joshua McCoy
Computer Science & Cinema and Digital Media
University of California, Davis
Davis, CA, USA
jamccoy@ucdavis.edu

ABSTRACT

WavefunctionCollapse (WFC) is a non-backtracking, greedy search algorithm that is known for being able to use a small number of constraints to generate large consistent output. Developers have explored the algorithm's extensibility and usability through various implementations spanning from 3D world generation to poetry creation. However, the algorithm has not been generalized and the implementations have been specifically tailoring to the context in which they are used. There has been no generalized integration of design constraints in WFC. In this paper, we explore WFC as a constraint satisfaction solver to integrate design principles and practices by modifying WFC. First, we extend the local constraint reasoning by incorporating constraints that can work over any distance and non-spatial constraints. Next, we further manipulate the generative space by introducing weight recalculation and dependencies. Lastly, we evaluate our design-focused variant of WFC against the original implementation to examine the computational time and memory usage. In summary, this paper describes a technical implementation of integrating design constraints into WFC and analyzes the computational trade-offs.

CCS CONCEPTS

• Applied computing → Computer games • Theory of computation → Constraint and logic programming• Computing methodologies → Probabilistic reasoning

KEYWORDS

Procedural Content Generation, Wave Function Collapse, Level Generation

FDG '19, August 26–30, 2019, San Luis Obispo, CA, USA

ACM 978-1-4503-7217-6/19/08…$15.00

https://doi.org/10.1145/3337722.3337752

1 Introduction

Procedural Content Generation (PCG) can reduce the resources spent on generating new assets by augmenting or enhancing different aspects of asset creation [1], [2]. In tasks like level generation, PCG accelerates asset creation which leads to less time spent on asset creation [3]–[5]. By speeding up the process, PCG reduces the time costs and can help reduce the overall production costs. There are costs associated with using PCG.

PCG techniques require a varying degree of upfront costs to implement and configure them into a state where they can generate content. These costs can be intimidating, but they depend on how much knowledge the developer wants to encode. Techniques that required a knowledge base [4], such as Answer Set Programming (ASP), might take longer to construct but the output will be more refined than other techniques that stochastically produce artifacts. Although encoding this information is a task within itself that can lead to fruitless output from programming difficulty. As a result, algorithms with minimal upfront costs are favored in hopes to accelerate an existing asset production pipeline.

Developers must decide if the upfront costs of PCG are worth the time savings. An example of these time costs is understanding how to configure a system to meet the design goals [4], [6]. It can be even more difficult than creating assets. Systems are hard to create but have a long tail of productivity. These upfront costs lead to developers avoiding techniques in favor of more streamlined algorithms to accelerate offline asset production. However, there have been pushes to make these more difficult to use techniques more accessible.

Techniques such as constraint satisfaction and ASP for PCG requires the developer to have a deep understanding of how these techniques work. In comparison to stochastic techniques, constraint satisfaction requires more information about the design space. The developer must know how to encode the design knowledge into the system which can slow down the process entirely. To combat this, mixed initiative tools were developed in order to give these unwieldy techniques more accessibility through developing a design tool for them. However, these tools are domain specific (e.g. a level design tool can only do level design) and require deep understanding in order to transfer to other design domains. This is why WaveFunctionCollapse (WFC)

[7], a constraint satisfaction algorithm, is a strange occurrence in how quickly it was adopted by the PCG community.

This approach lends PCG techniques to be mostly used during the development stage, also known as offline PCG. However, using PCG during gameplay, also known as online PCG, is possible but requires more rigorous testing to ensure non-generic output. An example of generic PCG output is *No Man's Sky*, a game that relies on online PCG, which leads to much of the game becoming generic after a few hours of play. Players criticized the game for its repetitiveness during initial release [8]. However, there are examples in which online PCG are successful, such as *Spelunky* [9]. A reason for why the PCG system in *Spelunky* succeeded and the PCG system in *No Man's Sky* did not stem from applying the right type of PCG.

WFC gained traction in the industry due to its ease of use [10]. Unlike other constraint satisfaction algorithms which require constraints and domain knowledge for input, WFC can work with small input in order to generate larger outputs [10]. Two examples of WFC usage are *Bad North* [11], a Viking real-time strategy game, by Plausible Concept and *Caves of Qud* [12], a science fantasy roguelike, by Freehold Games. Both games use a design focused variation of WFC for online level generation [10]. Both games show that WFC can be extended in order to reason over design spaces, such as levels, which has led to our intrigue in attempting to enhance WFC with more design-oriented constraints to improve the output.

In this paper, we contribute an enhancement of the WFC algorithm consisting of the addition of new design-friendly constraints. These new additions consist of non-local constraints, upper and lower bound guarantees, and modifying the generative space through dynamic weighting. We then present the results of the experiments and look at the computation time and memory usage. The results are then analyzed for the tradeoffs between the capabilities they add and the additional resources they consume.

2 Related Work

WFC was originally inspired by texture synthesis and quickly adopted by PCG communities [10]. Through social media, experimental game designers showcased WFC as a 2D level generator. It becomes clear that WFC could handle a higher dimensionality [13] and designers began using it to generate 3D levels. It was also discovered that WFC can generate content in other domains such as poetry generation [14]. The technique was tested to see the limits of WFC as a constraint solver such as reformulating WFC in ASP [10]. WFC has also been used in procedural content generation via machine learning in order to generate high-quality artifacts from a single input image [15].

Most of the work revolving around WFC has been focused on testing WFC's ability to generate levels based on image data and extending the design domains that WFC can reason over. For example, *Bad North* and *Caves of Qud* add design constraints into WFC through modification. *Bad North* adds in another search heuristic during generation to make levels that have at least one navigable path for agents. This is to ensure the level has at least one path in which enemies are able to take to attack the player's base. *Caves of Qud* has WFC go through a multipass system to add in more variety within the level. It also uses WFC with other map generation techniques to create more nuanced outputs.

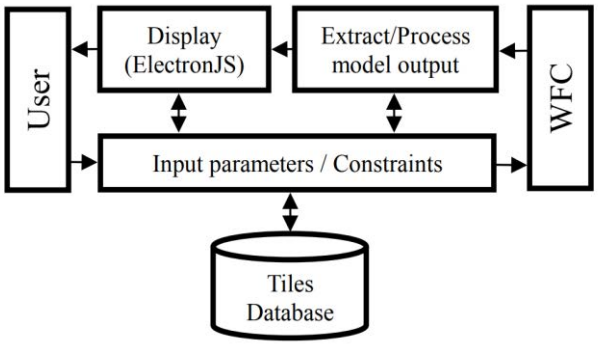

Figure 1: WFC system architecture

Both of these games use WFC during gameplay, or also known as online PCG. Most design with constraint satisfaction algorithms occurs during design time, or offline, with tools like *Tanagra* [16]. Using a mixed-initiative approach [17] [18], *Tanagra* allows level designers to have input during level generation. This allows for more design focused generation rather than the usual approach where a designer refines already generated content. As such, the work done in *Bad North* and *Caves of Qud* is interesting because it focuses on online design constraints. The work in this paper follows this online design modification to WFC.

3 Technical Description

This section provides an introduction to WFC in order to set the foundation for describing the additions. The first of these additions will be a weighted choice, which is included in the tests for the subsequent design constraints described in the paper. As for why weighted is notable, is due to how weighted choice changes all other design additions. The weighted choice is a modification to WFC, while other additions are an extension of WFC.

3.1 WFC

At its core, WFC is a constraint satisfaction solver that uses data-driven methods to search a generative space. It takes input parameters such as a collection of elements, their local neighbor constraints, and weights to generate the desired content (Figure 1). The inspiration for WFC is quantum physics concept bearing

the same name, however, WFC utilizes information theory [7], [10], in order to calculate the probability of an element. This probability is referred to as entropy, which is similar in concept to entropy from physics. As such, entropy is an important factor that drives the choices WFC make as it iterates through the generative space. This means, a developer can gain more control of the generative space by adding design constraints that manipulate entropy, which will be addressed soon.

For clarification purposes, the following definitions and descriptions will be used to explain WFC. Going forward, all examples will be grounded as image data. Tiles are the basic building blocks of a map which are usually 32x32 pixels. They are individual pieces that form a map when placed adjacently and are the same size. Neighbors are the local constraints within WFC that consist of tiles which are adjacent to a specific tile. Each of these tiles also has a designated weight which is represented as a percentage. A wave is a possibility space that defines a map and for our implementation, can be seen as a 3D matrix. Each wave element is an array of Boolean values that is the same length as the total number of tiles in the system. A conceptual example of a wave can be seen as a checkerboard where each square has a stack of numbered chips. When a chip is true, this means this chip remains in the stack. A chip is removed from the stack when it is set to false. We will come back to this example later, but first, an explanation of how WFC removes tiles from elements is required. The initial state, each stack contain all possible chips. In the first stage, a square will be randomly chosen and a chip will be randomly chosen from the stack. By choosing this chip, all other

Figure 2: A pair of tiles with symmetries ‘T’ (left) and ‘L’ (right) placed side by side are rotated 0°, 90°, 180°, and 270° counter clockwise where the ‘T’ (left) tile is the pivot

chips are removed from that stack. By removing these chips, adjacent squares affected by possible constraints. These constraints will be substantiated later in this section, for this example if a chip in an adjacent stack is removed, then there is a possibility of another stack will remove a different numbered chip.

The main loop for WFC is to observe an element, propagate the choice, and ban appropriately. Observing an element is similar to the physics namesake, in which looking at an array forces only one element in the array to be true. In the checkerboard example, if a stack is observed, then only one chip will remain in the stack.

Symmetry	0°	90°	180°	270°
X				
I				
\\				
L				
T				

Table 1: Tile symmetries ‘X’, ‘I’, ‘\\’, ‘L’, and ‘T’ rotated 0°, 90°, 180°, and 270°. a tile with symmetry ‘X’ will remain the same regardless of rotation, which means the cardinality of ‘X’ is 1. A tile with symmetries ‘I’ or ‘\\’ gives a cardinality of 2 while symmetries of ‘L’ or ‘T’ produces a cardinality of 4.

However, the removed items are not discarded. Instead, they are pushed into an array of banned items. Because WFC uses constraints, the banned items will be propagated out to the adjacent elements to check for incompatibility. The constraints themselves are developer defined or data-mined from an input sample. As for what is propagated, it is not the information about the chosen item, but instead the removed items. These removed items are considered to be banned, as they are banned from existing within the specific element. Through this, the banned items in the observation stage are propagated through the wave and each wave element enters a banning stage. The banning stage is to check the constraints if the element is in a state of compatibility. If there are incompatible items within the element, then those are banned. These banned items are then propagated throughout the wave and the banning stage continues until there are no more items to be propagated through the wave. However, if an element has no items left within it, then the entire wave would be considered unsatisfactory. As in there is no possible configuration in which the wave will have at least one item within each element.

WFC begins with generating an initial state from the input. The initial state consists of a set of tiles including the name, symmetry, weight, and its local constraints. From this, an array of all unique tiles is created from the name and symmetry

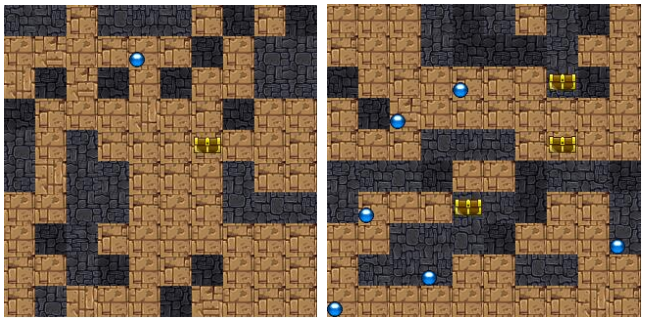

Figure 4: 10x10 tile maps are generated using non-local constraints to showcase the addition of items (chest and blue orb) with dependency (left) and frequency (right).

parameters by utilizing the rotations of tiles to generate sets of possible tiles. Table 1 showcases how a tile might be rotated. The symmetries are divided into 'X', 'I', '\\', 'L', and 'T' types, where the number of unique tiles is produced by rotating the input tile by 90° counter-clockwise.

Next, the algorithm creates a set of all possible neighbor tiles from the given constraints. These parameters are passed as a pair of tiles. For example, tile A and B have symmetries of 'T' and 'L', respectively. As the pair rotates, it forms the possible neighboring tiles above and below the pivot tile, which is the tile that remains in place as the neighbor tile rotates around it (Figure 3). For example, the 90° rotation of the tile A (left) and tile B (right) creates an above and below neighbor pair where tile B with a 90° rotation can be placed above tile A with a 90° rotation and vice versa. However, in order to get a proper rotation, one tile is assigned as a pivot tile. These pivot tiles will be considered the "fixed" position during the rotations, even though the pivot tile is still rotating.

After generating the set of neighbors, WFC calculates the total sum of all the tile weights per element. This sum is used to calculate the entropy of the element. Within the initial state, all elements within the wave have equal entropy. Do note that the entropy of each element is associated with the tiles. If a tile is removed, then so does the weight associated with that tile. After the first observation/propagation loop, the entropies will no longer be equal to each other and will be recalculated before the next observation stage. WFC now begins the observation propagation cycle.

First, WFC observes a wave element based on its entropy. Initially, entropy is the same throughout the wave, so a random element is chosen to be observed. Then, a random tile is chosen and all other tiles are banned and put in an array of elements to remove (ETR). This is the initial set of tiles to use for constraint comparison. Each tile added from the previous iteration is popped off and used for comparison. This is to ensure that the only tiles being affected are adjacent tiles. If there is any tile which is incompatible based on the ETR, then they will be removed from the element and added to the ETR. This continues until there are no more banned tiles within the ETR. At this point, the entropy for each element is recalculated to reflect the changes. WFC loops back to the observation stage. This process is repeated until the entire wave has been observed or a contradictive state, where at least one element has no possible tiles.

This process describes the base implementation of WFC. Moving forward, the paper will be altering parts of the base implementation of WFC to add forced observations, upper and lower bounds of elements observed, non-local constraints, weight recalculation, and other types of propagation.

3.2 Weighted Choice

The first addition to WFC is a modification within the algorithm that changes the way WFC chooses tiles. In the base implementation, the element choice is determined by entropy, which in turn is calculated from the weights given by the tiles. However, choosing a tile is random, which means the weight is negligible after an element is decided. As a result, the map produced does not fully represent the weight of the tile when choosing a tile. While random choice can cause unconventional results, the process fails to capture the importance of the tile's weight. Weighted choice, however, is able to produce a map that is reflective of the tile's weight. This work uses a combination of binary search with cumulative weight method to implement weighted choice.

3.3 Design Constraints for WFC

3.3.1. Non-local Constraints. The first design constraint added to WFC is the non-local constraint that modifies the main observation/propagation loop. To further test if the non-local constraints work within WFC, non-tile items will be used. Also, this allows WFC to give developers more control of the generative space (Figure 4). In order to do so, an additional observation step for items is added, which modifies the data structure. The reason why only an extra observation step is added for items is that they different than tiles. They are interacted with in a different way than just tiles. The player can interact with items during a game such as a treasure box, a door, or a key. They require some sort of use case within the game map they are in. Because of this constraint, the observation step needs to be altered in order to allow for non-local constraints. As such, we force another observation step to happen based on the dependencies of the item or tile.

The idea behind this extra observation step is to trigger the observation of a dependent item or tile within a defined area. To achieve this, a force function is incorporated into the observation step that mimics an observation. First, the developer needs to define the distance from the trigger item to calculate the subarea of the map to choose in. Within the observation phase, a check is performed to determine whether a forced observation is needed. If yes, the force function will calculate the subarea and choose within the subarea. If the object type is a tile, then WFC will run

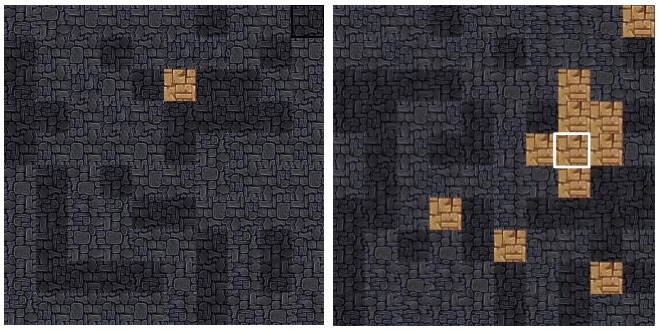

Figure 5: 10x10 tile maps are generated to showcase WFC with (right) and without (left) the weight recalculation constraint. The white box in the right image shows the tile with weight recalculation.

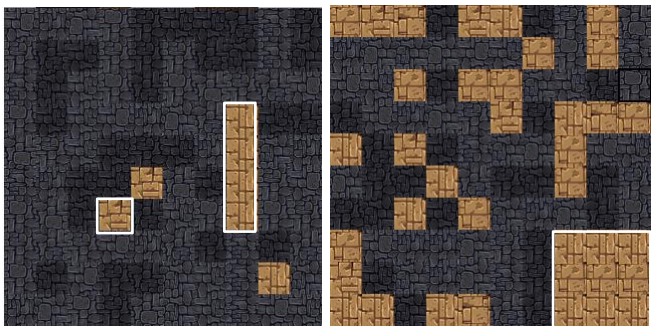

Figure 6: 10x10 tile maps are generated to showcase WFC with (right) and without (left) an area propagation of at most 5x5. The square white boxes in the left image shows the tile that triggers area propagation. All the white boxes in the left image are valid tiles when area propagation is triggered. The white box in the right image shows the area that potentially has area

the observation-propagation loop within the subarea until the desired area is completely observed. In other words, the force function will perform a small scale WFC algorithm in a subspace of the wave.

If the object is an item, then the algorithm executes a variation of the observation step that randomly chooses an element to place its dependency (Figure 4). Then, the algorithm will continue to execute the observation/propagation loop until the wave is completely observed. We introduce upper and lower bounds in the force function to give developers some control for the placement of items relative to each other by calculating the area of a bounding box from given parameters. The lower bounds provide WFC with the ability to observe somewhere that is not locally bound while the upper bound limits WFC's non-local reach. This capability allows for finer control of where to observe associated items. For example, a key or lock can be placed and then a forced observation of the complement item occurs N range of tiles away.

Suppose the developer wants to limit the number of a certain item being observed. To achieve this, a frequency counter is introduced to set a limit on the number of certain items within the map. Each time before an item is observed, WFC checks its frequency counter. This frequency counter acts as count down for a certain item. If the frequency counter is greater than zero, then the item is observed. Once the frequency is zero then WFC replaces the chosen item with an empty item to be placed. This, in effect, is the upper and lower bound control of generating content within WFC.

3.3.2 Weight Recalculation. The next type of constraint added to WFC is weight recalculation. It alters the probability space to force the saturation of certain tiles within an area. In order to achieve this, another trigger is added to the observation step to invoke the force function previously mentioned. If the weight of the tile is triggered to change, then the function iterates through the wave and removes the old weight and replaces with the new weight if the tile hasn't been removed. At this point, the entropy must be recalculated.

The decision of introducing the weight recalculation constraint lies in the relationship between an item and the item's surroundings. For example, if a lock and key were to exist then it might influence the weight of having enemies within the space or vice versa. In other words, the area around an item can also influence other factors such as enemy placement. In this hypothetical, if the developer wants to have fewer enemies surrounding the items then the weights will be recalculated based on this. As shown in Figure 5, a cluster of trigger tiles that contain the weight recalculation flag can be seen at the top left corner of the right image. This idea of environmental influences leads to the next constraint test within WFC.

3.3.3. Area Propagation. The last modification incorporated into WFC is called area propagation. The concept behind this is similar to that of the extra observation step where a specific area is calculated and then an event is forced. The difference here is that instead of choosing a tile or item for each element in the specified area. Within the specified area, groups of undesired objects are banned within the subspace, such as enemies from a slowly forming puzzle room. Area propagation modifies the propagation step within WFC. For example, if a light source within a dungeon is placed then the surrounding area should only have tiles associated with light, as opposed to dark. This gives developers flexibility in creating subareas within a map. Instead of choosing definite objects out of all possible choices within the surrounding area, area propagation takes away choices to constrict the generative space. As can be seen in Figure 6, the trigger tile produces a section of the map that is composed of valid tiles boxed in the left image.

4 Evaluation / Analysis

This paper examines the impacts of the additional design constraints to WFC by analyzing the execution time and memory usage for each modification mentioned in Section 3.

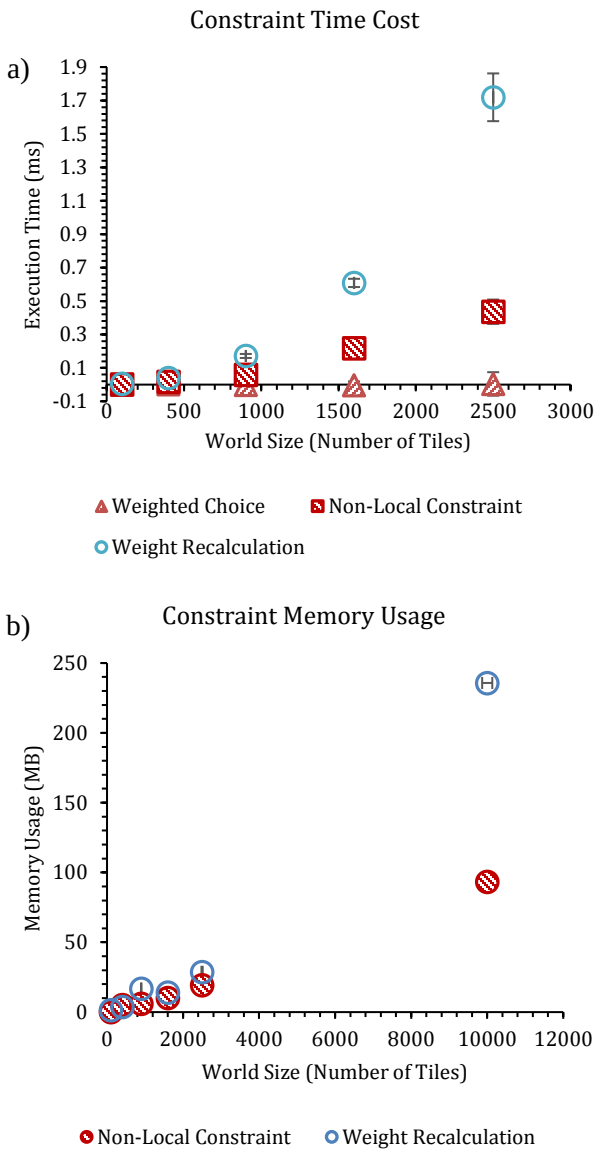

Figure 7: a) A summary graph of time cost for weighted choice (purple triangles), non-local constraint (red squares), and weight recalculation (blue circles) in seconds. b) A summary graph of memory usage for weighted choice (purple triangles), non-local constraint (red squares), and weight recalculation (blue circles) in MB.

These metrics show the tradeoffs when using these new constraints. Each modification (weighted choice, non-local, weight recalculation and area propagation) were tested against the base WFC implementation by enabling only the constraint tested. Of the additional constraints, area propagation has proven to be hard to conduct run time and memory analysis due to the increase in resulting conflict introduced by said constraint. Therefore, the analysis done on area propagation is the number of successful runs within 50 trials and the associated percentages.

For each analysis, we conducted 50 runs using various map sizes (10x10, 20x20, 30x30, 40x40, 50x50, and 100x100) where tiles are used as the units for the height and width. The reason behind this is to keep conformity. Tile information is passed in as an array of objects containing name, symmetry, weight, and type. Neighbors constraints are passed in as adjacent neighbor pairings in a neighbors array. For each neighbor pair, the neighbor value is defined by a string composed of two numbers. The first represents the name of the tile and the second is the rotation of the tile. The changes to input for each additional constraint is explained in their respective analysis sections.

For run time testing, WFC is time stamped when started and at the end of the wave generation. To try and avoid hardware dependencies, a single machine was used for all the testing. For memory testing, NodeJS's native implementation is used to check memory usage. It is important to note that NodeJS's process can be imprecise. However, the memory tests conducted for this paper are for preliminary testing purposes to see if there are any marginal changes to memory.

A summary graph of the runtime and memory analysis for all additions and the base implementation of various map sizes are depicted in Figure 7. As can be seen, memory usage increases for both non-local and weight recalculation constraints. Due to the possibility of optimization and caching, weighted choice constraint gave inconclusive results for the memory usage tests.

4.1 Weighted Choice

The weighted choice does not require input changes as it is an internal modification to the algorithm that is not dependent on the input. Thus, it uses the same input as that of the base implementation. In the time cost analysis, the average time stays within the margin to the baseline with map size up to 2500 tiles (Figure 8). However, the standard deviation increases as map size increases. In other words, the spread of time cost can be big with large map sizes. As shown in Table 1, run time can be delayed by as much as 100ms for a map size of 2500 tiles. Interestingly, the average run time for weighted choice is faster than the base implementation by almost 3 seconds for the 10,000-tiles map size where the maximum increase in run time is about 0.6 seconds and decrease is almost 10 seconds (Figure 8).

As mentioned previously, memory usage constraint gave inconclusive results in terms of WFC efficiency, but give some rough number to how computationally expensive WFC is. One possible explanation is a JavaScript optimization or caching is triggered. Further testing needs to be done to validate this. However, in Figure 8 there appears to be a trend of at worst case, the weighted choice becomes exponential in for the computational time

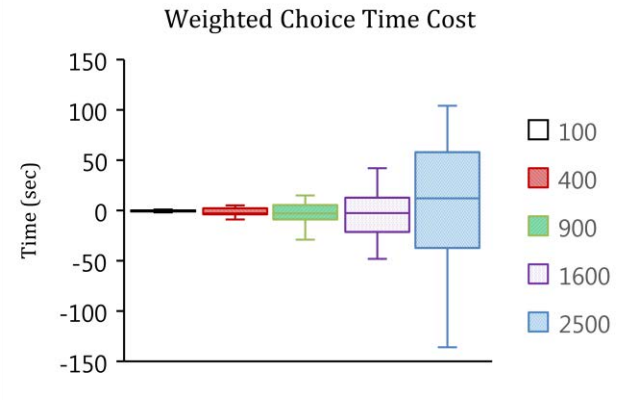

Figure 8. Time cost time for weighted choice in seconds. From left to right, each box plot corresponds to a map size within the legend in the order of top to bottom. The legend shows the total number of tiles within a map from 100 tiles (a 10x10 map) to either 2500 tiles (a 50x50 map) or 10k tiles (a 100x100 map).

4.2 Non-local constraints

In addition to the input from the base implementation, non-local constraints also require information on items and constraints. Each item requires name, weight, and frequency as parameters. Name is an identifier for each item. Weight is a value that depicts the relationship between items. Frequency is the number of times an item is required to be observed within the wave. Inputting a negative number for frequency indicates that there is no upper or lower bound for a certain item. The first input for items will always refer to an empty item. This is a placeholder item that indicates no items exist at a location. Therefore, the frequency is always negative. For subsequent items, a parameter called dependency is also passed to inform WFC of the bond between certain items. The value used to indicate dependency are the names of the items that are dependent on a certain item. In other words, if item A is dependent on item B then the parameter value for dependency of item B is A. The constraints are formulated in a way which each constraint has whether the constraint is for tiles or items, the names of the things being constrained, and the amount of distance between the constrained is passed into WFC.

The results for the time cost show that the average time increase for non-local constraints is the second most intensive. As depicted in Figure 9a, non-local constraints increased execution time that ranged from about 1ms for 100 tiles to 440ms for 2500 tiles. In fact, the trend for the increase in runtime relative to the map size can be fitted to a polynomial regression (Figure 9a), which provides a way to extrapolate time cost to higher map sizes.

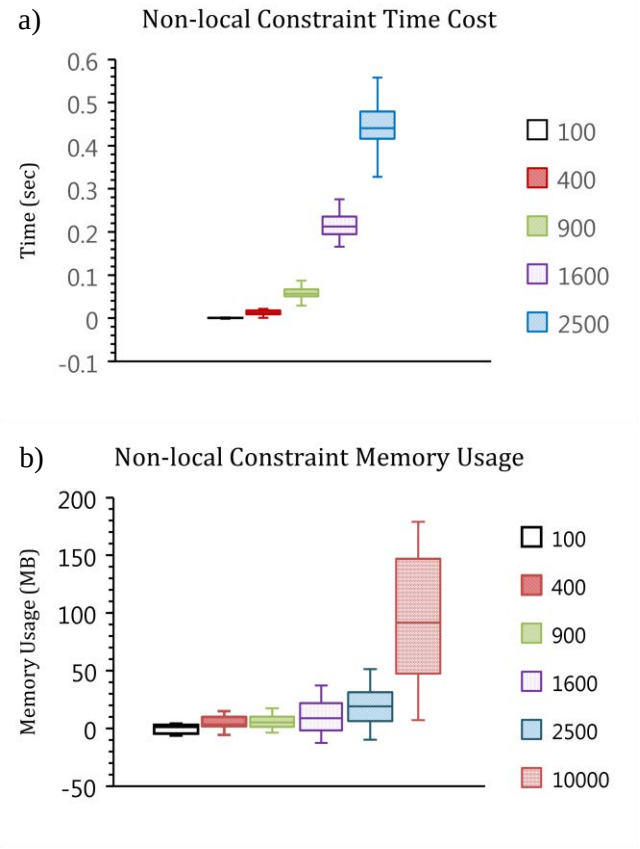

Figure 9. From left to right, each box plot corresponds to a map size within the legend in the order of top to bottom. The legend shows the total number of tiles within a map from 100 tiles (a 10x10 map) to either 2500 tiles (a 50x50 map) or 10k tiles (a 100x100 map). a) Time cost time for non-local constraints in seconds. b) Memory usage for non-local constraints in MB.

The reason there is an increase in runtime can be attributed to the extra steps taken to reason over the generative space. These extra steps, such as forced area observation and area propagation, generate additional observation/propagation loops which take times to execute. An example of such is the addition of items. When an item is placed and it has an associated rule than some sort of non-local checking done. This leads to an increased amount of observations done by the force function described in section 3.2.1. As a result, the rules will force additional observation/propagation loops that can contribute to the increase in time cost. As seen from Figure 9a, time cost for creating a map size of 2500 can be as much as 0.55 seconds while it is 1 ms and 15 ms for a map size of 100 and 400 tiles, respectively. This means, depending on the tolerability of lag, a technical artist may

want to evaluate the benefit of adding non-local constraints to WFC at map sizes that exceed 400 tiles.

The average memory usage increased from 1MB to about 20MB for map sizes of 100 tiles to 2500 tiles. The largest memory usage average is exhibited by a map size of 10k tiles (Figure 9b), which is about 90MB. From the results, it can be concluded that the number of tiles within a map has a direct relationship with memory usage. This makes sense because the area calculations that are triggered will increase memory usage and the number of area calculations increases as map sizes increase. As such, non-local constraint may be a good candidate to consider during both design time and runtime despite the increase in both runtime and memory usage.

4.3 Weight Recalculation

As mentioned before, entropy is the driving force in finding a solution within the generative space where the observation stage alters the entropy of each wave element. The addition of weight recalculation shifts the responsibility of calculating entropy onto itself where it alters the weight distribution of inputs as WFC iterates through the search space. This can lead to more unpredictable maps such that the change in weight can increase entropy which creates higher instability.

The input requirement for weight recalculation is passed by adding another parameter to pass into WFC. The parameter is nested under an object input with an attached rule. The most time intensive addition is weight recalculation. It exhibits a similar trend to non-local constraint where average time cost and standard deviation both increase exponentially with map size (Figure 10a). The increase is attributed to the fact that the entropy of the entire wave is recalculated each time a weight is updated. As a result, the runtime of up to N-1 tiles is added to the total execution time. The worst-case scenario is when the entire wave entropy is recalculated after each observation/propagation loop, which adds a runtime of (N-1)! tiles.

As for memory usage, the average change in memory ranges from 1MB to 3MB for map sizes 400 and below (Figure 10b). There appears to be an upward trend in memory usage with an increase in map sizes. From 100 to 2500 tiles, the average memory usage increased by almost 3000%. However, the relationship between map size and memory usage cannot be quantified with high precision because of the increase in the spread of values as map size increases. It is significant to note that the average memory usage can increase by about 30MB with a range of 3MB to almost 70MB for a 50x50 or 2500 tiles sized map. From the data collected, the pivotal point where memory usage change becomes significant is when map size is above 400 tiles. This means a technical artist needs to consider whether the benefit of weight recalculation is worth the cost of increased memory usage when the number of tiles exceeds 1600.

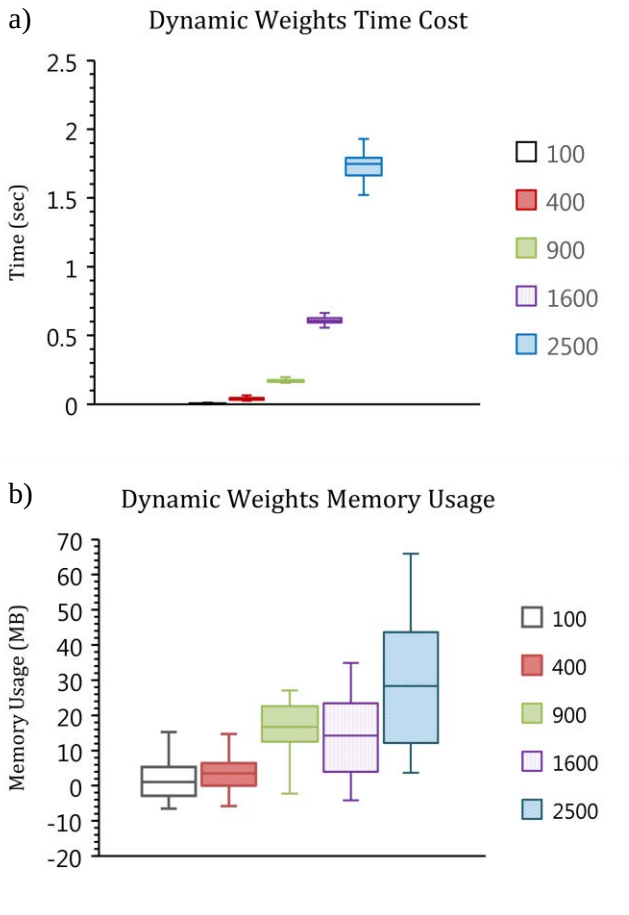

Figure 10: From left to right, each box plot corresponds to a map size within the legend in the order of top to bottom. The legend shows the total number of tiles within a map from 100 tiles (a 10x10 map) to 2500 tiles (a 50x50 map). a) Time cost time for weight recalculation in seconds. b) Memory usage for weight recalculation in MB.

4.4 Area Propagation

Area propagation is a modification that increased the contradiction rate. Instead of testing runtime and memory, this paper evaluated its conflict rate against the area of interest. The input tiles are separated into two categories and passed into WFC as labels for each tile. One of the tiles has a triggering effect that forces area propagation such that once it is observed then none of the tiles in the same category is banned from a defined area around the tile. The input parameters for area propagation are like that of non-local constraints but with an added type parameter that observation the category of tiles to keep. The defined area to propagate is calculated the same way as non-local area observation is calculated. The tested areas are 5x5, 7x7, and 9x9 tiles. These sizes were chosen because the smallest area around a

tile that does not include its immediate neighbors is 5x5, area size increases by two for both height and width simultaneously, and the test map size is 10x10. Other map sizes were not used because the rate of producing a map drops close to 0 maps/generation when map size is above 10x10. 50 runs were conducted for each propagation test area for 15 times. The resulting mean and standard deviation of successful map generations given in percentages for each test area is summarized in Table 2.

As can be seen, the area of propagation has no effect on the conflict rate. Regardless, the resulting conflict rate is around 60% for a map size of 100 tiles with the parameters and inputs defined previously. Further testing will need to be conducted to give a complete analysis of area propagation. However, it can be concluded that area propagation is best used as a constraint for design time rather than runtime due to the high conflict rate.

5 Conclusion / Future Works

This paper has shown that design constraints can be added to WFC by using non-local constraints, weight recalculation, and modifications of WFC's observation/propagation loop. This work was able to extend WFC past local constraints with the addition of nonlocal constraints like items, dependencies, area propagation, and weight recalculation. We have provided tests that show that most of these constraints are efficient enough to be used at runtime. Some have high conflict rates and would be better for integration within a mixed-initiative tool such as *Tanagra* or *Sentient Sketchbook* [19].

For future work, we plan to add more concepts such as architecture and models of mood or emotion for level generation. Next, we want to create a mixed-initiative authoring environment that utilizes WFC. Lastly, we plan to integrate this with the playable model of social interaction CiF [20] to encode contextual world knowledge. We are hopeful that with more design focused constraint spaces within WFC, CiF will be able to generate worlds that reflect the social state between characters.

REFERENCES

[1] G. Smith, "An Analog History of Procedural Content Generation," p. 6.

[2] G. Smith, "The Future of Procedural Content Generation in Games," p. 5.

[3] N. Shaker, A. Liapis, J. Togelius, R. Lopes, and R. Bidarra, "Constructive generation methods for dungeons and levels," in *Procedural Content Generation in Games*, Cham: Springer International Publishing, 2016, pp. 31–55.

[4] M. J. Nelson and A. M. Smith, "ASP with Applications to Mazes and Levels," in *Procedural Content Generation in Games*, Cham: Springer International Publishing, 2016, pp. 143–157.

[5] Y.-G. Cheong, M. O. Riedl, B.-C. Bae, and M. J. Nelson, "Planning with applications to quests and story," in *Procedural Content Generation in Games*, Cham: Springer International Publishing, 2016, pp. 123–141.

[6] D. Ashlock, S. Risi, and J. Togelius, "Representations for search-based methods," in *Procedural Content Generation in Games*, Cham: Springer International Publishing, 2016, pp. 159–179.

[7] ExUtumno, *https://github.com/mxgmn/WaveFunctionCollapse*. 2019.

[8] T. Challies, "https://www.challies.com/articles/no-mans-sky-and-10000-bowls-of-plain-oatmeal/," *Tim Challies*, 02-Sep-2016. .

[9] Derek Yu, *Spelunky*. Mossmouth, LLC, 2012.

[10] I. Karth and A. M. Smith, "WaveFunctionCollapse is constraint solving in the wild," in *Proceedings of the International Conference on the Foundations of Digital Games - FDG '17*, Hyannis, Massachusetts, 2017, pp. 1–10.

[11] *Bad North*. Plausible Concept, 2018.

[12] *Caves of Qud*. Freehold Games, 2019.

[13] "mxgmn / Basic3DWFC." [Online]. Available: https://bitbucket.org/mxgmn/basic3dwfc. [Accessed: 11-Jan-2019].

[14] M. O'Leary, *Oisín: Wave Function Collapse for poetry. Contribute to mewo2/oisin development by creating an account on GitHub*. 2019.

[15] I. Karth and A. M. Smith, "Addressing the Fundamental Tension of PCGML with Discriminative Learning," *ArXiv180904432 Cs Stat*, Sep. 2018.

[16] G. Smith, J. Whitehead, and M. Mateas, "Tanagra: Reactive Planning and Constraint Solving for Mixed-Initiative Level Design," *IEEE Trans. Comput. Intell. AI Games*, vol. 3, no. 3, pp. 201–215, Sep. 2011.

[17] A. Liapis, G. Smith, and N. Shaker, "Mixed-initiative content creation," in *Procedural Content Generation in Games*, Cham: Springer International Publishing, 2016, pp. 195–214.

[18] E. Butler, A. M. Smith, Y.-E. Liu, and Z. Popovic, "A mixed-initiative tool for designing level progressions in games," in *Proceedings of the 26th annual ACM symposium on User interface software and technology - UIST '13*, St. Andrews, Scotland, United Kingdom, 2013, pp. 377–386.

[19] A. Liapis, G. N. Yannakakis, and J. Togelius, "Sentient Sketchbook: Computer-Aided Game Level Authoring," p. 8.

[20] J. McCoy, M. Treanor, B. Samuel, A. A. Reed, M. Mateas, and N. Wardrip-Fruin, "Social Story Worlds With Comme il Faut," *IEEE Trans. Comput. Intell. AI Games*, vol. 6, no. 2, pp. 97–112, Jun. 2014.