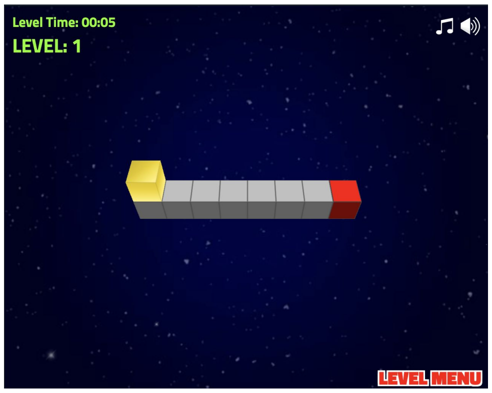
Pushing B-Cubed towards PSPACE-Completeness

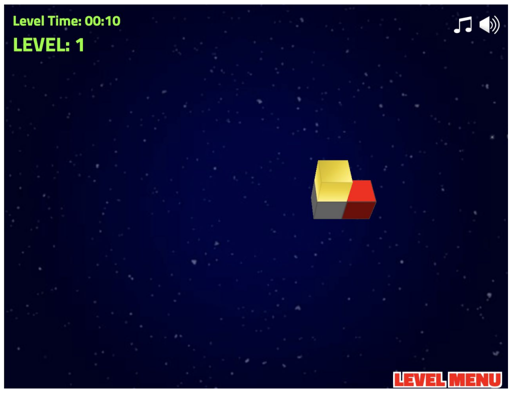
Maddie Burbage and Jared Berger

Abstract:
In this paper, we look to generalize the online puzzle B-Cubed, to push it from NP-completeness to likely being PSPACE-complete. While generic, grey-blocked B-Cubed levels could be easily proven to be NP-complete, we look to establish that by adding some simple modifications, B-Cubed could be made to have levels that take an exponential amount of time to solve with respect to the size of the level, which might suggest that this modified puzzle is in fact PSPACE-complete. Studying this puzzle may have larger implications for the potential exploration of an interesting modified Hamiltonian Path (or Hamiltonian Walk) problem in which each node in a graph has some frequency, f, and must be visited exactly that number of times by a walk, a problem which seems to be underexplored.

Introduction:

B-Cubed is an online block puzzle game. Hosted by the website Cool Math Games, the game is available to play for free online [1]. Levels appear as maps of grey (and potentially other specially colored) blocks, in addition to a red block representing the “finish” block. Controlling a yellow, moveable block which rests upon the map, the user clears a level by navigating from their starting position to the red “finish” block, while passing over each regular block in the map exactly once. In the images taken from level one below, we see that the player completes the level by moving their block seven spaces to the right. In the game, each time the yellow block leaves a block, that block disappears. In the right image below, we can see how the leftmost blocks disappear as the user travels from left to right.

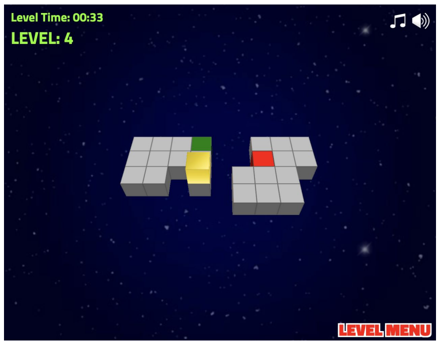


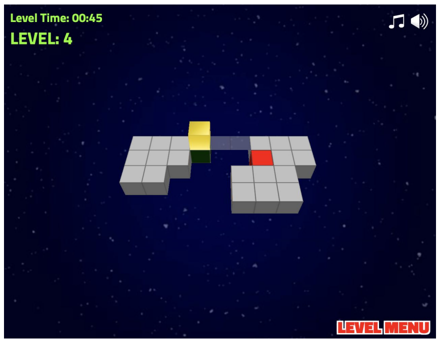


As it exists normally, it is quite straightforward to prove that B-Cubed levels containing regular grey blocks are NP-hard through a reduction from the Grid Graph Hamiltonian Path problem, which we know to be NP-complete. Andersson [2] takes a similar approach in order to establish the NP hardness of the puzzle Hashiwokakero, by reduction from Grid Graph Hamiltonian Circuit, but our proof would be simpler, as B-Cubed already closely mirrors the Grid Graph Hamiltonian Path problem.

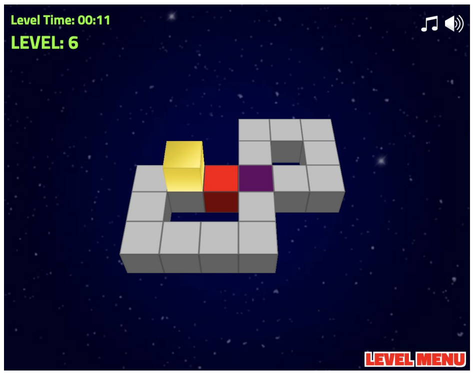
However, following the work of Greenblatt et. al [3] and Gabor and Williams [4], we look to push B-Cubed to potentially being PSPACE-hard, through the creation of levels that take an exponential amount of time to solve with regards to the size of the level. In order to do this, we look to the special blocks revealed in later B-Cubed levels.

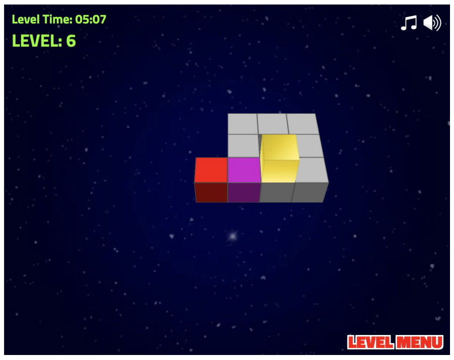
Special Blocks:
As the user progresses through the levels provided in B-Cubed, they begin to see special blocks appear in the map, each of which provides an additional feature. Two such blocks that we focus on are green blocks and purple blocks. Green blocks, when passed over, cause a group of other connected hidden blocks in the map to appear. We call this new group of blocks a bridge. Like grey blocks, bridges must be passed over exactly once to be cleared. The images below show level four, before and after having passed over the green block. Like the grey blocks, the green block disappears after being passed over exactly once. As we can see, stepping on the green block allows the user to reach the right side of the map, so that they can clear all the blocks and reach the red finish.



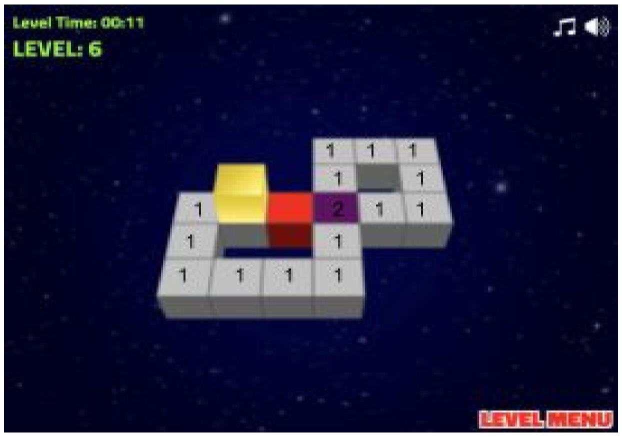


Next, we have purple blocks. Purple blocks need to be passed over exactly twice to be cleared, unlike the grey blocks which only must be passed over just once. As we can see below in the image from level six, a solution which clears all blocks requires stepping on the purple block twice. After being passed over once, the purple block lightens in color, and it then must be passed over one more time in order to be cleared.





Following the implications of the purple blocks, we generalize B-Cubed to B-Cubed with Frequencies, where every block now has a positive integer on it representing how many times the block may be passed over before it disappears. Each time a block is passed over, its number decreases by 1, until the whole block disappears if the number would have become 0. Below, we can see B-Cubed level six again, but this time with frequencies added to each block. Because the grey blocks only need to be passed over once to be cleared, they each have a frequency of 1, but because the purple block needs to be passed over twice, it has a frequency of 2. The red “finish” block does not require any frequency because to complete a level it must be stepped on a single time, once all the other blocks have been cleared. While purple blocks represent frequencies of 2, in our generalization, every block can have a frequency of any positive integer, meaning it must be passed over that number of times to be cleared. By introducing this additional complexity, our goal is for the puzzle to become PSPACE-complete.

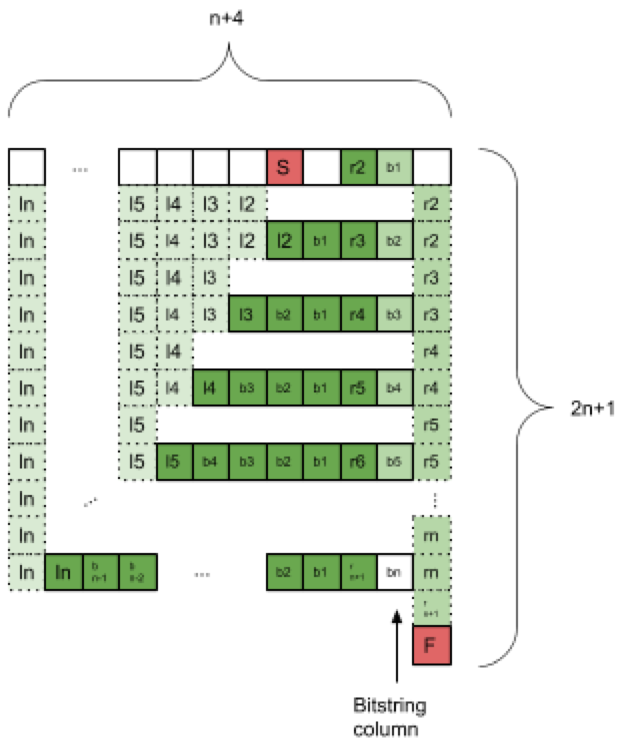


Our generalization also allows for green blocks to be used, and our rule for the generated bridges is that blocks in a bridge appear with a frequency of 1. A green block also has a frequency, so passing over it multiple times in a row, without passing over any part of the bridge in between, will have no effect except to decrease the remaining frequency on the green block. However, if the bridge has been cleared, stepping on its associated green block again will make the bridge blocks reappear.

Exponential Levels:
Constructing levels whose shortest solutions are exponentially long suggests that a puzzle may be PSPACE-complete [3]. Using these generalizations of green and purple blocks, we can produce a set of levels with an exponentially long solution for any n, where our level will simulate the production of all binary numbers in lexicographic order for a string n bits long.

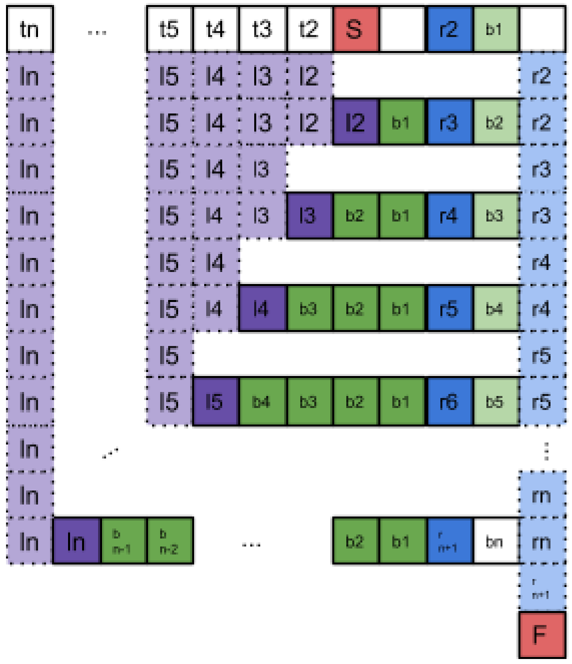
If we are generating binary strings of n bits, our levels (as shown on the diagram below) include (n+4) x (2n+1) blocks, which is $O(n^2)$ and polynomial in n. Further, each block has frequency of at most 2^{n-1}, if n is again the number of bits we are generating. However, we then need $\log(2^{n-1}) = O(n)$ bits to store each frequency, giving us an overall total of $O(n^2)$ blocks, each with a frequency which takes O(n) bits to store resulting in us storing a total amount of information that is polynomial in n. The final piece of information we must store is which blocks are special, and which bridges each green block may create. Only a constant amount of information is needed to identify the green, start, and finish blocks, keeping the total information polynomial in n. To track which bridges are controlled by each green block, a starting block of the bridge, direction of the bridge, and bridge length can be stored, so this information is also polynomial in n. Since there are at most n^2 green blocks, the total information needed for special blocks is also polynomial in n. Therefore, creating a level that minimally requires counting in binary from 0 to 2^n will require an exponential amount of time to solve, in regards to the size of the level.

This diagram shows the general construction of a level, without showing any frequencies yet. The player begins in the red S square (we have chosen to label it red for convenience), and ends in the red F square. White squares are regular blocks, each having a set frequency. Dark green squares turn on the light green bridges with the same labels. Dotted-outline bridges do not initially exist, but bridges with solid outlines exist at the start of the game.

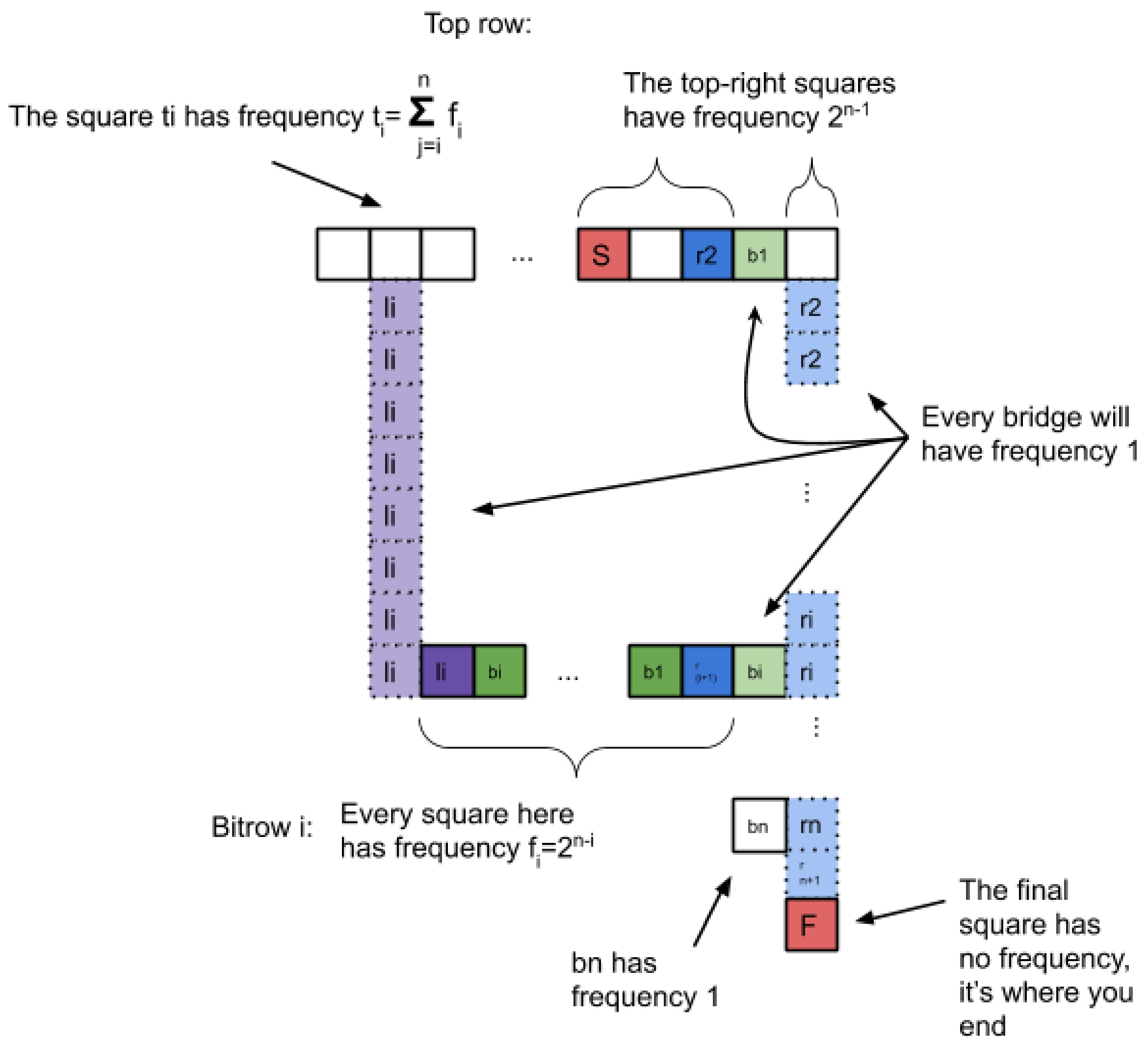


The three types of bridges are r-bridges, which form on the rightmost side of the board, b-bridges, which connect each row to the rightmost column, and l-bridges, which connect lower rows to the top row. The added benefit of the b-bridges is that when their column is read from bottom to top, n to 1, they represent a bit string that will be incremented in lexicographic order as the player solves the level. For each row, we understand the corresponding bit as being 0 if the block in the “bitstring column” in that row currently **does** exist and a 1 if the bridge in that row **does not** currently exist. We note this state every time a block in the “bitstring column” appears or is cleared. One note is that the square bn (at the bottom of the bitstring column) is not actually a bridge, because it does not need to be turned on multiple times, but it still should be read as part of the bitstring.

If we label squares in the top row with the number of the bridge that they are above, we can calculate frequencies for every square. Here are the labels given to every square, and the bridge type is color-coded:



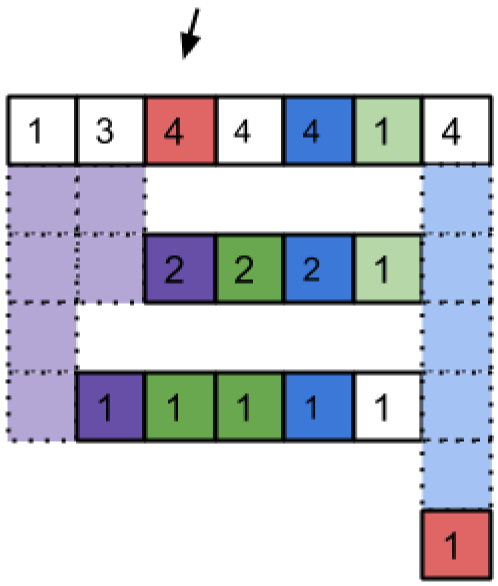
Now we can calculate the frequencies of every square, based on what type of block it is:



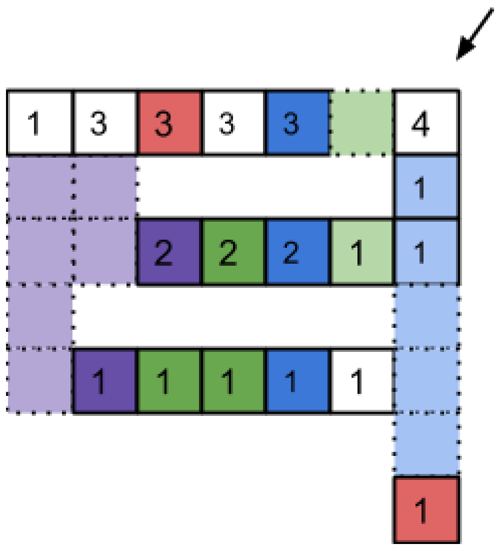
Top row:
The square ti has frequency $t_i = \sum_{j=i}^{n} f_i$
The top-right squares have frequency 2^{n-1}
S
r2
b1
r2
r2
li
li
li
li
li
li
li
li
li
bi
b1
$r_{(i+1)}$
bi
ri
ri
Every bridge will have frequency 1
Bitrow i:
Every square here has frequency $f_i = 2^{n-i}$
bn
rn
r_{n+1}
F
bn has frequency 1
The final square has no frequency, it's where you end

We can also show an example level, with n=3:
(note: dotted lines implies a block is not currently present)

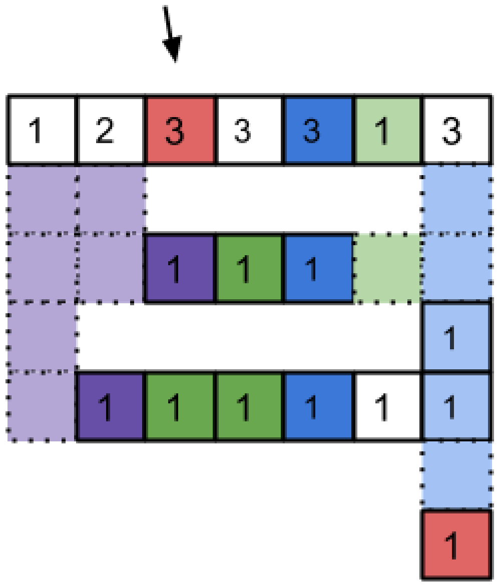
The starting diagram, at state 000:



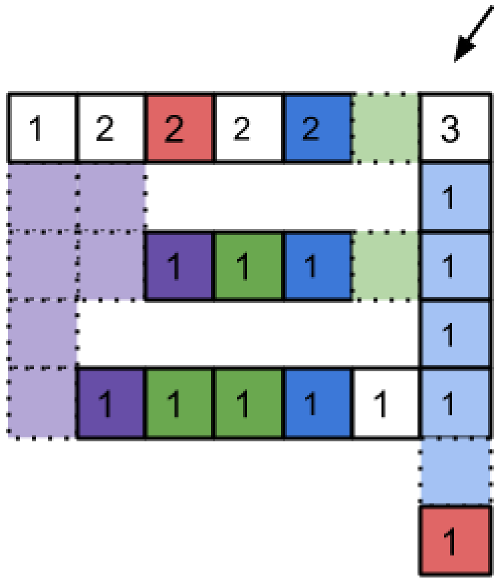
State 001, after walking right through row 1:



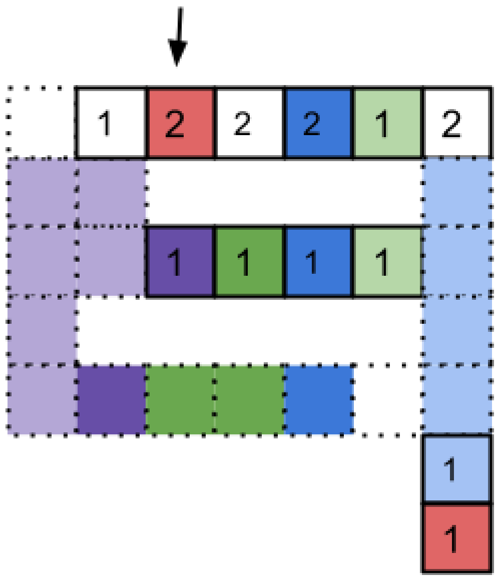
State 010, after passing left through row 2:



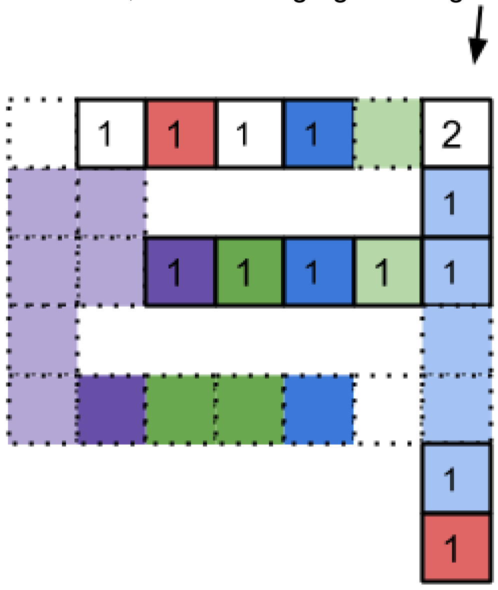
State 011, after passing right through row 1:



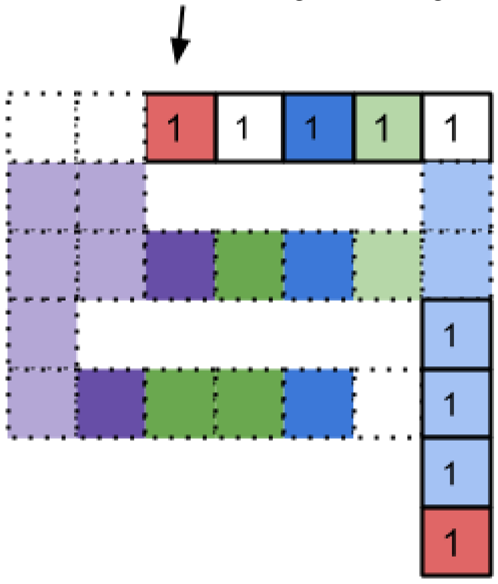
State 100, after walking left through row 3:



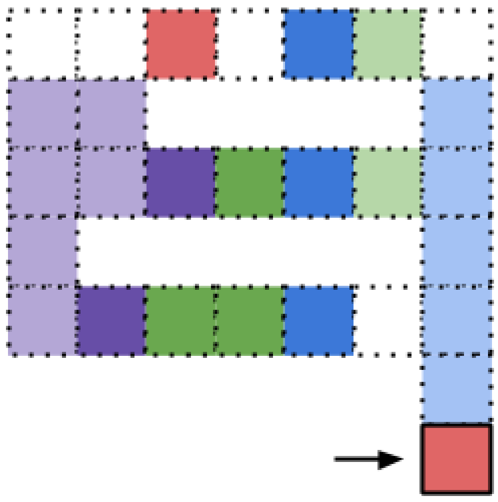
State 101, after walking right through row 1:



State 110, after walking left through row 2:



State 111, when finishing the puzzle:



Exponential Levels:

Lexicographic Order:
We created these levels to require an exponential amount of moves to traverse, and this is ensured by the fact that our levels generate all 2^n strings for the input value, n. These strings are counted in lexicographic order, and are represented by the b-bridges of each row.

To count all binary numbers for a string of length n in lexicographic order, each string can be created from the last by following a successor rule. If a string has suffix 01^{i-1}, it will be changed to 10^{i-1}, but if no such suffix exists, the rightmost bit of the string will be changed from 0 to 1. Thus we can consider two active bits, either the bit at position i from the right, or the bit at position 1. In constructing the lexicographic sequence, we'll alternate between these active bits, always changing the suffix starting with the active bit. Then, starting from the string 0^n and ending at 1^n, all 2^n strings will be produced.

Lexicographic Levels:
We will now prove that our exponential levels must be solved in a way that causes our binary string column to create all strings in lexicographic order. We'll first prove that a solution exists where the player always modifies the bitstring by changing the suffix starting at the current active bit.

Here we prove that every time the active bit is the least-significant bit, the player can change it. The player begins our level at the top row and must alternate between the top and lower rows, by the design of the bridges between rows. This means that every alternate change to the bitstring only affects the least-significant bit. Because the player turns off the b1 bridge every time they exit the top row, they are setting the least-significant bit to 1. Every lower row has a green block that creates bridge b1, so leaving any lower row will change bit b1 to be 0 again. Since these blocks are the only way to make the least-significant bit a 0, and the player alternates between lower rows and the top row, the player always changes the least-significant bit to a 1 immediately after it becomes a 0. At the start of the puzzle, this bit is also a 0, and the player's first change to the bitstring is to cross bridge b1 and make it a 1. So every time the bitstring has a final bit of 0, meaning it cannot have a suffix of 01^i, and the active bit would be bit 1, the player is also only changing bit 1. Therefore the player always follows this active bit for lexicographic order.

We can also prove that we follow the changes of the other active bit for lexicographic order. To access a lower row, i, bridge bi must be on, and every r-bridge from r2 to ri must be on. These r-bridges can be turned on when the previous row has been traversed, and must be turned off when the corresponding row is accessed. So the only way for them to all be on is if they were turned on in reverse order, and the player traversed the rows from ri to r2. This order also ensures that the b-bridges to rows 2 through (i-1) are off, because they can only turn back on by entering a lower row after entering a higher row, and we entered these rows from lowest to highest. After exiting row 1 to walk to row i, bridge b1 will also be off, so if bridge bi is on, the suffix of the string is 01^{i-1} at this point. Once row i is traversed, and bridge bi is off, we also change all higher bits to be 0 by turning on their bridges. This creates the suffix 10^{i-1}, from suffix 01^{i-1}. Whenever we enter a row from 2 to n, this suffix change will happen, and whenever the suffix is 10^{i-1}, we are allowed to enter row i. When the string has these suffixes, bit i is the active bit in lexicographic order, and because we are always able to change the bitstring in accordance with this active bit, we are adhering to lexicographic order every alternating change. Because we change the suffixes of both active bits in the correct way every time we are supposed to, our bitstring will be produced following lexicographic order.

Further, following the ruler sequence to count lexicographically from 0 to 2^n corresponds exactly to stepping on every block its proper frequency number of times. If we follow the ruler sequence to generate all n-bit binary strings, then for each bit i, we set that bit equal to 1 exactly 2^{n-i} times. Every time we set a bit equal to 1 we turn off that bit's bridge and travel across that row. In our levels, the frequency of any non-bridge blocks in row i (besides the top row) is exactly 2^{n-i}. For the top row, the frequency of all non-bridge blocks that aren't above an l-bridge are also 2^{n-1}, which follows the same rule. Therefore, by following lexicographic order to generate all 2^n binary strings, we can simultaneously pass over those top-row blocks and each block in our lower rows exactly the correct number of times.

We also pass over the other non-bridge blocks on the top row exactly the right number of times. These blocks are above an l-bridge from a lower row. Because we pass through each row i, from 2 to n, exactly 2^{n-i} times, its l-bridge appears that many times. For a player solving the game in lexicographic order, they would move from the l-bridge up to the top row, and immediately walk to the right to cross the first bridge again. So blocks above the l-bridges are passed through as many times as the block to their left and the block below them are passed through. For the block above the ith l-bridge, this is the sum of the times a player traverses a row from i to n, or $\sum_{k=i}^{n} 2^{n-k}$ However, our level has been constructed so each block above an l-bridge has exactly that frequency.

Finally, we know we set each bridge exactly the right number of times. L-bridges are set by blocks on the same row as they exit from, so you can exit a row whenever you traverse it, and there will be no l-bridges remaining when you are finished with a row. The r-bridge for row i is set by a block in the previous row, and because that row is traversed $2^{n-(i-1)}$ times, that bridge will be set that many times. That r-bridge will be used once every time the player traverses every row from i to n, which is $\sum_{k=i}^{n} 2^{n-k}$. This is the sum $2^0+2^1+...+2^{n-i}$, which equals $2^{n-(i-1)}-1$, but the bridge will be crossed one last time as the player goes to the finish, so the player will use that bridge exactly $2^{n-(i-1)}$ times. Finally, the ith b-bridge is set at the beginning, and reset every time a lower row is traversed. The number of times a lower row is traversed is $\sum_{k=i+1}^{n} 2^{n-k}$, and this sum equals $2^{n-i}-1$, so the ith b-bridge is present 2^{n-i} times, which is exactly how many times the player will traverse that row.

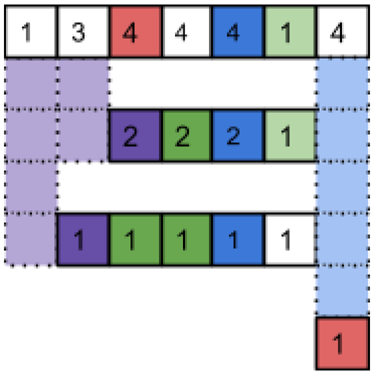
Finally, after we end up at state 1^n (11111…..), we know we will have just travelled across the top row from left to right, because our last step is to switch from 1^{n-1} 0 to 1^n. At this point, because we're following lexicographic order, we would be ready to set a new bit, bit n+1, from 0 to 1. However, in this case, we don't have another bit to set—just our finish block to reach. Because our finish block is located in the same place as the start of our newest bit row would be, we know that every r-bridge must be present, whose blocks all have frequencies of 1. Furthermore, every other block will be absent, because we proved already that their frequencies are only enough to change the bitstring's state to 1^n by lexicographic order. Therefore, we can travel from the top row along a path of blocks with frequency 1 to our finish block, and reach our finish block having cleared all blocks in the puzzle.

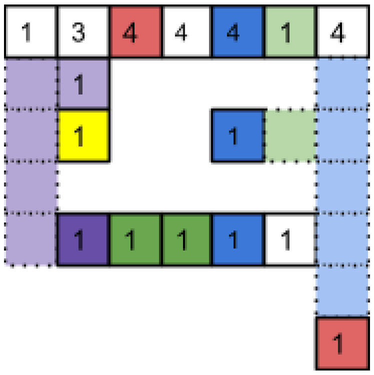
Why Lexicographic Order is the Only Solution:

We have shown that a solution exists, which lexicographically generates every binary string of n bits. This is an exponential solution in regard to the size of the level, since the level progresses through 2^n different states. Now, we will show that this described solution is the only possible solution to our constructed B-Cubed level. We will prove this by demonstrating that any divergence from our solution explained above wastes finite resources, which are the block frequencies.

In our levels, in order to reach the finish block, the level has to reach the state of being all 1's. Further, as we proved above, the frequencies on our blocks correspond to the number of moves required to follow the ruler sequence exactly. We know the player only ever has the choice of remaining in a row or moving to one other row, because in lower rows the player can only go back to the top, and in the top row, the only lower row available is dependent on the combination of open b and r bridges. This stems from the state of the bitstring column, with the only allowed row being row i for the suffix 01^{i-1}. So the only way for them to diverge from our lexicographic solution without immediately getting stuck by traveling back and forth within rows. But, we also know that because the frequencies in each row correspond exactly to how many times we must traverse that row completely when following the ruler sequence, divergence or wandering within rows uses up frequencies which we will need later. These frequencies work so that each time the user sets a bit, they must traverse the row completely from right to left—or left to right in the case of our least significant bit row. If at any point, the user wanders or takes an extra turn, they will be using up frequencies of blocks. If this happens, a situation will necessarily arise where the user runs out of blocks or gets stuck.

Consider again our example of a level which contains three bitrows (see the figure below, which demonstrates a level before and after a divergence from our solution). Imagine a user is in row 2, standing on the bridge b2. Instead of following our solution and traveling just from right to left, they turn around when they get to the purple 2, going back to the green 2, and then continuing, proceeding as usual. After performing these moves, the user will end up on the yellow highlighted block. This leaves a hanging blue 1 block in the second row, which is essentially a dead end. We know we need to pass over this block to clear the level, but we have an additional "dead end" block with frequency in our level 1 which is our red finish block. We know we cannot reach both of these blocks without getting stuck, so therefore we no longer have any possible way to clear the level. While in this situation, the inability to solve the level is clear right away because row two had blocks with frequency 2, any similar divergent move would eventually result in some blocks in a row being cleared too early, resulting in us either getting stuck or unable to reach our finish block.

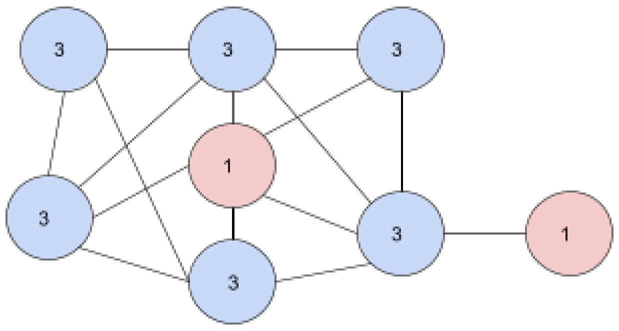


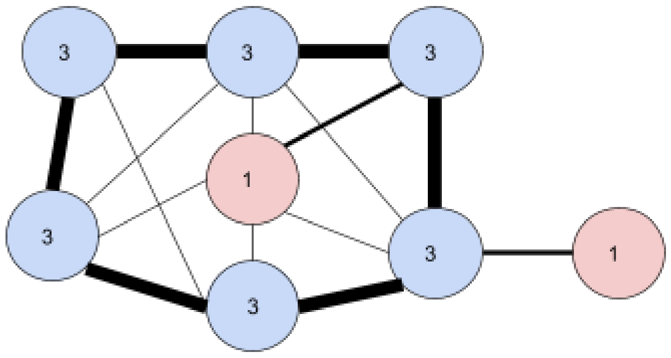


Future Work:

Specific to the B-Cubed puzzle, one potential extension is to consider exponential levels that contain fewer of the special green blocks. In our constructions, we relied upon bridges to force players to traverse the rows representing bits greater than the least significant bit from right to left, and in turn to traverse in our levels in clockwise loops. However, it could be interesting to consider levels that rely less on the green blocks to determine whether there exist levels with fewer bridges that are still able to constrain a player to progressing through all 2^n binary states.

Further, studying this generalization of B-Cubed points us towards an interesting, more general graph theoretical problem. What if we considered a modified Hamiltonian path—or more accurately, Hamiltonian walk—problem such that instead of a walk needing to visit each vertex in a graph exactly once, it needs to visit each vertex some frequency f number of times? For example, consider the sample graphs below. In these graphs, we could think of a potential modified Hamitonian walk as one which visits the red nodes each exactly once and the blue nodes each exactly three times. This specific example happens to have an intuitive solution, shown by the right image, where we could start at the rightmost red node, travel to a blue node, visit all blue nodes three times (as shown by the thick edges), then travel to the remaining red node. However, outside the context of B-Cubed and its special tiles, further research could explore whether adding frequencies to nodes is enough to push this version of Hamiltonian walk in general to be PSPACE-complete.





References:

[1] https://www.coolmathgames.com/0-b-cubed

[2] D. Andersson. Hashiwokakero is NP-complete, Information Processing Letters, Volume 109, Issue 19, 2009, Pages 1145-1146, ISSN 0020-0190, https://doi.org/10.1016/j.ipl.2009.07.017.

[3] A. Greenblatt, J. Kopinsky, B. North, M. Tyrrell, and A. Williams, MazezaM Levels with Exponentially Long Solutions, 20th Japan Conference on Discrete and Computational Geometry, Graphs, and Games (JCDCGGG 2017), 2017.

[4] J. Gabor and A. Williams, Switches Are PSPACE Complete, 30th Canadian Conference on Computational Geometry (CCCG 2018), 2018.