Puzzle Poetry

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

Daniel Patel
Spring 2020.

Technical Project Team Member
Kaan Katircioglu

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Signature _**Daniel Patel**_____ Date _4/27/2020_
Daniel Patel

Approved _____ Date _____
Nathan Brunelle, Department of Computer Science

INTRODUCTION: TURNING POEMS INTO PUZZLES

Research Objective: Partition Poems into Polyominoes

A Sonnet is a type of poem in iambic pentameter consisting of fourteen lines split into three quartets followed by a couplet. Sometimes the first two quartets are referred to as an octet and the final quartet and couplet as a sectet. Brad Pasanek, a professor of English at the University of Virginia came up with the idea of turning these sonnets into puzzles by splitting them up into tiles of different sizes. Specifically, Professor Pasanek was looking to split the sonnet into the octet and sectet and tile each separately before recombining. The octet would be partitioned into one of each type of pentomino and tetromino where pentominoes are polyominoes of size and tetrominoes are polyominoes of size four, while the sectet could be tiled by one of each pentomino. An added stipulation to the tiling was that each individual word could not be split across multiple polyominoes.

Initial Approach: Brute Force

When attempting to accomplish this task, Professor Pasanek initially found a list of all possible solutions for tiling a ten by six grid with pentominoes, and wrote a program to use this list and individually go through solutions to check if any fit with the word constraints. He however could not find a similar list of solutions for the ten by eight grid with pentominoes and tetrominoes likely due in part to the number of these tilings being in the billions (Pasanek, 2019). (Alloy, 2019)Because of this, he attempted to write a program to list each of these solutions so that he could then check these solutions against word patterns of individual sonnets he wished to tile. The algorithm written to list the solutions was written in R and attempted to check every possible combination of every orientation of each class of tile at each possible location. The program written, when run on the UVA computing cluster, was able to generate around 12 Million out of the billions of solutions in four days, meaning that it would have to be orders of magnitude faster for it to complete in a reasonable amount of time.

Prior Research: Reduction to Exact Cover

Last year a group of students attempted to tackle this problem via various reductions to NP Complete problems. Their most successful attempt was a reduction to exact cover. Specifically, the students found an implementation of Algorithm X, a famous algorithm written by Donald Knuth for solving the exact cover problem. They used this implementation as a black box and focused on transforming the given input into the input required by the implementation and then transforming the output of the algorithm into the format they desired. For the sectet, their solution ran Algorithm X in a trivial amount of time but the generation of viable subsets needed in preprocessing was estimated to take anywhere from seven thousand days on the low end to 10^{64} days on the high end depending on if the model used was polynomial or exponential. Because of this, more students were asked this semester to attempt to crack the problem of tiling sonnets (Alloy, 2019).

(Smith)APPROACHING THE PROBLEM USING A BRANCH AND BOUND TECHNIQUE

Initial Focus: Solving the Sectet in Reasonable Time

When first approaching this problem my team decided to focus on developing an algorithm that could tile the ten by six sectet at the bottom of a sonnet in a way that was somewhat scalable in a reasonable amount of time, and then, if we were able to accomplish this, focus on scaling the solution to work for the octet. To accomplish this task, we chose to employ something following the Branch and Bound (BnB) design paradigm. BnB is represents the whole search space as a tree with the root being the starting location for the algorithm and the leaves being possible solutions. The algorithm recursively traverses the branches of the tree, and uses specified optimizations to stop looking at branches that no longer can contain viable solutions, a process called bounding. The team then coupled this paradigm with a conversion of all possible values to long integers so that any bounding checks could be done with bit fiddling. Bit fiddling is using the simplest instructions in a computer's instruction set such as bitwise "and" and "or" to accomplish the necessary operations in a minimal number of clock cycles. In doing this, runtime can be minimized as many more complicated instructions can take hundreds or even thousands of clock cycles compared to the handful used by these instructions. I wrote my implementation using Java while my teammate attempted to do the same in Python.

Data Representation: Specific Details

To represent a poem, the lines were broken by hand into syllables which were then assigned a character based on which word they were a part of. This list of syllables was used to generate the list of words used as constraints for the algorithm. An example of this preprocessing that had to be done manually is given in Figure 1.

```
Pale Sorrow's victims wert thou once among,          A B B C C D E F G G
Tho' now releas'd in woodlands wild to rove?         H I J J K L L M N O
Say---hast thou felt from friends some cruel wrong,  P Q Q R S T U V W X
Or diedst thou---martyr of disastrous love?          Y Z [ \ \ ] ^ ^ ^ _
Ah! songstress sad! that such my lot might be,       ` a a b c d e f g h
To sigh and sing at liberty---like thee!             i j k l m n n n o p
```

Figure 1: The sectet of Charlotte Smith's "Sonnet III. To a Nightingale," and the its representation in the format used as input for the program (Smith).

As shown in Figure 2, the ten by six board to be tiled can be represented in the lower sixty bits of a long integer. Similarly, each word in the poem and each pentomino that could be used to tile the poem can be represented as an unsigned long in this way. The solution output by the algorithm happened when the board was set to all 1s and the and the format was twelve long integers each representing a pentomino from a separate class in a position and orientation such that none overlapped or split words.

59	58	57	56	55	54	53	52	51	50
49	48	47	46	45	44	43	42	41	40
39	38	37	36	35	34	33	32	31	30
29	28	27	26	25	24	23	22	21	20
19	18	17	16	15	14	13	12	11	10
9	8	7	6	5	4	3	2	1	0

In base 2
0000010000000011100000000000000000000000000000000000000000000000

In hex
0x0403804000000000

Figure 2: A representation of a board with the X pentomino placed in the top left corner in both binary and hexadecimal

First Attempt: Choosing Orientation Before Position

The first version of the algorithm composed by the team was one that took all the possible orientations of the pentominoes, sixty-three split into twelve groups, and selected one from each of these before trying to determine if any possible set of translations for these tiles would result in the tiles fully covering the board without overlapping and without violating the word boundaries. On the sectet shown in figure 1, this appeared like it would take four days to go through the full search space and, because of this, was never run to completion.

Second Attempt: Choosing Translations Before Selecting Orientation

When the first attempt was found to be to slow, it was hypothesized that choosing a translation for a for a class of pentominoes before selecting the orientation from that set would lead to a better runtime. Because of this, the algorithm was reworked to pass the full set of sixty-three pentominoes through the recursive function to only select from each class after a translation had been chosen. This produced marginally better results but still remained far too slow to warrant further optimizations.

Third Attempt: Selecting from Sets of Words

When the prior attempt as found to be too slow, the team decided to scrap the majority of the algorithm and look into both finding a new way to represent the search space as well as look into different design patterns. One possible answer looked into was switching to a Dancing Links based implementation of exact cover, but this proved to be slower than BnB so it was not pursued further. The other path investigated was that of generating the tiles to be placed by forming them from sets of adjacent words whose syllable count totaled to the size of the tiles to be placed. These sets of words were then sorted into categories based on which class of pentomino they belonged to. From here the BnB algorithm could just take a tile from each set

and see if any covered the same syllable as the words, translations, and orientations were already accounted for in their generation. While this attempt proved to be fairly slow in practice due to inefficiency in generating the sets of words, it gave useful information about what work could be done before running BnB, and therefore not repeated numerous times throughout the search.

Final Solution: Pre-Generating Viable Tiles

When the generation of sets of words proved to be too slow, it was discovered that an equivalent output could be generated by going through each orientation of each class of pentomino and saving every possible translation where it did not cross a word boundary. These lists could then be used with the second half of the previous algorithm to tile the sectet. With this method, the algorithm was able to tile the poem in figure 1 in about three seconds and could find all tilings of a ten by six board in about four hours. With this success, the team turned to optimizing, and found that two useful optimizations were looking at the contiguous open spaces in a board and checking if their area was a multiple of tile size and, for the first few tiles, removing all tiles from the lists yet to be selected from that covered some of the same area as the placed tile making these checks run fewer times overall. With these optimizations, the algorithm was able to find both possible solutions for the sectet shown in figure 1 in a quarter of a second, and all possible tilings of the six by ten board in forty-three seconds.

DELIVERABLE AND FUTURE WORK

Deliverable: Creating a Visualization of Algorithm Output

By the time the optimizations detailed above were completed it was determined there was no longer enough time in the semester to expand the algorithm to work with octets due primarily to the fact that octets, by having eighty syllables, could not be completely represented by a long integer in the same way a sectet could since long integers only contain 64 bits. Because of this, everything would have to be refactored with a different method for representing the information such as Java's BigInteger Class. Due to this fact, the team instead focused on developing a tool to take a poem in the form of the input specified above, run the algorithm to find a set of tiles that worked for partitioning the poem, and then creating a graphical representation of these tiles. This representation was generated using the SDL2 graphical library in C++ an example output of this program is shown in figure 3.

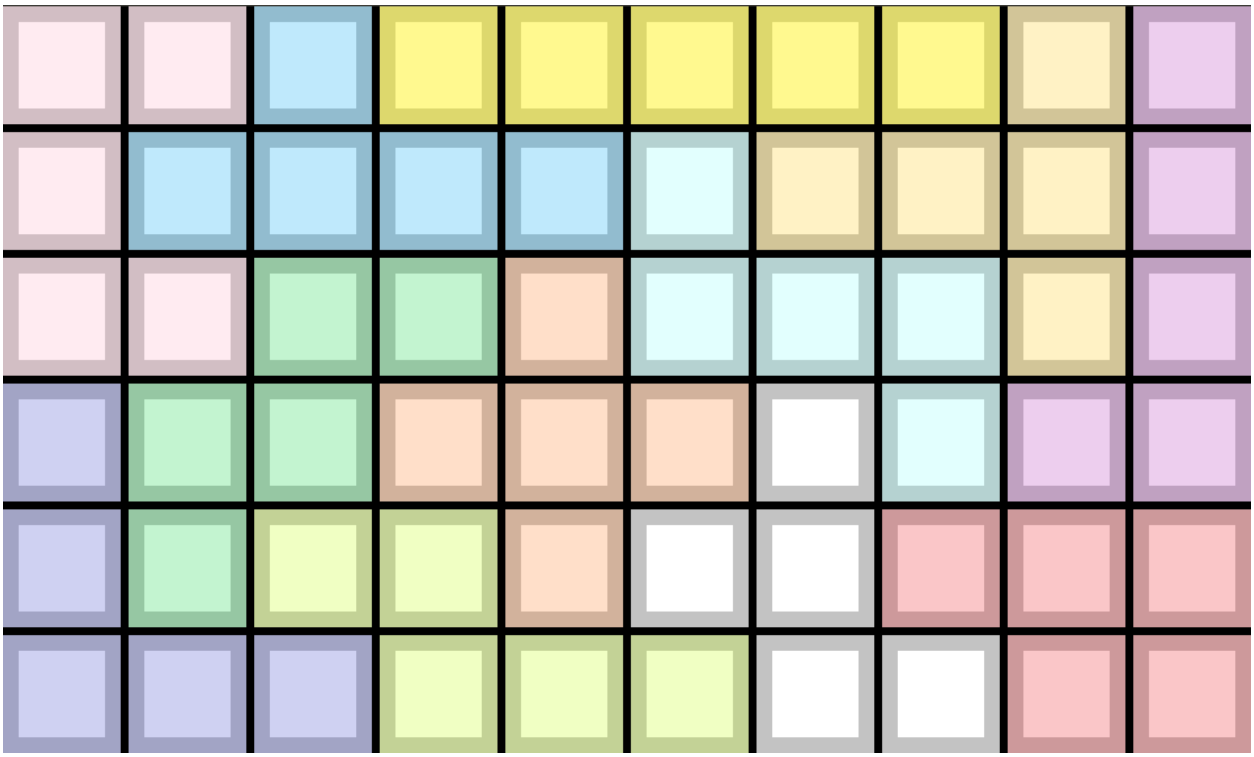

Figure 3: The first output of the program when running with the sectet of Charlotte Smith's "Sonnet III. To a Nightingale," as input.

Future Work: Expanding to Octet and NP Completeness

Given the success of the program on sectet, it seems like the algorithm is very promising for being able to tile an octet, once refactored, in some manageable amount of time if run on the UVA computing cluster. Because of this, a possible path for future research is expanding the algorithm created in this research to work with the octet, and possibly adding further optimizations to save on compute time. Another avenue for future research from this project is determining if the problem is actually NP Completely, like is assumed, of if it falls into some other category of difficulty. This would likely have to be done by finding a reduction both to and from other NP Complete problems.

References

Alloy, J. S. (2019, May 4). DEVELOPING AN ALGORITHM FOR THE TILING OF SHAKESPEAREAN SONNETS. Charlottesville, Virginia, United States.

Pasanek, B. (2019). *puzzlepoesis*. Retrieved from bpasanek.github.io: https://bpasanek.github.io/puzzlepoesis/pieces.html

Smith, C. (n.d.). *Sonnet III: To a Nightengale.*