Implementing algorithm for efficient spatial optimization:

How can the implementation of a computer application algorithm addressing strategic placement in both two-dimensional and three-dimensional, contribute to efficient spatial optimization?

Cristian Nițu
Dimitrios Tsiplakis
Gregorio Amah-Tchoutchoui
Jure Kastelic
Martin Petrov
Mila Spasova

The Department of Advanced Computing Sciences
Maastricht University
Maastricht, The Netherlands

I. ABSTRACT

Abstract—This project aims to address complex “packing” problems, specifically the three-dimensional knapsack problem, using the concept of pentominoes. This project focuses on the development of a computer application for solving the three-dimensional knapsack problem with pentominoes, planar structures formed by attaching five squares of size 1x1 to each other. The application is implemented in Java and consists of three main phases. In the first phase, the project addresses the problem of finding if a given set of pentominoes can completely cover a predetermined size rectangle. The objective is to efficiently arrange pentominoes to maximize space utilization. The implementation of brute force and pruning ensured successful arrangements of pentominoes. The second phase introduces a Tetris-style game utilizing the 12 pentomino shapes. Our game features a functional GUI, a branching algorithm for optimal pentomino ordering, and a strategic bot capable of making moves within the game. Finally, in the third phase the efficient implementation of the modified greedy permitted us to solve three-dimensional knapsack problems, considering parcel shapes, sizes, and values. The findings of this study confirm the effectiveness of our algorithmic approaches in addressing complex “packing” problems such as pentomino arrangement, Tetris gameplay, and three-dimensional knapsack problems.

II. INTRODUCTION

While prior scientific research sheds light on certain spatial optimization techniques, the state-of-the-art in addressing the strategic placement of pentominoes in both two-dimensional and three-dimensional spaces remains an open area for exploration. Prior research in computational optimization [1,2] has primarily focused on combinatorial problems, including variations of the knapsack problem, but the specific challenges posed by the unique shapes of pentominoes constitute an unexplored research area in prior scientific reports.

To tackle this problem, our project employs a multi-phases approach. All phases uniquely contribute to our goal of enhancing space optimization comprehension. Our algorithms draw inspiration from previous approaches, such as the greedy and pruning algorithm. Although, we modified them to better address our spatial optimization arrangement challenges. The choice of the greedy and pruning algorithm is motivated by their proven efficiency in combinatorial problems[3,4].

Achieving optimal spatial utilization through strategic placement has wide-ranging applications, impacting various industries such as shipping or air transport, for example. In the context of shipping, achieving optimal spatial utilization ensures that cargo containers are loaded most effectively onto ships. This not only maximizes the quantity of goods that can be transported in a single shipment but also contributes to fuel efficiency by minimizing wasted space[5,6] .

However, for amateur programmers, spatial optimization algorithms are complex and can encounter considerable computational challenges. Especially, in our case, when addressing arrangement of objects in both two-dimensional and three-dimensional spaces.
Therefore, in order to solve the obstacles posed by computational complexity, the project attempts to develop and implement advanced algorithms that strategically place objects, specifically pentominoes, within predefined spaces. Our aim is to find and implement efficient solutions that

balance the need for optimal arrangements in both two-dimensional and three-dimensional scenarios. By exploring the adaptability of existing algorithms, such as the greedy algorithm, to improve computational efficiency.

Therefore, it would be interesting to try to answer this question: **How can the implementation of a computer application algorithm addressing strategic placement in both two-dimensional and three-dimensional contribute to efficient spatial optimization?**

In order to answer this primary question, we structure our report around three interconnected secondary questions, each one of them corresponding to a different phase of the project. The first phase focuses on the understanding of the strategic arrangement, therefore it would be interesting to explore this question: **How can we efficiently implement an algorithm to arrange pentominoes in a predefined 2D space, addressing complexities in packing problems?** The second phase concentrates on the influence of strategic arrangement and the development of an intelligent bot on gameplay performances. Consequently we analyze this question: **How can the strategic arrangement of the 12 pentominoes in our Tetris-style game, along with the development of a bot influence gameplay performance and high score?** The third phase dive into three-dimensional knapsack problem. For this reason, it would be interesting to study this question: **How can the implementation of a computer application for three-dimensional knapsack problems contribute to efficient cargo-space optimization, considering various parcel types, sizes, and values?**

We will be discussing the in-depth inner workings of our algorithms for each phase in Section II, then the way we tracked performance and the outcomes in Sections III and IV respectively. In Section V we will give some introspection on the data of the experiments and to end with Section VI we will bring it all together into answers for the questions that we were tasked with answering.

III. METHODS

A. Sorting Algorithm

When we started we wanted to have a simple algorithm, namely the Brute-Force approach, the performance of which we could compare with more sophisticated approaches.

The algorithm works recursively, meaning that it “saves” states and returns back to them if the path that they followed didn’t result in a solution. It places a piece as long as it’s placement doesn’t break any rules (i.e. having part of it overlap another piece or go outside the bounds of the board.) and then check all of the possible combinations and placements of remaining pieces, If it returned it would choose another piece to start with, and if every first piece was exhausted at the first location, it would do the same with the next location, which is coordinate next to the one we started with.

It becomes apparent through analysis that this way of searching for solutions is incredibly inefficient, something which is exacerbated by the fact that the computer will try possibly thousands of combinations on branches that are impossible to solve. This is a result of our program not

checking for dead spaces which get formed when a gap gets created that is not a multiple of five in size. As a result no combination of Pentominoes is able to fill the gap. We quickly realized that we would have to find another approach, since even when we approached the finalization of the algorithm's code, the time to solve would be unacceptably long in any real world scenario.

Instead of completely scrapping our work we decided to expand upon it and fix a couple of the biggest time wasters that existed in our approach. Namely putting certain pieces in certain positions that would result in the aforementioned Dead-spaces. For example, the X piece could not be put in a corner because any orientation would cause a hole of size one to appear. If it was the first piece placed it would cause all the moves that would follow it to be practically useless and a big amount of time to be wasted.

We employed a series of conditionals that all operated similarly to reduce the net total tries. Although as we will see we reduced the **amount** of tries the **speed** was either marginally faster or, as was the case most of the time, virtually unchanged. This is due to us underestimating the detriment that dozens of if-statements multiplied by the amount of iterations would have on the speed of our algorithms. This will be analyzed further in Chapters IV., V. and VI..

Finally, we decided to scrap whatever we had created in favor of a simpler and much faster and consistent alternative. We implemented an algorithm that each time it places a piece, it checks for Dead-spaces by looking at the neighboring cells of an unoccupied spot and then summing the instances into a tally. If the tally is not divisible by five we don't place the piece and try the next one in line. This would be both the simplest method that we have tried and the best one in all the possible metrics.

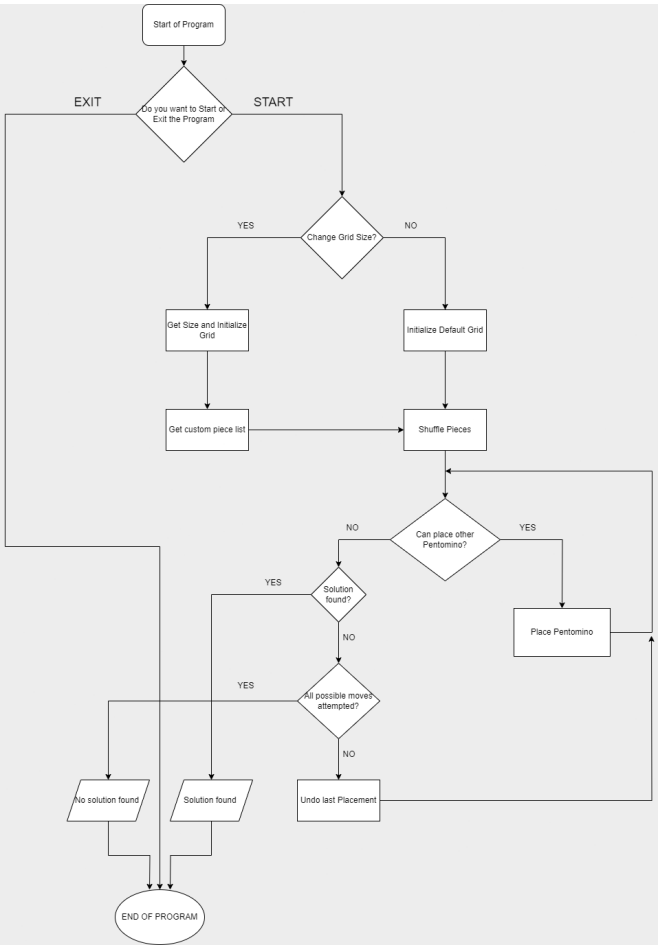

Fig. 1. Flowchart of 2D-Sorting-Algorithm

B. The game of Tetris with Pentominoes

B.i. The Game-Play

Due to the Phase 1 subsection of our project we were thankfully able to recycle a lot of the methods that had to do with operations on the pieces themselves, such as flipping them, and rotating them to certain orientations.

All of the pieces are housed on a 5X15 array board. In the Main loop of the game a piece is picked and then dropped. It decrements on the y-axis of the board on certain set intervals, while being controlled by the player on the x-axis, until one of its nodes found itself above an occupied foreign node. Subsequently it would stop the movement of the piece, and start from the beginning once more until it filled the end clause.

We also implemented some quality-of-life features such as the High-Score board, a field on the game screen that showed the next piece to be dropped, allowing for strategizing into the future and the pause functionality.

The greatest obstacle that we would have to face was the creation of the AI-opponent which the player would have to face. Even though coding the logic would be no easy task, finding the balance of challenge and fun would be the more difficult out of the two. Research in the field of play related psychology reveals that a player has to win about a third of the time to continue playing the game[7], the difficulty should also not be too easy since then it will feel to them like it has no purpose.

B.ii. The User-Interface

For the UI we used the JavaFX library and created three screens. The start screen gives the player the choice to either Play, look at the Scoreboard and Quit the application using three buttons. The Game-Play screen consists of the visualized 5X15 array which hosts the pieces, the field which shows the next piece to fall, and the player's score. The High-Score screen shows all of the previous score counts with the player name that is attributed to them.

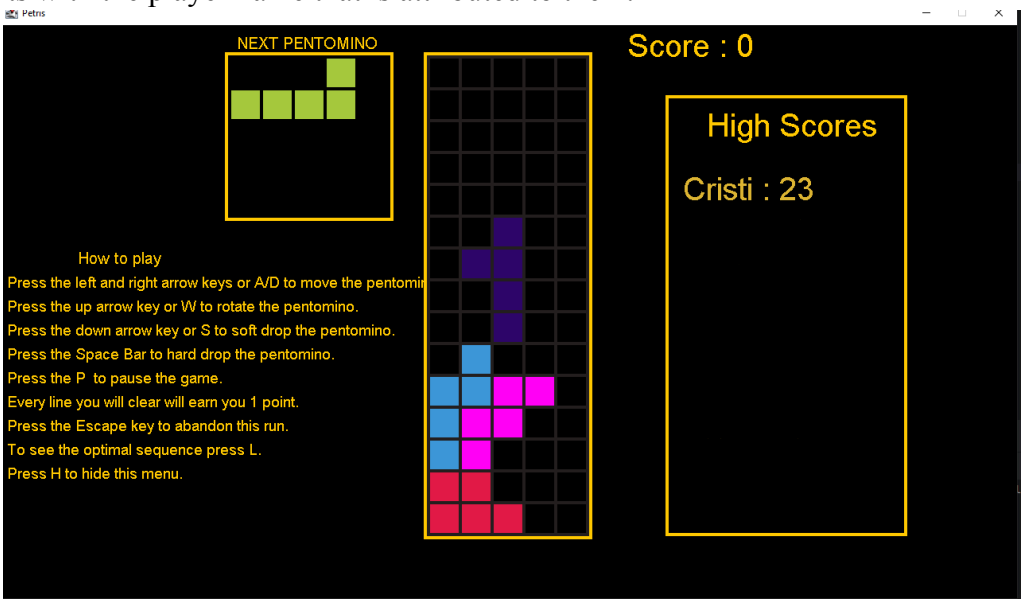

Fig. 2. The Main Screen of the Game

B.iii. The Bot and Optimal Order

When first considering and researching the possible algorithms that we could use for the bot, two options seemed to be the best contenders. The greedy algorithm which is based on mathematics and the Genetic algorithm which is based on machine-learning. The Genetic Algorithm was quickly discounted despite being the more sophisticated approach being also capable of discovering more elaborate and rewarding play styles. This is due to its complexity and required time investment, not only resulting from the additional coding time but also the time that would be spent training it, which could approach days if not weeks for the same skill level as the Greedy algorithm.

The Greedy algorithm is a way to find the best move possible by evaluating all of them and scoring them based on a selection of different criteria. Possible variables could be the amount of closed-in gaps the placement of the piece would result in, the amount of gaps the piece would fill and of course the elimination of lines. Even though the Greedy Algorithm gets beaten by the Genetic under some circumstances, its relative simplicity and close performance were the reasons why we chose it over all the others.

In order to assist the bot in reaching the highest score possible we would also have to find the order of Pentominoes that simplifies the process and improves its performance the best. Due to the simple shapes of the Pentominoes, many different orders that perform the same way exist. We decided to settle with the “LNUVZWPXIFTY” but this is not the only one.

We were able to find this sequence by utilizing the algorithm from Phase 1 that solved our grid and presented us with a solution that was able to be carried out by the computer without breaking any of the game-logic laws.

C. Three-Dimensional Sorting Algorithm

C.i. The Algorithm

In order to solve the problem of filling the 3D knapsack, we designed two separate Algorithms. Albeit similar in their implementation they strive to achieve different goals.

The first tries to fill the board in a manner that leaves as few unoccupied nodes as possible. It works recursively in a fashion akin to the 2D Algorithm, by placing a piece and then attempting all possible resulting combinations, if no solution is found it goes back and tries another piece at the same location or goes to the following one.

The second prioritizes score and utilizes a slightly different technique. At first it fills as many positions as it can on the board with the piece that rewards the highest score. Then it fills out the remaining gaps with the piece that is most capable of doing so. Thus we have maximized the amount of highest scoring pieces that can fit and then minimized the spatial waste.

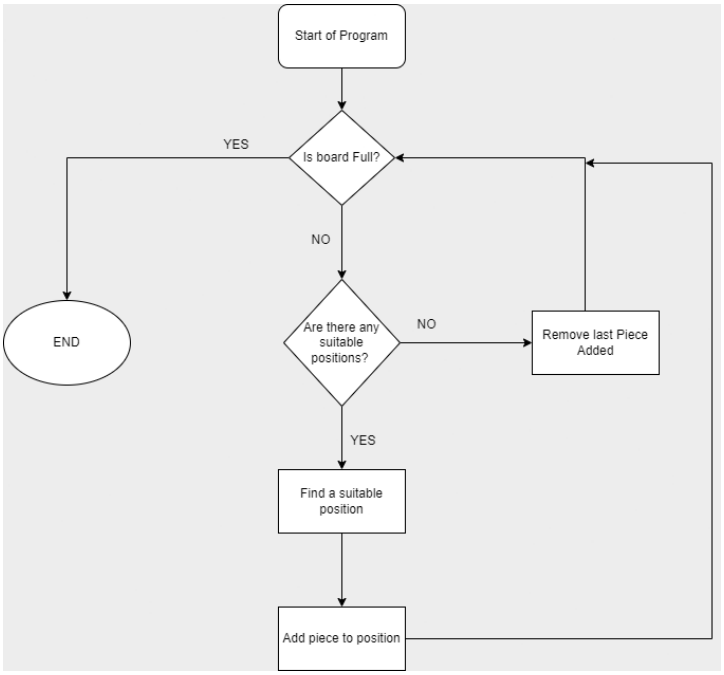

Fig. 3. Flowchart of 3D-Sorting-Algorithm

C.ii. Visualization

Accompanying the answers to the first two questions are GUIs that showcase visually the state of the data in the back-end. Although many of the core concepts carry over to this problem, the increased dimensions of the Pentomino Container also meant that we would need to find a way to portray them. In the JavaFx library exists a shape called Box. It shares characteristics with any other ordinary cuboid such as length, width and height as well as other cosmetic ones like color.

We found a simple and effective way of representing data in our render to be utilizing a “Mapping” function that checks if a node in the Array was occupied, and if it is, a cube is placed in the corresponding position in the 3D space by multiplying the size of the cube with the coordinates in the Array. This is done for every possible coordinate and in the end we get the fully populated container.

Naturally, merely just creating those cubes would not be enough since we still didn’t have a way to show it to the user. We used a PerspectiveCamera object for this which also comes from the JavaFx library and through trial-and-error found a placement where we considered it to be fulfilling its purpose.

Additionally, we created a Screen that allows the user to choose the specific Algorithm that he wants to run, as well as a 3D render of all the pieces that partake in the search for a solution.

IV. EXPERIMENTS

To test out the efficiency of all our sorting algorithms we ran the different versions a hundred times each on a grid of size 5X12 with the same selection of pieces, that being all of them, on the same hardware with no applications active outside the IDE running the program itself. We took the best case, being the fastest time of all the executions, the base case, being the average, and the worst case, being the worst time.

Any tests that may be carried out using the same programs will most definitely not get the same time results as we did. This is due to hardware and software specifications not being the same across computers. Even different versions of the same Operating System under the same processor could yield wildly different results.

We also compared how the amount of conditionals affected metrics in the Pruned Algorithm such as the quantity of pieces added, and ultimately removed, the amount of conditionals the cpu computed and the time it took for a solution to be found.

Finally we analyzed whether or not it is possible to fill a truck of size 16.5X2.5X4.0 with three different types of parcels A:1.0X1.0X2.0, B:1.0X1.5X2.0 and C:1.5X1.5X1.5 OR the 3D Pentominoes L, P and T. Then we gave each Pentomino and Parcel a numerical score and made the program find the solution that maximized the knapsack's total score. Pieces A and L get a score of three, B and P get four, and C and T five.

V. RESULTS

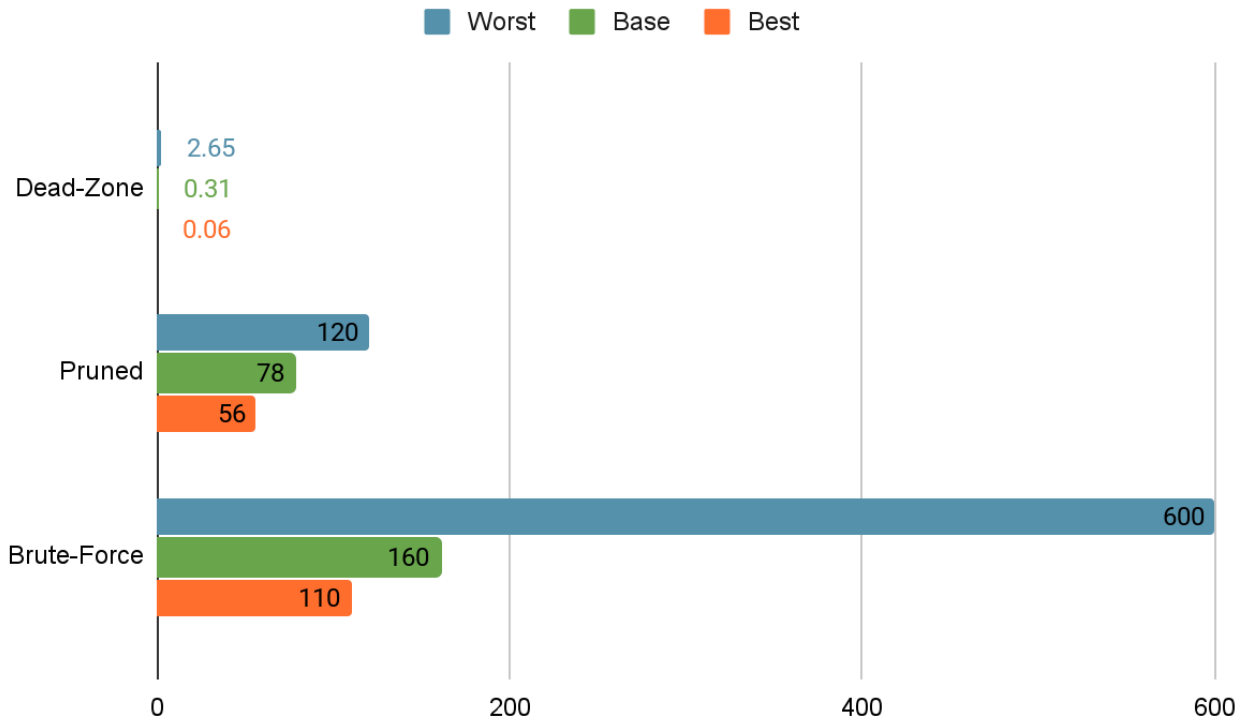

Fig. 4. Graph Showcasing speeds of Respective Algorithms. Top: Dead-Zone, Middle: Pruned, Bottom: Brute-Force. Score counted in seconds

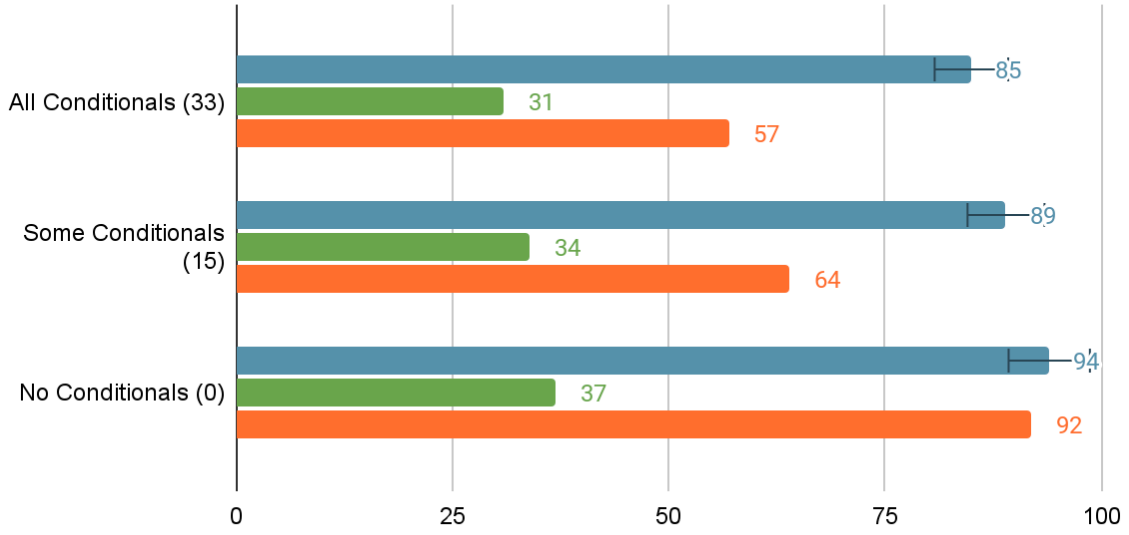

Fig. 5. Graph showing the difference in performance of different versions of the Pruned Algorithm depending on the amount of conditionals that check for illegal placement. Blue signifies time in seconds, green the total added pieces in millions and orange the amount of conditionals that were checked by the computer in tens of millions.

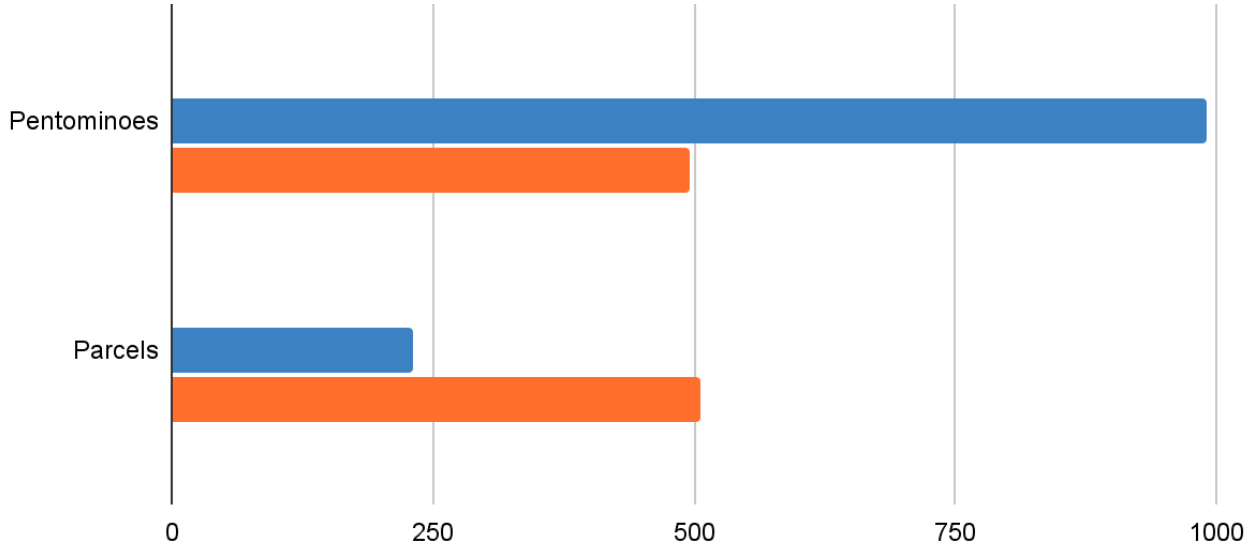

Fig. 6. Comparison of Score values for filling the Knapsack with Parcels and Pentominoes. The blue bar shows the maximum score achieved and the orange on the value of the score adjusted for average volume of the pieces (in tens).

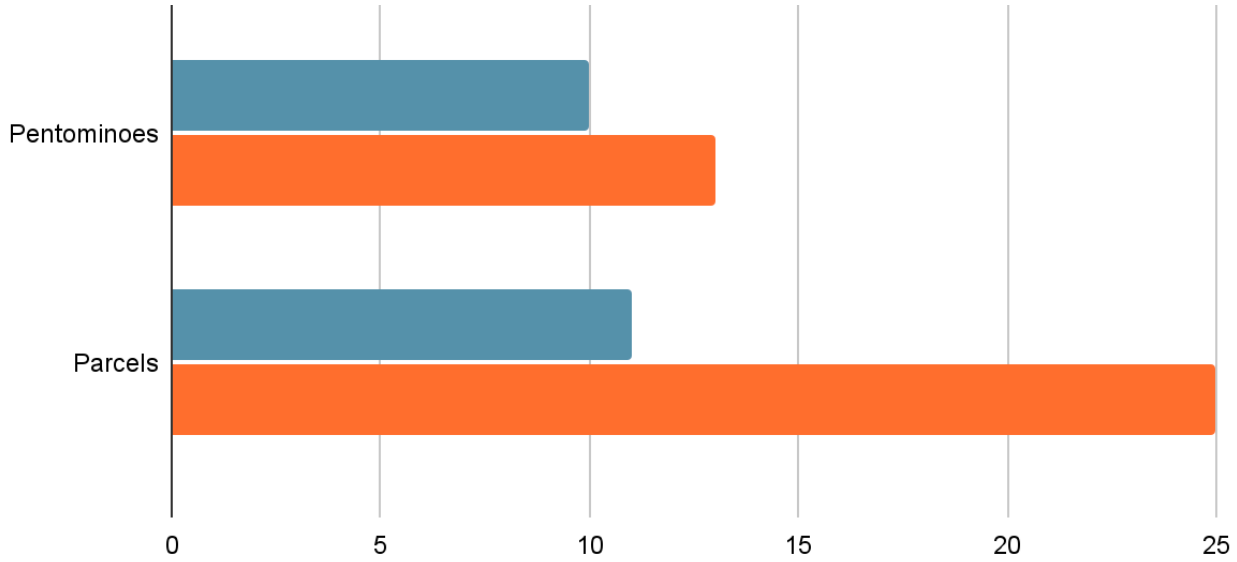

Fig. 7. Comparison of Fill Algorithm between Pentominoes and Parcels. The blue bar symbolizes the time taken to reach a solution in seconds and the orange one shows how much empty volume was left (in tens).

VI. DISCUSSION

Looking at the final results it is easy to see that not all of the algorithms that we wrote are equally as efficient in solving our first research question. The ones which base their success on luck also do the same with the time spent on a solution. It should be noted that even though the pure Brute-Force algorithm had the worst best case times, it does have the capacity to mimic the times of the Dead Zone approach since it could hypothetically get a solution on its first guess.

The pruned approach to the Brute-Force algorithm sees less differentiation between its best case and worst case than any other approach, at a 2.1x difference compared to 5.4x for the prune-less version and 44.1x for the Dead Zone one.

The smarter Dead-Zone approach is the fastest than all of the competitors utilizing its automated and faster elimination of choice method than the handmade that we implemented in the pruning Algorithm.

The results of the experiment that analyzes the impact of conditionals on the performance of the Algorithm can be seen in Fig. 4.. The values of Conditionals Judged and Total Pieces Added are inversely related to the Time spent to find a solution and the amount of Conditionals that exist in the pruner function. As a result when there are more checks in place that judge the positioning of a piece, more time is taken to find the solution, due to the added processes that the cpu needs to take care of. The accuracy of the program is nevertheless increased and this could be utilized in scenarios where that is important.

Looking at the results of Fig. 6. and Fig. 7. we can conclude a few things.

First of all we found it impossible to fill the entire truck with only parcels. We can reach a minimum amount of empty cells that approaches zero but doesn't reach it. The same applies to

Filling it with Pentominoes but this is because of Algorithm has imperfections. A perfect fill is theoretically and practically possible but not something we achieved.

As for the Score based Algorithms a Score of close to 5000 points was able to be reached by both the Parcel version and the Pentomino one, when adjusted for their volumes. For that outcome we used the function (Average Volume x Score Reached), where Average Volume was 22 for the Parcels and five for the Pentominoes. The closeness of the results means that even if different pieces and values are given to the program a viable and satisfactory solution will still be found.

VII. CONCLUSION

To answer our initial question, an Algorithm that utilizes scans of the board before continuing onto a path using relative positions of empty nodes on it will give us the most fruitful results due to it not spending its time on branches that would be impossible to solve and not having to evaluate dozens of conditional statements many thousands of times over its search.

Real World sorting problems could be assisted, if not entirely solved, by our 3D sorting Algorithm. Its flexibility in regards to objects, and their accompanying scores, allows the user to set their own parameters and solve their custom problems without the need to engineer a solution themselves.

Our group acknowledges the existence of a well-known algorithm type called the “Dancing Links” Algorithm which returns possibly faster times than our fastest 2D Sorting Algorithm. Due to the time constraint that we were facing we were unable to implement it, and of course test it, but it would be of minimal importance since our Dead-Zone approach is faster than any human being’s perception.

Through our research we expect to see more advanced algorithms come to light that utilize the concepts of our implementations as their base. If the pitfalls of our own solutions, such as the ones in the 3D Pentomino fill algorithm, get solved their utility in real world scenarios, being predominantly the transport business where fully taking advantage of the given space matters, could increase and become more widespread.

REFERENCES

1. Branke, J. (1999). Memory enhanced evolutionary algorithms for changing optimization problems. Proceedings of the 1999 Congress of Evolutionary Computation-CEC99 (Cat. No, 99TH8406). https://doi.org/10.1109/CEC.1999.785465
2. Gandomi, A.H. Yang, XS. & Alavi, A.H. (2013). Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. Engineering with Computers 29, 17–35. https://doi.org/10.1007/s00366-011-0241-y
3. Liu, L. (2011). Solving 0-1 Knapsack Problems by Greedy Method and Dynamic Programming Method. Advanced Materials Research, 282-283: 570-573. https://doi.org/10.4028/www.scientific.net/AMR.282-283.570
4. Lauri, J., Dutta, S., Grassia. M., Ajwani. D. (2020). Learning fine-grained search space pruning and heuristics for combinatorial optimization. https://doi.org/10.48550/arXiv.2001.01230
5. Mahnken. D. (2023). How to Reduce the Cost of Logistics with Truck Optimization. https://www.saloodo.com/blog/how-to-reduce-the-cost-of-logistics-with-truck-optimization/
6. Murray. C. (2023). Logistics Management: Optimizing Efficiency and Cost. https://www.iienstitu.com/en/blog/logistics-managementn-optimizing-efficiency-and-cost
7. Panksepp, J., Beatty, W. W. (1980). Social deprivation and play in rats. *Behavioral & Neural Biology. 30*(2): 197–206. https://doi.org/10.1016/S0163-1047(80)91077-8