Project 1-1

The packing problem

Samantha Cijntje

Pie de Boer

Emre Karabulut

Agata Oskroba

Liutauras Padimanskas

Liwia Padowska

Jadon Smith

Department of Data Science and Knowledge Engineering

Maastricht University

January 17, 2022

Abstract

This research paper aims to provide various strategies applicable to solving 2-dimensional and 3-dimensional packing problems. Packing problems refer to problems in which certain objects need to be placed into a constrained container. A solution to a packing problem could serve as a starting point for shipping companies. The research question for this paper is: What strategies are more suitable to maximise space utilisation? Three quantitative experiments were conducted.

The first experiment was designed to find out what effect certain fitness factors have on the efficiency of packing. This experiment tested a greedy algorithm in filling a two-dimensional space. In order to improve results, a genetic algorithm was employed with an emphasis on determining the weights of the factors. Various factors were tested in different sets to determine their effectiveness and the usefulness of the genetic algorithm. According to the results, increasing the number of factors shows an improvement in packing space efficiency. This report does not cover more than six factors.

The second experiment is based on a cargo space with the dimensions of 16.5 m x 2.5 m x 4.0 m and two sets of three types of parcels. The first set contains parcels shaped as L, P, and T letters that consist of five cubes, where each cube is of size 0.5 m x 0.5 m x 0.5 m. The second set contains parcel A (1.0 m x 1.0 m x 2.0 m), parcel B (1.0 m x 1.5 m x 2.0 m) and parcel C (1.5 m x 1.5 m x 1.5 m). This experiment was conducted to determine the maximum value that may be stored in the cargo space using each of the two sets of parcels with consecutively assigned values of 3, 4, and 5. In this experiment, a greedy algorithm was compared to an extended greedy algorithm, along with different sorting options. The weights of fitness factors applied were optimised using a genetic algorithm. The results found that for the first set a maximum value of 1230 could be reached using the descending ratio sorting technique of the greedy algorithm. For the second set, the highest possible value of 233 has been reached with a descending volume sorting. Both of these results were obtained using weights optimised by the genetic algorithm.

Lastly, the third experiment was conducted to learn whether the above-outlined sets could fill a cargo space without any gaps. To find an exact cover, a dancing links technique was used for an efficient implementation of Algorithm X. For further investigation of the DLX performance, we sequentially averaged 100 runs and concluded that an exact cover can be found for the first set using only L and P parcels, while an exact cover could not be found for the second set.

From these experiments, we can conclude that the strategy that is most suitable to maximise space utilisation is Algorithm X incorporated with the Dancing Links technique (DLX). Although, it does not take the value of parcels into account. To tackle this problem, a greedy approach with a genetic algorithm optimization proved to be the most suitable.

Content

1. Introduction

Packing problems refer to problems in which certain objects need to be placed in a bounded container. This report presents various solutions for packing problems in 2D and 3D spaces. With a focus on reducing the complexity, this paper outlines the O-2DPP (*Ordered 2-Dimensional Packing Problem*) and the 3DPP (*3-Dimensional Packing Problem)* that ultimately lead to packing items into a certain space as efficiently as possible. Therefore, the following research question was formulated: What strategies are more suitable to maximise space utilisation? To answer this research question, the following sub-questions shall be answered first:

a. What effect do the various factors have on the efficiency of packing?
b. If parcels of three different types represent values of 3, 4 and 5 units, then what is the maximum value that can be stored in the cargo space?
c. Is it possible to fill the complete cargo space with three types of parcels, without having any gaps?

As real-life packages have various shapes, the objects that will be considered for packing problems are so-called *pentominoes*. A pentomino is a two-dimensional structure obtained by joining five squares of side length 0.5 m. There are 12 pentomino shapes and they represent the following letters of the Latin alphabet: *P, X, F, V, W, Y, T, Z, U, N, L, I*. Moreover, if one imagines packages being transported via a conveyer belt, packing faces a new constraint: the order in which they arrive. For this purpose, we aimed to solve an O-2DPP. Furthermore, as cargo space is 3-dimensional, another dimension had to be introduced to solve the 3DPP. Additionally, a 3D visualisation of this problem had to be implemented to present the solution.

Furthermore, as considering all 12 different pentominoes might be too complicated for a 3DPP, we decided to focus on a subset of these shapes (pentocubes): L, P, and T, created with five cubes, where each cube is of size 0.5 m x 0.5 m x 0.5 m. Besides the pentominoes, another 3 shapes were introduced: parcel A (1.0 m x 1.0 m x 2.0 m), parcel B (1.0 m x 1.5 m x 2.0 m) and parcel C (1.5 m x 1.5 m x 1.5 m). Moreover, the parcels are assigned specific values, namely 3, 4 and 5 for A/L, B/P and C/T parcels in the respective order.

The relevance of this research lies in the fact that a solution to a packing problem could be helpful to companies in the packing industry (Pinsinger, 2002), whose goals are to pack a space efficiently. Some logistics companies may prioritise profit, while others may prioritise the number of parcels packed. The strategies discussed in this paper could aid the improvement of decision making processes by shipping companies.

The study of packing problems has been researched by various scientists. Most scientists agree that a heuristic approach to solve this problem is optimal. Some scientists, such as de Castro-Silva, et al. (2003), propose a greedy search heuristic for solving the three-dimensional bin packing

problem. The idea of a greedy algorithm is that the algorithm looks for the best way to fill a single bin by the parcels, where these parcels cannot overlap their boundaries with another parcel and a parcel that is inserted into the cargo-space is in a feasible position (de Castro-Silva, et al., 2003). Other scientists, such as Côté, et al. (2018), propose the meet-in-the-middle principle to solve packing problems (Côté, et al., 2018).

The knapsack 3D problem can be seen as an exact cover problem. Donald Knuth developed Algorithm X to solve exact cover problems. Unlike the greedy algorithm, this algorithm employs backtracking techniques. Additionally, Knuth suggests using the dancing links data structure to implement Algorithm X (Knuth, 2000).

After reviewing the literature on this subject, we decided to approach the subject using different algorithms. Intending to find a way to fill the spaces efficiently, a greedy algorithm has been used. The greedy algorithm is a combinatorial optimization algorithm. We also used a genetic algorithm to optimise the greedy algorithm. A genetic algorithm leverages the evolutionary generational cycle to create a high-quality solution that improves overtime to these challenges (Mirjalili, 2018). To find an exact cover of the 3DPP we used a DLX algorithm. The DLX uses dancing links as a data structure for Algorithm X.

In this report, the methods used are outlined first. Constraints and fitness are discussed along with each algorithm and its implementation. In the experiment section, the experiments which were conducted to answer our research questions are introduced. The results without any interpretation can be found in the results section, while their interpretation can be found in the discussion section along with knowledge from previous studies. In the conclusion, we answer each sub-question and use their conclusion to answer our main question.

2. Methods

2.1. Constraints

Dimensions
For the O-2DPP, the width of the grid on which all experiments were conducted was fixed to 5 units. For the 3DPP, fixed cargo space was assumed to be 16.5 m x 2.5 m x 4.0 m. For simplicity, we turned these metric dimensions into discrete values (units). This was done by multiplying all dimensions by two. E.g. 16.5 m x 2.5 m x 4 m would be 33 x 5 x 8.

Rotations
The only restriction regarding orientation was imposed in the first problem, the O-2DPP, where the flipping of packages was prohibited.

2.2. Fitness

Since a cargo space needed to be filled in a certain manner, different fitness factors had to be implemented. Because the algorithms in the next sections, Greedy Algorithm and Extended Greedy Algorithm, must determine the optimum location for a given parcel in the container, a fitness function that provides a fitness score had to be created. The fitness function is a sum of all fitness factors for a given packing problem multiplied by their weights. First, all weights of value 1 were considered, however they were later optimised using a genetic algorithm as described later in section 2.4.

To research the optimal solution of the ordered two-dimensional packing problem, we constructed a list of six factors that would be important for the algorithms:

A. **Height of the board:** height of the upper space left
B. **Empty squares:** the number of locked-in empty holes created
C. **Lines cleared:** a row without gaps
D. **Bumpiness:** variation of column height, the sum of obsolete differences between two adjacent columns
E. **Touching sides:** how many lateral grids are filled with pentominoes
F. **Touching bottom:** the number of blocks touching the bottom of the grid

However, as adding another dimension influences fitness factors, other ones had to be considered. The different factors for the 3DPP were:

I. **Total empty space:** the number of empty spaces of dimensions 1x1x1 in the cargo space
II. **Touching x wall:** the number of 1x1 grids on the left wall of the container that are touching packages
III. **Touching y wall:** the number of 1x1 grids on the front wall of the container that are touching packages
IV. **Touching z wall**: the number of 1x1 grids on the bottom wall of the container that are touching packages
V. **Number of touching parcel sides:** the number of connected 1x1 sides of a parcel to another parcel
VI. **Number of x gaps between parcels:** the number of gaps between parcels looking from the x-direction
VII. **Number of y gaps between parcels:** the number of gaps between parcels looking from the y-direction
VIII. **Number of z gaps between parcels:** the number of gaps between parcels looking from the z-direction
IX. **Total container value:** the total value of all parcels in the container

2.3. Greedy algorithm

We used the greedy approach to find a solution for filling the space efficiently, as both O-2DPP and 3DPP are NP-Hard packing problems and it is a combinatorial optimization. In the hopes of arriving at a globally optimum solution, the algorithm makes a locally optimal choice. Clearly, greedy algorithms don't always produce the best results. However, the greedy method is highly effective for a large range of significant situations (Vince, 2002). The greedy algorithm uses a breadth-first searching technique and it doesn't backtrack.

In general, the greedy algorithm evaluates only one parcel's placement at a time and is not aware of the next parcel. It does so based on the fitness score that is calculated according to the factors described in section 2.2. Once the best move is chosen, the parcel is placed in the container and the process starts again for the next parcel on the list. For the O-2DPP, the algorithm had to work based on a randomly generated next parcel. However, as the 3DPP is based on specific parcel types and amounts provided as user input, the list is sorted in the following ways:

1. **Ascending volume:** meaning that the lowest volume appears first
2. **Descending volume:** meaning that highest volume appears first
3. **Ascending value to volume ratio:** meaning that the least value-dense parcels appear first
4. **Descending value to volume ratio:** meaning that the most value-dense parcels appear first

As an example, if an ascending volume sorting is applied, all *A* parcels will appear at the beginning of the list and are evaluated one by one. Furthermore, since all pentocubes have a volume of 5, sorting them based on volume was irrelevant.

Moreover, as an attempt to optimize the greedy approach for the 3DPP to reach greater value, we tried to evaluate all three parcels at the same time. To facilitate this, a unique list is created from user input with removed duplicate parcels to decrease the execution time. This way, whenever we run out of one parcel type, it does not appear in the current list anymore. We decided to name this approach an Extended Greedy Algorithm.

2.3.1 Database

The databases used to solve packing problems in this paper were constructed out of arrays that represent all rotations of each parcel. The 2D shapes were represented as 2D arrays, where a parcel could be distinguished from empty space by its piece number. For the 3DPP also flips of parcels in a 3D space were included. Naturally, the 3D parcels were represented as 3D arrays. As another dimension increases the number of possible rotations, their number for each parcel increases to as much as 24 for L and P pentocubes. Such representation of different parcels was appropriate for our program implementation and was especially useful for our visualisations that use the values in a 3D array to paint each grid.

2.3.2 Pseudocode of implementation

As outlined above, for 3DPP, the 3D container C with dimensions of *width* x *length* x *height* (w_c x l_c x h_c) is to be filled. The greedy algorithm evaluates the packing problem for one parcel p at a time (see Fig. 1). The p is taken from the input list L^*, which is derived from the original input list L and is sorted with respect to the priorities outlined in section 2.3.

```
L* = L
Pick parcel p as the first element of sorted list L*
while |L*| > 0
  f_max = 0
  s_best = 0, 0 ,0
  for each x { 0, 1….. , w_c}
    for each y { 0, 1….. , l_c}
      for each z { 0,1…… , h_c}
        for each r {0, 1…. , r_p }
          if f_p > f_max
            f_best = f_p
            s_best = s_x , y , z
            r_best = r
          end if
        end for
      end for
    end for
  end for
  if there is no overlapping
    Place the p in its r_best to s_x,y,z in C
  end if
  Remove the piece from L*
end while
```

Fig. 1 Pseudocode for the greedy algorithm

```
while |L| > 0
  create the list U
  f_max = 0
  s_best = 0, 0 ,0
  for each p in list U
    for each x { 0, 1….. , w_c}
      for each y { 0, 1….. , l_c}
        for each z { 0,1…… , h_c}
          for each r {0, 1…. , r_p }
            if f_p > f_max
              p_best= p
              f_best = f_p
              s_best = s_x , y , z
              r_best= r
            end if
          end for
        end for
      end for
    end for
  end for
  if there is no overlapping
    Place the p_best in its r_best to s_x,y,z in C
  end if
  Remove the piece from L
end while
```

Fig. 2 Pseudocode for the extended greedy algorithm

As long as $|L^*| > 0$, p is chosen as the first element of L^*. Before evaluating the best possible move for p, the maximum fitness value (f_{max}) and best spot (s_{best}) are set to zeros. Then, every rotation r_t of p is tried in all possible spots ($s_{x,y,z}$) without overlapping in the C. The best placement

(s_{best}) and rotation (r_{best}) are evaluated according to the calculated fitness score f_p as mentioned in section 2.2. If $f_p > f_{max}$, f_{max} is assigned f_p's value. After the evaluation process p is placed to s_{best} in its r_{best}. In case there is no possible placement for p in C, p is removed from the L^* and this process is repeated until $L^* = \emptyset$ or no more parcels can be placed in C.
As an addition, for the Extended Greedy Algorithm (see Fig. 2), the best parcel type, p_{best} is determined after checking each type of parcel present in L. This is facilitated by the unique list U which contains only one from each specific type of parcel present in L.

2.4. Genetic algorithm

To find the weights for fitness factors that would maximise the total value which can be achieved in packing a two-dimensional and three-dimensional space, a genetic algorithm was implemented. A genetic algorithm uses the evolutionary generational cycle to provide a high-quality solution to these problems that improve over time (Mirjalili, 2018).

A general outline of the genetic algorithm could be summarised as follows. First, the initial population is created that comprises a set of chromosomes, which represents a set of possible solutions – in this case, a set of weights (the importance of each factor), which are randomly generated from a uniform distribution on an interval (0,1). Then, an evaluation step begins when the population's members are assigned their fitness scores. In the case of the O-2DPP, the score is based on the amount of fully-filled horizontal lines. When it comes to the 3DPP, the score represents the total value of parcels accumulated in the container. If a termination condition is not met, the algorithm starts looping and a new, better generation is created. In our case, this happens when a certain generation number isn't reached. It is based on the biological concept of natural selection, which refers to the survival of offspring with the best attributes inherited from their parents. As in genetics, a genetic algorithm requires methods that diversify the next generations such as crossover and mutation which will be described further in the "Implementation" section (Jacobson, et al., 2015).

Binary tournament selection
To select future parents from each starting population, we chose a tournament selection with a tournament size of two. This approach is simple to use because it does not involve any sorting. It also preserves diversity while providing the optimal solution (Santhi, et al., 2021). Although the low tournament size can increase the convergence time of the algorithm due to the reduced selection pressure, we decided to favour the diversity and simpler implementation over a decreased time. Thus, we compare every two consecutive sets of weights by scores they yield in the game and keep the one with a higher score. This way, diversity is preserved which is a huge advantage over elitist selection methods, such as truncation or roulette wheel methods, that have a risk of converging to local minima, as they favour fittest individuals (Jacobson, et al., 2015).

Discrete crossover

As in the tournament selection the size of the population is decreased by half, from every two parents we need to generate 4 offspring to compensate for selection losses. While creating offspring we implement a discrete crossover method with a 50% rate, so that each offspring's gene has a 50% chance of being selected from the mother or the father, which is handled using a Math library (Oracle Corporation, 2021). Such a high crossover rate promotes the search for new solutions while still keeping the genetic information from fitter individuals intact for the next generations (Jacobson, et al., 2015). The random real number determines which parent's genes will be taken for the child.

Parent 1: 1 1 1 0 1 0 0 1 0

Parent 2: 1 0 0 0 1 0 1 1 0

Offspring 1: 1 1 0 0 1 0 1 1 0 (Umbarkar, et al., 2015)

Mutation

To increase diversity in the gene pool while creating new offspring, we applied a mutation method that is performed if a randomly selected number on an interval (0,1) is smaller than a fixed mutation rate which we set to 0.1. Thus, each time there is a 10% chance of a mutation to occur, which is performed by assigning a random weight value to the current gene of offspring.

2.4.1 Pseudocode of implementation

```
initialise random 1st generation
for (g = 1 to N_gen)
        for (i_pop = 0 to N_pop)
                Simulates the 3D-Packing with i
        end for
        for(i_pop = 0 to N_pop)
                Pair up consecutive pairs
                Save the fitter one according to the value from each simulation
        end for
        for (i_pool = 0 to N_pool )
                Pair up fit individuals
                for (i = 0 to 4)
                        Discrete Crossover: Create new individual using i_n and i_n+1 genes
                        Mutate
                end for
                Move 4 new generated individuals to g+1
        end for
end for
```

Fig. 3 Pseudocode for the genetic algorithm.

The genetic algorithm is described in the pseudocode above (see Fig. 3). Parameter *g* represents the generation number, while N_{gen} stands for the number of generations to be evaluated. Next, as each individual i_{pop} in the population of size N_{pop} represents a set of chromosomes, each i_{pop} comprises the set of weights that are used to simulate the 3DPP. After the binary tournament selection, the population size is decreased by half, so is mating pool size N_{pool}. Then, for each individual i_{pool} in the mating pool, a discrete crossover and mutation take place. At the end of the recurring process, all newly generated individuals are moved to the next generation.

2.5. DLX

To find a solution that would fit as many parcels as possible without leaving empty space for the three-dimensional packing problem, the dancing links technique was implemented to efficiently run Algorithm X. The knapsack 3D problem is a classic example of an exact cover problem. Other noteworthy exact cover problems are the pentomino tiling, Sudoku and the n-queens problem. An exact cover problem is a kind of constraint satisfaction problem. The components for a constraint satisfaction problem are:
1) A set of variables, X $\{x_1, x_2, x_3..., x_n\}$
2) A set of domains, D $\{D_1, D_2, D_3..., D_n\}$
3) A set of constraints that allow specified combinations (Russell, et al., 2011 p.165).

In mathematics, the exact cover problem can be represented using a binary matrix. To solve an exact cover problem, the subsets of rows such that each column contains element 1 exactly once must be found. In Fig. 4, the rows {1,3,5} represent a solution to a simple exact cover problem given some binary matrix A with dimensions 5 x 6.

1	0	0	0	1	0
0	0	0	1	0	0
0	1	0	0	0	0
0	0	0	1	1	0
0	0	1	1	0	1

Fig. 4 Binary matrix A

Algorithm X is an algorithm developed by Donald Knuth for finding all of the solutions to an exact cover problem represented by a given binary matrix. The algorithm is non-deterministic, depth-first, recursive and uses backtracking (Knuth, 2000).
If the algorithm was run with regular matrices, it would take a lot of computing power because, in Algorithm X, rows and columns have to be regularly eliminated and restored from the matrix. It would also make keeping track of deleted lines a complex process. Therefore, the dancing links technique was chosen to tackle these problems.

Dancing Links uses a two-dimensional doubly linked list (DLL). DLL is a data structure technique that is suitable for tackling backtracking problems. Each node in the mentioned structure is a data object that represents 1 in a binary matrix and it contains its name and data. In addition, it also has 4 fields: UP, DOWN, RIGHT, LEFT (Knuth, 2000, p.5).

An example of our DLL implementation for a binary matrix B (see Fig. 5) is presented in Fig. 6.

0	1
1	1

Fig. 5 Binary Matrix B

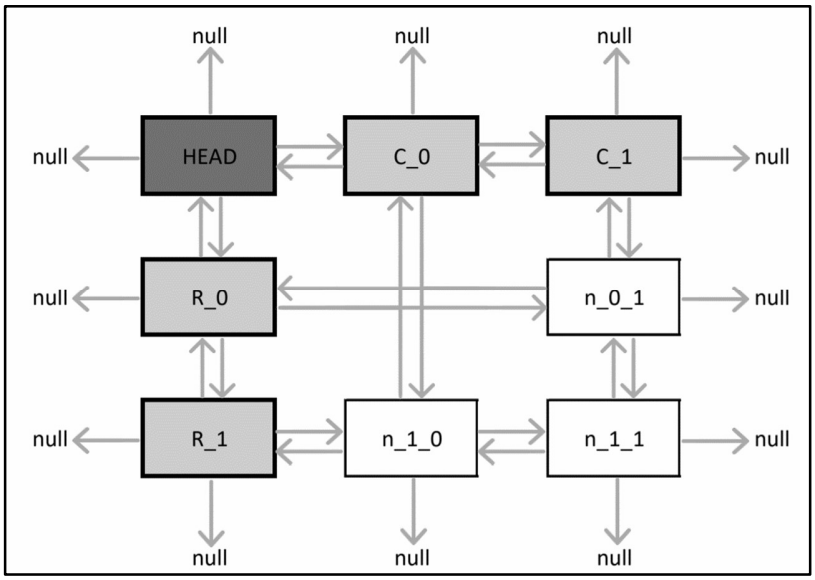


Fig. 6 Two-dimensional DLL structure

The powerful thing about this approach is that instead of deleting redundant lines, we break or restore references between nodes. It clarifies the fast execution of the algorithm. In our project, breaking the reference happens only vertically, thus the nodes are still fully visible horizontally. This makes the uncovering of the nodes a much easier process.

Algorithm X together with DLL (DLX) is only used to solve exact cover problems (Knuth, 2000, p.3). To combat the nature of DLX, we modified the algorithm to evaluate all of the partial solutions (the leaf nodes of the search tree). To evaluate partial solutions the program checks the number of columns that are present in the leaf node solution.

2.5.4 Database

The database for the dancing links algorithm is a binary 2D array that is created by placing each type of parcel to all positions in the container. After each placement, the 3D binary array which represents the container is converted to a 1D binary array and added to the database. Therefore, each row in the database represents the state of the container with only one placement.

2.5.5 Implementation details

In Fig. 7, the pseudocode for Algorithm X can be found.

```
while matrix A has columns left
    Choose a column c with the least amount of 1's in it
    Choose a row r that has a 1 in column c
    Declare row r in the partial solution
    for each column j such that A_{r,j} is equal to 1
        Delete column j from A
        for each row i such that A_{i,j} = 1,
            Delete row i from A
        end for
    end for
end while
```

Fig. 7 Pseudocode for Algorithm X.

While matrix A has columns left, pick a column c, such that c has the least amount of 1s. Proceed by picking a row r, having an element 1 in c. The row r is declared as a partial solution. Subsequently, loop over columns j and delete j for which $A_{r,j}$ equals 1. Loop over rows i and remove the rows that satisfy $A_{i,j}$ equals 1. Recursively repeat algorithm on reduced matrix. (Knuth, 2000, p.4).

2.6. Visualisations

Data visualisation is useful for exploring data structure, detecting unusual groups, identifying trends and clusters, spotting local patterns and presenting results (Unwin, 2020). We have used the

JavaFX library to create a 3D universe and arrays to represent parcels and the container (see appendix 1). The reason for presenting the visualisations in this way corresponds with the motivation behind this research. This type of visualisation could be helpful to companies in the cutting and packing industry. Companies can input the number of parcels of each kind, their importance, and visualise how they need to be packed into the container.

2.7. Implementation

For project design, object-oriented programming was used. Implementation of interface parcel has facilitated the creation of a generic greedy algorithm, which uses an interface type that refers to both Cube and Pentocube classes, as they both implement interface parcel. The UML diagrams in Appendices 2 and 3 outline how different classes are interconnected.

3. Experiments

The computer that was used to conduct the experiments has the following specifications; RAM: 8GB 1333 MHz, CPU: i5-8300H 4 cores with hyperthreading at 2.30GHz.

In expectation of measuring the performance while evaluating different factors, comparing them, and quantifying the impact of the genetic algorithm for the O-2DPP, experiments took place. A sufficient sample size of 10.000 runs was desirable, and in order to run such a large experiment, the tester class was created. The tester class can run any factor while setting any amount of runs and can save data about each run once that run converges. For this experiment, the final score of each of the 10.000 runs was saved. The score is based on the number of filled lines without gaps. For any run with a group of more than one factor, the GA can be used to optimise the weights of different factors.

To compare and analyse the data gathered for the O-2DPP, we tested different groups of factors in order to determine the effectiveness of the factors and determine the usefulness of the GA. The following groups of factors were tested:

1) No factors
2) Factor: A
3) Factors: A,B,C,D
4) Factors: A,B,C,D,E,F

This experiment focuses on our first research question. Namely, what effect do the various factors have on the efficiency of packing? From this experiment, we will be able to conclude if having more factors will increase its efficiency and if introducing a genetic algorithm to optimise the weights is a good strategy.

Moreover, for the 3DPP, experiments that compare the performance of the greedy and the extended greedy algorithms were conducted, with different sorting options. After that, equally distributed weights were optimised by the genetic algorithm. The settings for the cube and pentocube experiments with genetic algorithm were the following (see Fig. 8):

	Settings for cubes	**Settings for pentocubes**
Number of parcel types	90 A, 90 B, 90 C	264 L, 264 P, 264 T
Values of parcel types	3, 4, 5	3, 4, 5
Sorting type	Descending volume	Descending ratio
Initial population size	64	32
Mutation rate	0.1	0.1

Fig. 8 The settings for genetic algorithm optimization of weights

As outlined in the table above, as the infinite amount of parcels of each type had to be assumed, we decided to provide the maximum possible amount for all of them considering their volumes and the volume of the container, which in this case is equivalent to infinity. Moreover, the sorting options yielding the best results during the experiments with equal weights were considered as an attempt to find the highest values. The initial population size for pentocubes was two times smaller, as it was considered sufficient and decreased the execution time.

The weights obtained from the genetic algorithm were applied to the fitness function of the greedy algorithm and experiments that once more compared different sorting options were conducted. The aspects that were taken into account included the execution time, the value of the container, and the percentage of it being filled. Additionally, in the below results section, the number of different parcel types present in the solution was also shown. This experiment would lead to a conclusion on what the maximum value is that we can reach using our algorithms. By sorting it in different ways we can also see how the type of sorting that is chosen would affect the maximum value.

Moreover, we had to conduct more experiments to answer the research question regarding whether it is possible to fill a container of dimensions 33 x 8 x 5 with cubes or pentocubes. To achieve a faster answer using the DLX, a database we created for a 1 x 8 x 5 container for pentocubes was used. From a geometric point of view, this shows that we can fill the 33 x 8 x 5 container, since there is an exact cover for 1 x 8 x 5 the container would just need 33 layers of the same cover. Besides this, we wanted to investigate how increasing dimensions and swapping the order of the cubes or pentocubes would influence run-time. We assumed an infinite amount of either pentocubes (L, P, T) or cubes (A, B, C) in this experiment. For accuracy, the provided data is generated by taking the average of 100 runs.

4. Results

The following section presents experimental results that were obtained while testing the performance of the greedy, genetic, and dancing links algorithms in solving the packing problems.

While evaluating the O-2DPP, the amount of fully-filled lines in the 2D space was considered as a score, as mentioned prior in section 3. For comparison, different fitness factors were used to evaluate the best move in the greedy algorithm. With the first 3 groups, all of the factors had equal weight, were tested, and their average score was saved. However, once there was a group with more than one factor the genetic algorithm was put to the test to optimise the weight of each factor. Thus, for the 4 and 6-factor experiments, the greedy algorithm was combined with the GA, as shown in the table below (see Fig. 9).

Legend: A = Height, B = Lines Cleared, C = "Bumpiness" of top row,

D = Holes, E = Touching sides, F = Touching bottom.

	Factors considered	**Average Score After 10000 runs**
Weights Distributed Equally	-	0.561
	A	3.221
	A B, C, D	6.972
Weights Generated by GA*	A, B, C, D	8,126
	A, B, C, D, E, F	9.945

Fig. 9 Comparison of average scores for the greedy algorithm with different fitness factors considered and optimised by the genetic algorithm.

For the 3DPP, the greedy algorithm and the extended greedy algorithms were compared after and before being optimised with the genetic algorithm (see Fig. 10). The following table outlines the values that were obtained for the execution time, total container value, and percentage filled. One can also see what amounts of different parcels were used. For the experiments on pentocubes, the parcels weren't sorted regarding volume, as all pentocubes have the same volume, which is why values for this type of experiment are not provided.

As can be seen in the table below, the maximum value obtained for cubes without a genetic algorithm optimization is 230. It is also the highest percentage filled of the container achieved for cubes, with a value of 99.55%. It was obtained for both the descending volume sorting for the greedy algorithm and the extended greedy algorithm. The GA optimization added 3 points to the

value, scoring 233 in the same cases. As for the pentocubes, the initial value was 1185, which was improved to 1230. Both of these highest values occurred for the descending ratio sorting.

Algorithm			**Greedy**				**Extended Greedy**
Type of sorting			Volume		Value/Volume Ratio		
			Ascending	Descending	Ascending	Descending	
Cubes	Without GA	Execution time [s]	0.55	0.44	0.54	0.42	1.19
		Total Value	192.00	**230.00**	188.00	192.00	**230.00**
		Percentage filled [%]	77.58	98.33	84.24	77.58	98.33
		Number of A - B - C	64 - 0 - 0	8 - 24 -22	8 - 41 - 0	77 - 5- 8	8 - 24 -22
	With GA	Execution time [s]	0.45	0.27	0.38	0.41	0.74
		Total Value	192.00	**233.00**	200.00	192.00	**233.00**
		Percentage filled [%]	77.58	99.55	90.91	77.58	99.55
		Number of A - B - C	64 - 0 - 0	9 - 24 - 22	0 - 50 - 0	64 - 0 - 0	9 - 24 - 22
Pentocubes	Without GA	Execution time [s]	-	-	15.01	7.13	49.41
		Total Value	-	-	781.00	**1185.00**	990.00
		Percentage filled [%]	-	-	98.48	90.15	99.24
		Number of L - P - T	-	-	259 - 1 - 0	2 - 1 - 235	59 - 202 - 1
	With GA	Execution time [s]	-	-	14.55	6.56	47.55
		Total Value	-	-	778.00	**1230.00**	840.00
		Percentage filled [%]	-	-	97.73	93.94	99.24
		Number of L - P - T	-	-	225 - 2 - 1	2 - 6 - 240	211 - 48 - 3

Fig. 10 Comparison of the execution time, total container value, percentage of the container filled and the number of different parcel types present in the container for both greedy algorithms.

Moving to experiments regarding the performance of the DLX algorithm, in Fig. 11 and Fig. 12, the column ‘rows’, represents the number of rows in our dancing links data structure. If we choose the first row (dimensions 1 x 5 x 8) as the basis for the row factor and set its value to 1.00, we can express the relative size of each ‘type of run’. If we again pick row 1 (dimensions 1 x 5 x 8) to be 1.00 as RunFactor, we see the relative increase in runtime with the other type of runs.

It suggests that an increase in overall volume increases runtime. It can be seen that besides the increase in size, the order of the “dimensions” also affects runtime. This can be explained due to how the dancing links structure is formed with the given database and how Algorithm X performs with it.

Dimensions	Parcel Type	Exact Cover	Rows	RowFactor	Runtime DLX [ms]	RunFactor
1 x 5 x 8	Pento	yes	388	1.00	84.4	1.00
1 x 8 x 5	Pento	yes	388	1.00	114.8	1.36
2 x 8 x 5	Pento	yes	1156	2.98	328.2	3.89
2 x 5 x 8	Pento	yes	1156	2.98	142.1	1.68
3 x 5 x 8	Pento	yes	2408	6.21	1455.7	17.24

Fig.11 Analysis of runtimes using pentocube parcels.

Dimensions	Parcel Type	Exact Cover	Rows	RowFactor	Runtime DLX [ms]	RunFactor
2 x 4 x 8	Cubes	yes	38	1.00	71.5	1.00
3 x 5 x 8	Cubes	no	174	4.58	141.8	1.90
32 x 4 x 8	Cubes	yes	3735	98.29	862.3	11.57
33 x 5 x 8	Cubes	no	5754	151.42	50830.4	682.29

Fig. 12 Analysis of runtime using cube parcels.

5. Discussion

This section will focus on discussing the results obtained for the experiments aiming to compare the different approaches in packing of 2D and 3D spaces.

By analysing the O-2DPP, from Figure 9, it can be observed that having more factors has led to improved performance. However, as this paper did not evaluate more than 6 factors, it can not be stated that considering more factors will further increase the performance. Following that statement, it can also be observed that in the instance where more than one factor is being used, a genetic algorithm assigning different weights to each fitness factor is beneficial.

When it comes to the 3DPP, it can be seen that a greedy algorithm is a more optimal solution than the extended greedy algorithm if considering the execution time. It takes approximately 3 times less time for the greedy algorithm to find the solution, which is because while the greedy algorithm analyses only one parcel at a time, the extended version analyses as many as 3 parcels.

Moreover, for the cubes, both the descending volume sorting and extended version does indeed yield the best results for the total container value and its percentage filled. However, the basic greedy approach arrives for the same values for a descending volume sorting faster. The genetic algorithm optimization does increase the results for cubes, as the total values obtained in all cases are either the same or greater. It also arrived at the highest value of 233 for both the descending volume sorting and the extended greedy approach. In this case, we came the closest to obtaining an exact cover for cubes with 99.55 % of the container filled.

When it comes to pentocubes, the basic greedy approach with a descending ratio sorting beats the extended version's results. Thus, the attempt to maximise the total value obtained appears to be futile. The genetic algorithm improves the value only in the case of the setting on which it was trained, the descending ratio. This led to obtaining the highest value of 1230 and 93.94% filled, which is only 90 less than the theoretical highest value possible (1320 if an exact cover would only involve pentocubes T).

From these results, we could see that using a genetic algorithm to optimise weights improves the greedy algorithm. According to Santhi et al., 2021, the success of using genetic algorithms is based on selecting the right parents for generating new offspring, efficient crossover operators and mutation search operators to guide the system to obtain an optimised solution (Santhi, et al., 2021). The selection method as well as the crossover operator needs to be suitable for the problem. This implies that these elements are suitable for our current needs, however, we believe that they could still be improved in the future.

What is also noteworthy is that with both the cubes and pentominoes, the descending sorting is more efficient. Scientists such as Rinnooy Kan, et al., 1993, and Julstrom, 2005, suggest descending value density sorting (Rinnooy Kan, et al., 1993; Julstrom, 2005). While this was an effective choice for the pentocubes, it was not the best choice for the cubes. As seen in the table in (Fig. 10), descending volume was more efficient than descending ratio sorting. Additionally, it could also be deduced that the extended greedy algorithm is less effective when it comes to execution time. This is because of the time complexity of the greedy algorithm O(n) (Naeimi, et al., 2004). The linearity of the time complexity implies that the extended greedy algorithm will take three times longer to go through the database.

As for DLX, it performed below our expectations. We speculate that it might be our implementation at fault. The program would sometimes run into deep recursion and crash before finding a perfect cover. This helped us realise that the search tree becomes enormous for bigger and more complex problems.

For the time this research was conducted, we found that visualising the data in the form of a three-dimensional model helped with uncovering details that would go unnoticed without it.

5.1 Limitations

The limitations of this study include for the O-2DPP that we only focussed on six specific factors and did not explore other factor possibilities. For the 3DPP, we did not optimise our Algorithm X. Moreover, we only trained the genetic algorithm on a specific sorting setting. Lastly, this study only focussed on specific parcel shapes. In reality, not all packages have squared edges.

6. Conclusion

Answering the first sub-question, it can be concluded from the O-2DPP that having a sufficient amount of factors, will achieve an efficient packing solution. Sufficient in this case refers to what will achieve a result that is suitable to the goals set by the company or person packing the space. Following that, using a genetic algorithm to find what the most efficient weights are for each parameter can achieve good results. It is important to take into account that the O-2DPP algorithm has only been tested on the six specific factors. The potential for this strategy is still unknown, as not all possible parameters or all permutations of them were tested. There is the possibility that a completely different algorithm would be superior, such as giving each position of the field a score, and ranking placements according to the sum of each position that they take.

To answer the second sub-question for the 3DPP, only the different variations of the greedy algorithm and an extended greedy algorithm were used. For the cubes, the highest possible value was found to equal 233 and was achieved by optimising the weights of the greedy algorithm using

a genetic algorithm. Moreover, for the pentocubes, a maximum value of 1230 was noted while using a descending ratio sorting and once again, weights optimised by the genetic algorithm.

Furthermore, answering the third sub-question for the 3DPP, the outlined algorithms did not provide a solution for an exact cover using A, B and/or C parcels. The greedy approach, on the other hand, came close, occupying 99.55 % of the available space, which can be regarded as a commendable achievement. For the pentocubes, however, an exact cover was found by the DLX algorithm, which included only L and P parcels. Although this result indeed covered 100% of space, it didn't yield the highest container value, as T parcels are the most valuable and were not used in the exact cover.

Thus, if the parcels are not allocated precise values and the desired goal is to maximise the proportion of the cargo space filled, it can be concluded that the DLX method is more appropriate to maximise space utilisation. However, if the parcels are indeed assigned different values and maximising the total container value is of the highest importance, the greedy approach with the genetic algorithm optimization is a more suitable choice. As the genetic algorithm was trained on a certain sorting option, naturally the best result was achieved at the descending ratio sorting.

6.2 Further studies

Based on the limitations stated in the previous section, we formulated some suggestions for further studies. Further research on which factors exist that would improve the performance of a greedy algorithm when it comes to the O-2DPP might be deemed important. Additionally, investigation on another set of factors is worth exploring. For the exact cover problem, further work could be focused on optimising Algorithm X by finding ways to prune the search tree. Since our genetic algorithm is only trained on a specific sorting, an idea for future research might involve adjusting the genetic algorithm's parameters, such as the mutation rate and investigating whether it is possible to achieve a higher result. Lastly, since the packing problem discussed in this report entails a container with squared edges and packages of the same fashion, we suggest introducing other shapes of parcels and a higher amount of them.

7. References

Côté, J., & Iori, M. (2018). The Meet-in-the-Middle Principle for Cutting and Packing Problems. *INFORMS Journal On Computing*, *30*(4), 646-661. https://doi.org/10.1287/ijoc.2018.0806

de Castro Silva, J., Soma, N., & Maculan, N. (2003). A greedy search for the three-dimensional bin packing problem: the packing static stability case. *International Transactions In Operational Research*, *10*(2), 141-153. https://doi.org/10.1111/1475-3995.00400

Jacobson, L., & Kanber, B. (2015). Implementation of a Basic Genetic Algorithm. In *Genetic Algorithms in Java Basics* (1st ed., pp. 21–45). Apress.

Julstrom, B. (2005). Greedy, genetic, and greedy genetic algorithms for the quadratic knapsack problem. *Proceedings Of The 2005 Conference On Genetic And Evolutionary Computation - GECCO '05*. https://doi.org/10.1145/1068009.1068111

Knuth, D. (2000). Dancing Links. *Millenial Perspectives In Computer Science*, 187-214. Retrieved 17 January 2022, from http://arXiv:cs/0011047.

Mirjalili, S. (2018). Genetic Algorithm. *Studies In Computational Intelligence*, 43-55. https://doi.org/10.1007/978-3-319-93025-1_4

Naeimi, H., & DeHon, A. (2004). A greedy algorithm for tolerating defective crosspoints in nanoPLA design. *Proceedings. 2004 IEEE International Conference On Field-Programmable Technology (IEEE Cat. No.04EX921)*. https://doi.org/10.1109/fpt.2004.1393250

Oracle Corporation. (2021, October 11). *Math (Java Platform SE 8)*. Java™ Platform, Standard Edition 8 API Specification. Retrieved November 13, 2021, from https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html

Pisinger, D. (2002). Heuristics for the container loading problem. *European Journal Of Operational Research*, *141*(2), 382-392. https://doi.org/10.1016/s0377-2217(02)00132-7

Rinnooy Kan, A., Stougie, L., & Vercellis, C. (1993). A class of generalized greedy algorithms for the multi-knapsack problem. *Discrete Applied Mathematics*, *42*(2-3), 279-290. https://doi.org/10.1016/0166-218x(93)90051-o

Russell, S., & Norvig, P. (2011). *Artificial intelligence*. Pearson Education.

Santhi, D. K., & Vinodhini, D. V. (2021). Study on Selection Methods of Parents and Crossover in Genetic Algorithm. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 280–285. https://doi.org/10.32628/cseit217256

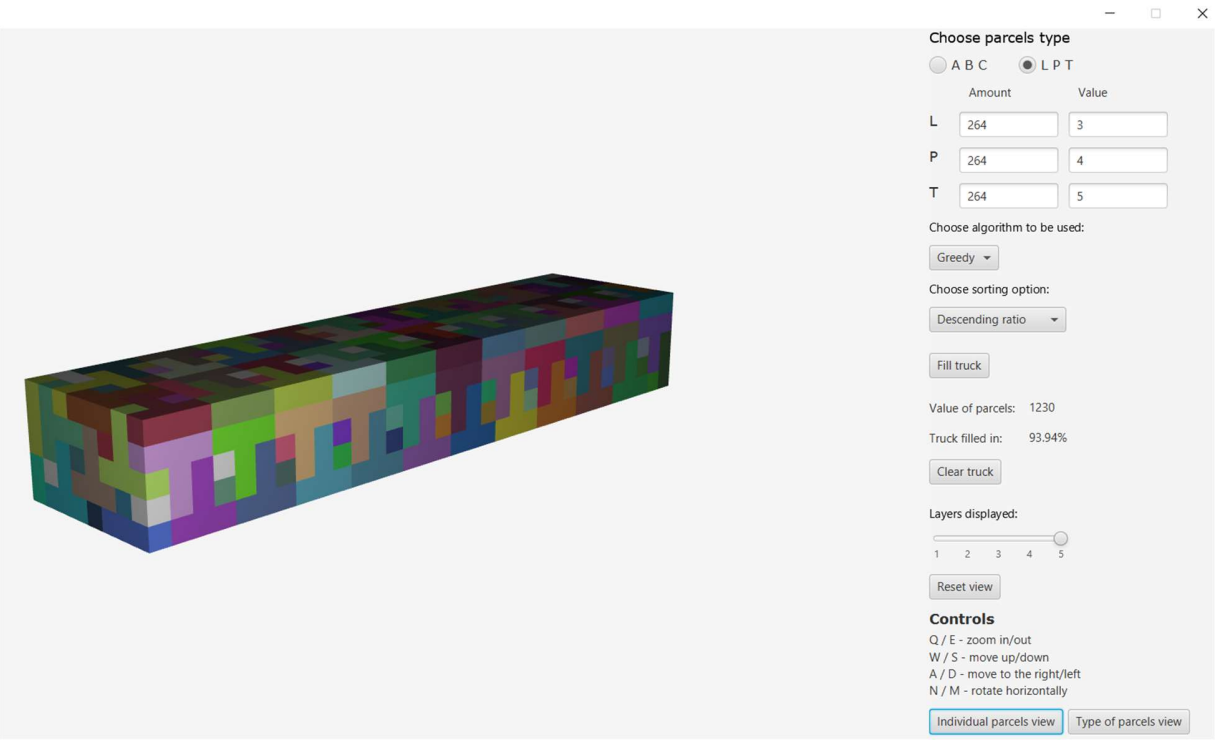
Umbarkar, A., & Sheth, P. (2015). CROSSOVER OPERATORS IN GENETIC ALGORITHMS: A REVIEW. *ICTACT Journal On Soft Computing*, *06*(01), 1083-1092. https://doi.org/10.21917/ijsc.2015.0150

Unwin, A. (2020). Why is Data Visualization Important? What is Important in Data Visualization?. *2.1*. https://doi.org/10.1162/99608f92.8ae4d525

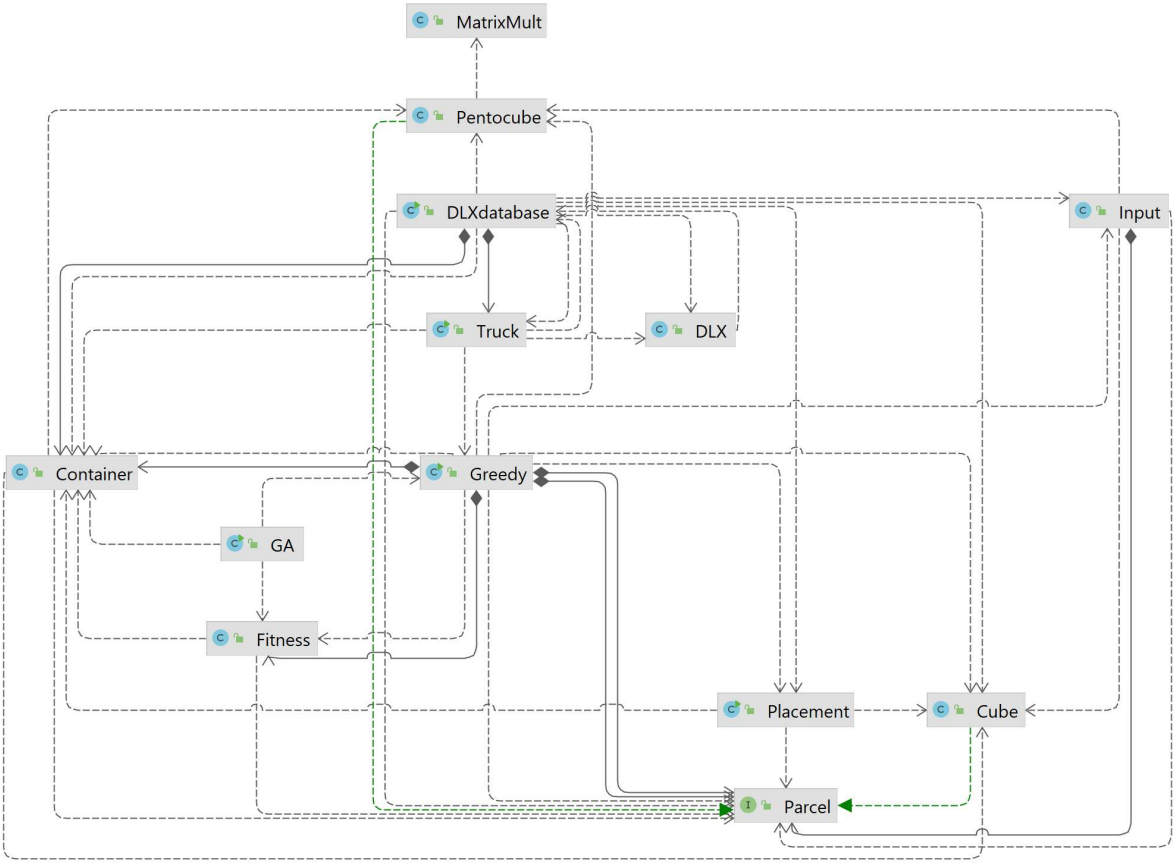
Vince, A. (2002). A framework for the greedy algorithm. Discrete Applied Mathematics, 121(1–3), 247–260. https://doi.org/10.1016/s0166-218x(01)00362-6

8. Appendices

Appendix 1: Visualisation of the solution to the first sub-question for pentocubes.



Appendix 2: UML diagram featuring all classes used to solve 3DPP.



Appendix 3: Extended UML diagram featuring all classes used to solve 3DPP.