**University Maastricht  
Bachelor Data Science and Knowledge Engineering**  
  
**Packing cargo efficiently**Insight into the three-dimensional knapsack problem  
and three approaches to solve it

**Project 1 Phase 3  
Group 8**

**Group members:**  
Adam Eljasiak  
Nicola Gheza  
Daniel Kaestner  
Raffaele Piccini  
Henri Viigimäe  
Simon Wengeler  
  
**Project coordinator:**  
Jan Paredis  
  
**Project examiners:**  
Pietro Bonizzi  
Evgueni Smirnov

Preface  
The following report is the result of a student project that is part of the educational program of the Bachelor Knowledge Engineering at Maastricht University. As such its purpose is in part to comprise the results of said project. In addition it can give insight into possible approaches to solving knapsack problems in a different context. However, due to its primary function, the report only investigates three-dimensional knapsack problems in particular, although the same principles can be applied to similar problems.   
  
Furthermore, it presents the results of experiments involving several algorithmic approaches and evaluates their performance. Therefore, the report mainly offers insight into the advantages and disadvantages of each algorithm and their performance under certain conditions.

Summary  
While the immediate context of the following report is a specific assignment (see 1.1), that particular problem will only be discussed very briefly. Instead the point of focus will be the presentation of different approaches for solving three-dimensional knapsack problems, e.g. the optimisation of packing cargo in a restricted space.  
  
Three algorithms were implemented to solve the knapsack problem given by the project assignment: a greedy approximation algorithm, a hill-climbing algorithm and a genetic algorithm. The idea and implementation behind all three algorithms is mostly based on knowledge acquired previously in the study program Knowledge Engineering. The main purpose of the later described experiments was to test which factors of the algorithms have an impact on their performance and under which conditions that performance is optimal. Furthermore, the experiments serve as a way to determine the overall best approach to the knapsack problem for practical purposes.  
  
To summarise the results, it became clear that the genetic algorithm performs best out of the tested algorithms under the given conditions. It finds a very good solution (presumably the best) in a reasonable amount of time and is thus sensible to put to use in a user-friendly application. However, its performance in terms of computation time lacks significantly for different conditions, especially a larger number of packages. Furthermore, it is restricted to finding near-optimal solutions for rectangular packages only.

Table of contents

[1. Introduction 5](#_Toc440968015)

[1.1 Assignment description 5](#_Toc440968016)

[1.2 Problem definition 5](#_Toc440968017)

[1.3 Structure 6](#_Toc440968018)

[2. Algorithms for the knapsack problem 6](#_Toc440968019)

[2.1 Greedy approximation algorithm 6](#_Toc440968020)

[2.2 Hill-climbing algorithm 7](#_Toc440968021)

[2.3 Genetic algorithm 7](#_Toc440968022)

[3. Implementation 8](#_Toc440968023)

[4. System Guide 8](#_Toc440968024)

[3. Assignment results 8](#_Toc440968025)

[3.1 Using rectangular packages 8](#_Toc440968026)

[3.2 Using pentomino-shaped packages 8](#_Toc440968027)

[4. Experiments and results 8](#_Toc440968028)

[4.1 Principles of evaluation 8](#_Toc440968029)

[4.1.1 Measures of performance 8](#_Toc440968030)

[4.1.2 Comparison between results 8](#_Toc440968031)

[4.2 Change of universal factors 9](#_Toc440968032)

[4.2.1 Package type diversity 9](#_Toc440968033)

[4.2.2 Package and container size 9](#_Toc440968034)

[4.2.3 Specified and unspecified package numbers 9](#_Toc440968035)

[4.3 Greedy algorithm 9](#_Toc440968036)

[4.3.1 Different methods of selection order 9](#_Toc440968037)

[4.3.2 Rotation 9](#_Toc440968038)

[4.3.3 Finding and filling empty space 9](#_Toc440968039)

[4.4 Hill-climbing algorithm 9](#_Toc440968040)

[4.4.1 Varying neighbourhoods 9](#_Toc440968041)

[4.5 Genetic algorithm 9](#_Toc440968042)

[4.5.1 Population size 10](#_Toc440968043)

[4.5.2 Crossover points 11](#_Toc440968044)

[4.5.3 Mutation 11](#_Toc440968045)

[4.5.4 Universal factors 12](#_Toc440968046)

[5. Conclusions 12](#_Toc440968047)

[5.1 Greedy algorithm 12](#_Toc440968048)

[5.2 Hill-climbing algorithm 12](#_Toc440968049)

[5.3 Genetic algorithm 12](#_Toc440968050)

[5.4 Comparison between algorithms 12](#_Toc440968051)

[Appendix A Experiment results 13](#_Toc440968052)

[A.1 Greedy algorithm 13](#_Toc440968053)

[A.2 Hill-climbing algorithm 13](#_Toc440968054)

[A.3 Genetic algorithm 13](#_Toc440968055)

# 1. Introduction

## 1.1 Assignment description

The assignment for the project was to build a computer application with a user friendly interface that can be used for solving so-called three dimensional knapsack problems.

The assumptions are that a company owns trucks with a cargo space of 16.5 m long, 2.5 m wide and 4.0 m high and that it transports parcels of three different types: A, B and C. The sizes of the types are:  
  
 A: 1.0 x 1.0 x 2.0  
 B: 1.0 x 1.5 x 2.0  
 c: 1.5 x 1.5 x1.5  
  
A parcel of a given type also has a certain value, denoted by vA, vB and vC for types A, B and C respectively. The computer application should compute, for a given set of parcels (that may or may not fit into a truck), a packing that maximises the total value.

The application does not have to find the best answer in all cases, but it should be able to find a good approximation. The application should also be able to present a 3D-visualisation of its answers from different perspectives and should be used to answer the following questions (see 3.1 and 3.2):

1. Is it possible to fill the complete cargo space with A, B and/or C parcels, without having any gaps?
2. If parcels of type A,B and C represent values of 3, 4 and 5 units respectively, then what is the maximum value that can be stored in the cargo space?

In addition, after answering the two previous questions, it should be assumed that the company transports pentomino shaped parcels of types L, P and T (see Appendix A, Figure 1), where each of these pentominoes consists of 5 cubes of size 0.5 x 0.5 x 0.5. On the basis of those assumptions the following questions were posed:

1. Is it possible to fill the complete cargo space with L, P and/or T parcels without having any gaps?
2. If parcels of type L, P and T represent values of 3, 4 and 5 units respectively, then what is the maximum value that can be stored in the cargo space?

Beyond these fixed tasks experiments should be conducted which could provide insight into the performance of different algorithms.

## 1.2 Problem definition

The main purpose of the application is to devise an algorithm (or multiple) that maximises the total value of the solution while fitting all the packages within the given space without overlapping, defined as a three-dimensional knapsack problem.

While similar kinds of optimisation or knapsack problems can occur in a wide variety of fields and similar algorithmic approaches to the ones chosen for the purpose of this project may be applicable, this project focuses on the packing of a three-dimensional space. Consequently the algorithms developed during the research are optimised to fill a cargo space of a truck or any similar sort of container.

## 1.3 Structure

Chapter 2 of the report describes the three algorithmic approaches to solve the assigned problem (a greedy approximation algorithm, a hill climbing and a genetic algorithm) as well as some aspects of their implementation in the application that was the result of this project. Chapter 3 gives concise answers to the four individual questions posed by the project assignment (see 1.1) without going into a lot of detail regarding the implication of the results. In chapter 4 several experiments are described in which certain parameters crucial for the performance of the three chosen algorithms are varied, including their results. Lastly, in chapter 5, conclusions are drawn from the previously described results of the experiments.

# 2. Algorithms for the knapsack problem

## 2.1 Greedy approximation algorithm

While not in the form of a three-dimensional knapsack problem, such as the one that is subject of this project report, the idea of a so called greedy approximation algorithm originates from the American mathematical scientist George Dantzig (1957). In his version of the algorithm the items (in this case packages) to be placed in the knapsack are sorted by their value per weight (which is the volume for this problem) and then placed in the knapsack in the resulting sequence.

That same principle is applied in the greedy approximation algorithm. From the packages that are chosen by the user to be placed in the cargo space the ones with the highest value to volume ratio are placed first as long as there is a supply of them. When the supply of packages of the first type is exhausted and there is empty space left, the next type of package will be placed. That process is repeated until all packages have been placed or none of the packages left can be placed anymore.

The placement method employed in the application (both in the greedy and the genetic algorithm) tries to place a new package in the top right front corner of the cargo space and moves on in the sequence if it cannot be placed. From that initial position the package is first moved as far back, then as far left and finally as far down in the cargo space as possible (corresponding to movements along the y-axis, x-axis and z-axis, see Figure 1). Additionally the algorithm will then test whether the package can still be moved in any of the three directions listed above.



Figure 1 – Placement mechanism for the greedy and the genetic algorithm

## 2.2 Hill-climbing algorithm

The hill-climbing search algorithm is simply a loop that continually moves in the direction of increasing value, that is, uphill. It terminates when no successor has a higher value.  
The algorithm starts with an arbitrary solution to a problem and attempts to find a better solution by incrementally changing a single element of the solution. If the change produces a better solution, the previous solution is replaced by the new solution. This process is repeated until no further improvements can be found.[[1]](#footnote-1)

**function** HILL-CLIMBING(problem) **returns** a state that is a local maximum

current ← MAKE-NODE(problem.INITIAL-STATE)

**loop do**

neighbor ← a highest-valued successor of current

**if** neighbor.VALUE ≤ current.VALUE **then return** current.STATE

current ← neighbor

The implemented hill-climbing algorithm starts with an arbitrary solution where the cargo space is filled randomly with packages. Using this first solution a specific amount of neighbouring solutions can be generated by removing a random package in the cargo and trying to fill the remaining empty space with different packages to see whether the value increases. After creating the neighbours a method chooses the best improved solution based on an objective function that aims to maximise the total value of the packing.

In the algorithm that was implemented for the project rotation of the packages can be allowed or prohibited and the neighbourhood size can be changed.

## 2.3 Genetic algorithm

The third algorithm implemented in the program is a genetic algorithm. It evolves chromosomes that represent a solution to the problem which are encoded in an appropriate manner. The encoding for the genetic algorithm implemented for this project is one proposed by Lawrence Davis in 1985. In order to solve a two-dimensional bin packing problem using a genetic algorithm, according to him “the [chromosome representation] that worked best was a simple list of rectangles to be packed. [A] decoding algorithm proceeded by placing the first member of the list into the first place it would fit in the bin […] and so forth”[[2]](#footnote-2) until the chromosome was interpreted in its entirety.

The same principle is applied in the genetic algorithm developed during this project. Chromosomes represent an order of packages. They are placed according to the same placement scheme as the one used for the greedy approximation algorithm (see Figure 1).  
The genetic algorithm uses a “modified crossover”2 in order to retain a favourable order of the packages as well as make sure that only the designated amount of packages of each type are placed. The fitness of each individual is determined by the value that the decoded chromosome amounts to. While the default mutation method is the switching of “genes” (i.e. changing the placement order of the packages), genes can optionally be altered by setting a different state of rotation for the package they represent.

For the purpose of experimenting with the effects of changing vital methods and parameters on the performance of the genetic algorithm, three different selection methods have been implemented in the program. Furthermore it is possible to change the crossover frequency (during reproduction/combination of chromosomes), the frequency of mutation for both variants of it as well as parameters specific to the selection methods.

# 3. Implementation

# 4. System Guide

# 3. Assignment results

## 3.1 Using rectangular packages

The best result in regard to filling the entire cargo space was obtained by the genetic algorithm. It was not able to completely fill the cargo space but only left a single gap with the dimensions 1.5m x 1m x 0.5m. That result was achieved using tournament selection with the following settings: . Since the genetic algorithm was the best performing algorithm that was implemented, the conclusion and answer to the question posed is that the cargo space cannot be filled completely using only packages of type A, B and/or C.

As for the best result achieved by any of the algorithms maximising the value of a packing, the genetic algorithm achieved a value of using the same settings as those described above. Both maximising the occupied space and the value resulted in the same packing.

## 3.2 Using pentomino-shaped packages

Using a brute-force backtracking algorithm a solution could be found in which the entire cargo space is filled with pentominoes. The cargo space is filled with eight layers of pentominoes measuring 16.5m x 2.5m x 0.5m or alternatively 33 layers measuring 0.5m x 2.5m x 4m.

The highest value achieved using the same algorithm was 1120.

# 4. Experiments and results

## 4.1 Principles of evaluation

### 4.1.1 Measures of performance

The most important factor taken into consideration during the evaluation of an algorithm’s performance is its ability to maximise the value of a packing. Additionally, the amount of time it takes the algorithm to compute a solution is considered as well since it has relevance in terms of practicability as an application for actual use. It also gives some insight into the complexity/amount of computations that the algorithm requires to find a solution.

### 4.1.2 Comparison between results

In order to be able to compare individual results all factors that might play into the performance of the algorithms other than the one which is being tested were kept constant. This applies both to the tests done on a single algorithm, e.g. changing the population size for the genetic algorithm, as well as to the comparisons between multiple algorithms.

It should be noted that the number of packages of each available type is “infinitely large” for the purpose of the test (i.e. just enough to fill the entire volume of the cargo space with just one type) for all experiments except if specified otherwise. The same applies to the packages that are used. Packages with different notations and dimensions will be specified as such.

## 4.2 Change of universal factors

### 4.2.1 Package type diversity

In order to properly evaluate the results attained when varying the number of different types of packages that are placed, instead of evaluating the performance in regard to the total achieved value it is evaluated based on amount of empty space left in the cargo space.

### 4.2.2 Package and container size

### 4.2.3 Specified and unspecified package numbers

## 4.3 Greedy algorithm

### 4.3.1 Different methods of selection order

As described in 2.1 one method of choosing the order of placement for the greedy approximation algorithm is to order all packages by their value per volume. Using this method type A (packages are placed first, then type C (and finally type B (packages.

A different method that can be employed is to disregard the value of the packages (considering that they have reasonably similar values per volume) and instead order packages by increasing volume. For obvious reasons this method is only viable in any way if the values per volume of each type of package are reasonably similar. If a package D were to be introduced with the dimensions 0.5m x 2m x 1.5m and a value of vD = 1, it would be placed first due to its low volume of 1.5m3 but would ultimately yield unsatisfactory results (see Figure 3), especially if an infinite supply of all packages were given.

A third possible method is to randomly select the placement order of the packages. In that case, the final result is the best achieved by the algorithm over a certain number of runs.

### 4.3.2 Rotation

### 4.3.3 Finding and filling empty space

A method for finding and filling empty space in which additional packages could be placed is mostly useful for a randomised placement order. Using the randomised method of selection it is a lot more likely that gaps are created when packages of different types are placed adjacent to each other leaving room because of their differing dimensions.

## 4.4 Hill-climbing algorithm

### 4.4.1 Varying neighbourhoods

## 4.5 Genetic algorithm

The selection of individuals that are allowed to reproduce can have an effect on a genetic algorithm’s performance. The three selection methods tested in this project are elitist (EL), tournament (TS) and roulette selection (RO). The first chooses only individuals out of the fittest in the population, the second creates a “tournament” (a part of the population) and chooses the fittest individuals for reproduction and the third assigns certain probabilities to each individual to be selected (higher for fitter individuals).

In the following experiments all tests have been conducted for all three selection methods under identical conditions (as far as it was possible to do so). They were performed using an i5-4690k processor with 3.5GHz clock speed. The average values have been calculated from the results gathered over 25 runs of the algorithm. If not specified otherwise, the following parameters were used: .

It should be noted that in all of the following figures the connections between data points is for clarity and should not necessarily imply that the data gathered contained the additional implied information. Furthermore, all graphs display both the average value achieved by the algorithm (on the left, blue-coloured y-axis) as well as its average runtime (on the right, orange-coloured y-axis).

### 4.5.1 Population size

For any genetic algorithm the population size can have a dramatic effect on its performance. The larger genetic diversity can vastly decrease the number of generations it takes the algorithm to find a solution (as long as there is a definitive solution) or improve the solution given a specific number of generations to run for. At the same time there may be an increase in computation time due to the need to deal with combining, mutating and evaluating more individuals.

As can be seen in Figure 2 with increasing population size the performance of all algorithms in terms of the average value increases. The increase is especially large for the tournament selection method due to the fact that selecting a tournament from a small population (or selecting a very small tournament) results in (almost) random selection. Further proof for the importance of a large enough pool of individuals to choose from is the significant decrease of computation time for the tournament selection method. Since the algorithm either reaches a satisfactory result (in this case a value larger than 230) or runs for the full 1500 generations, this drop-off signifies a much quicker convergence on a good result.

Figure 2 – Effect of population size on the GA’s performance (Appendix A.3, Table 1)

Other experiments performed using more and different types of packages (instead of only 83 A, 55 B and 50 C) saw a significant increase in runtime and a decrease in performance. The latter could be alleviated to some degree by increasing the population size, even though runtimes remained high. This shows the importance of a large genetic diversity if there are more possibilities to order different kinds of packages due to more package diversity. Furthermore, the high runtimes of more than 30 seconds show one of the major disadvantages of genetic algorithms.

### 4.5.2 Crossover points

As would be expected, using no crossovers, and therefore no recombination of individuals, results in lacklustre performance of the genetic algorithm (which can be seen in Figure 3) since the only way of changing the chromosome is mutation. However, an interesting observation is the fact that using only one crossover point barely increases performance, neither in respect to the values achieved nor to the runtime of the program.

Figure 3 – Effect of different numbers of crossover points on the GA’s performance (Appendix A.3, Table 2)

### 4.5.3 Mutation

As described in 2.3 the mutation method used for the genetic algorithm is one that only swaps single genes. As became apparent in the conducted experiments this is crucial for changing the chromosomes between generations enough to produce sufficiently different individuals (see also Appendix B.3, Figure 1). Beyond that, the special “modified crossover” implemented in the algorithm causes an effect that goes beyond copying parts of the parents’ chromosomes and thus “mutates” the children’s chromosomes. Interestingly enough additionally mutating genes by changing the package’s rotation state results in significantly worse performance (see Figure 4).

Figure 4 - Effect of adding other mutation on the GA's performance

This result may be linked to the tendency that the best solutions achieved by the algorithms are usually quite strictly sorted packings of the cargo. Packages of the same type usually occupy continuous areas of the cargo space. In that case rotating a package that would otherwise have been aligned with all of the other packages of its type can significantly reduce the fitness of the resulting individual. Since the previously described tests all used a supply of packages just large enough to fill the cargo space (at least the volume) with each type of package

### 4.5.4 Universal factors

As with the two other algorithms

# 5. 3D-Representation

## 5.1 Creating 3D-Cubes

In order to represent the truck polygons are used. Given an array of packages containing x, y and z position as well as the width, height and length of every package. The program is able to calculate the eight corner points of every package. With this information it creates a cube consisting of six polygons for every face of the package.

## 5.2 Perspective

To avoid a wrong representation the cubes have to be drawn in a specific order. This order is determined by the cubes z-position, where the cube with the lowest z-position is draw first. Being drawn first it will be overdrawn by cubes with a bigger z-position. In addition, since it is possible to see at most three faces of a cube the same sorting principle is applied on the cubes and only the three front faces are drawn.

## 5.3 Rotating

To rotate cube around a specific angle and axis the program applies rotation matrices in the third Dimension. These matrices are applied on every corner and the of a cube position to achieve a good rotation.

# 5. Conclusions

## 5.1 Greedy algorithm

## 5.2 Hill-climbing algorithm

## 5.3 Genetic algorithm

The performance of the genetic algorithms for the specific task of filling the cargo space given by the project assignment with A, B and C packages is the best out of the three algorithms. Using the standard parameters described in 4.4 it finds a very good solution to the specific knapsack problem in a reasonably short amount of time.

However, the conducted experiments have shown that the algorithm (with its current parameters) is most fit to perform that single task. Especially when package numbers increase the computation time increases vastly and the achieved values become less optimal.

## 5.4 Comparison between algorithms

# Appendix A Experiment results

## A.1 Greedy algorithm

## A.2 Hill-climbing algorithm

## A.3 Genetic algorithm

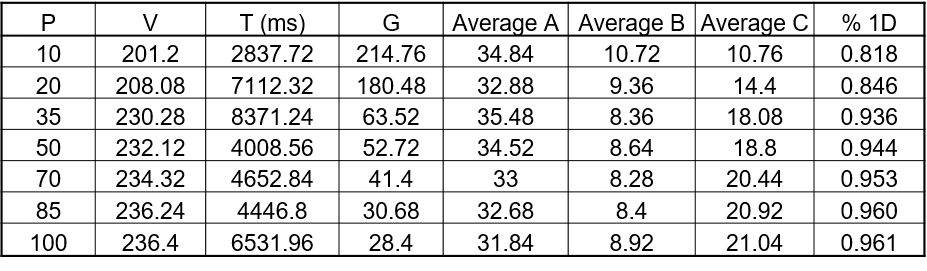


Table 1 – Effect of population size on the GA’s performance

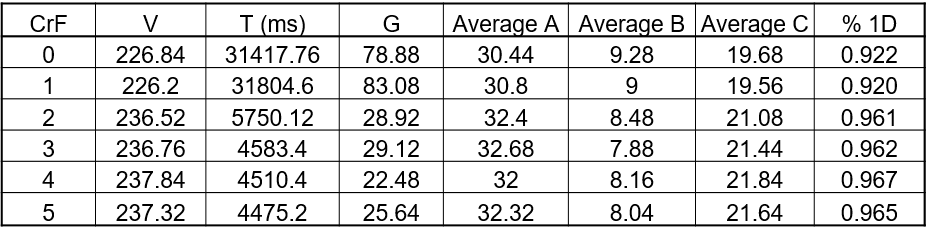


Table 2 – Effect of different numbers of crossover points on the GA’s performance  
CrF = crossover frequency

Figure 5 - Effect of different swap probabilities on the GA's performance

1. Russell, Stuart J., & Norving, Peter (2009). Beyond Classical Search. In *Artificial Intelligence: A Modern Approach* (pp. 120-124). Upper Saddle River, New Jersey: Prentice Hall [↑](#footnote-ref-1)
2. Davis, Lawrence (1985). Applying Adaptive Algorithms to Epistatic Domains. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp.162-164). Los Altos, CA: Morgan Kaufmann Publishers, Inc. [↑](#footnote-ref-2)