**University Maastricht  
Bachelor Data Science and Knowledge Engineering**  
  
**<Title>  
<Subtitle>**

**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 but in addition can give insight into possible approaches to solving knapsack problems in a different context for anyone interested. Yet, due to its primary function, the report will only deal with 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 such algorithmic approaches and judges their performance. Therefore the report can mainly give 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 main focal point will be the presentation of different approaches to the more general challenge: the solving of three-dimensional knapsack problems, e.g. the optimisation of a packing in a restricted space.  
  
The idea and implementation of all three algorithms for which experimental results will be presented in this report is mostly based on knowledge acquired previously in the study program Knowledge Engineering. All of the experiments as well as the measures of performance

Table of contents

[1. Introduction 5](#_Toc440449974)

[1.1 Assignment description 5](#_Toc440449975)

[1.2 Problem definition 5](#_Toc440449976)

[1.3 Structure 6](#_Toc440449977)

[2. Algorithms for the knapsack problem 6](#_Toc440449978)

[2.1 Greedy approximation algorithm 6](#_Toc440449979)

[2.2 Hill climbing algorithm 7](#_Toc440449980)

[2.3 Genetic algorithm 7](#_Toc440449981)

[3. Assignment results 8](#_Toc440449982)

[3.1 Using rectangular packages 8](#_Toc440449983)

[3.2 Using pentomino-shaped packages 8](#_Toc440449984)

[4. Experiments and results 8](#_Toc440449985)

[4.1 Principles of evaluation 8](#_Toc440449986)

[4.1.1 Measures of performance 8](#_Toc440449987)

[4.1.2 Comparison between results 8](#_Toc440449988)

[4.2 Change of universal factors 9](#_Toc440449989)

[4.2.1 Package type diversity 9](#_Toc440449990)

[4.2.2 Package and container size 9](#_Toc440449991)

[4.2.3 Specified and unspecified package numbers 9](#_Toc440449992)

[4.3 Greedy algorithm 9](#_Toc440449993)

[4.3.1 Different methods of selection order 9](#_Toc440449994)

[4.3.2 Rotation 9](#_Toc440449995)

[4.3.3 Finding and filling empty space 9](#_Toc440449996)

[4.4 Hill climbing algorithm 9](#_Toc440449997)

[4.4.1 Varying neighbourhoods 9](#_Toc440449998)

[4.5 Genetic algorithm 9](#_Toc440449999)

[4.5.1 Different generation sizes 9](#_Toc440450000)

[4.5.2 Different selection methods 9](#_Toc440450001)

[4.5.3 Different fitness evaluation 9](#_Toc440450002)

[5. Conclusions 9](#_Toc440450003)

[5.1 Greedy algorithm 9](#_Toc440450004)

[5.2 Hill climbing algorithm 9](#_Toc440450005)

[5.3 Genetic algorithm 9](#_Toc440450006)

[5.4 Comparison between algorithms 9](#_Toc440450007)

[Appendix A Experiment results 9](#_Toc440450008)

[A.1 Greedy algorithm 9](#_Toc440450009)

[A.2 Hill climbing algorithm 9](#_Toc440450010)

[A.3 Genetic algorithm 9](#_Toc440450011)

# 

# 1. Introduction

## 1.1 Assignment description

The assignment for the project on which this report is based 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 make a 3D-visualisation of its answers – from different perspectives.  
  
The application 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 questions above, it should now 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. For 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 all of the tasks above we were advised to conduct experiments of our own once we had one or multiple functioning algorithms.

## 1.2 Problem definition

All tasks given in the project assignment deal with the optimisation of a packing of certain types of packages in a constricted three-dimensional space. Since the main purpose of the application is to optimise the total value of a packing while also fitting all the included packages in the given space, the problem at hand could be defined as a three-dimensional knapsack problem. With the constraints being the dimensions in which all of the packages included in the solution had to fit without overlapping, the main goal of the algorithm(s) to be built was to maximise the total value of the included packages while adhering to the constraints.

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, the problem at hand in particular is focused on the packing of a three-dimensional space. As such the algorithms developed during the research are fit to optimally fill a cargo space of a truck (as is the context of the assignment) 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. The first part of the chapter deals with the variation of aspects of the problem that are applicable to all algorithms, the later parts describe experiments on individual aspects of each algorithm. 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 stems from the American mathematical scientist George Dantzig. 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.

The basic implementation of that principle in our algorithm is the following. 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 tries to place a new package in the top right front corner of the cargo space. (If the package overlaps with a different package in that initial position, it is not considered for any other placement anymore and the algorithm will attempt to place the next package.) 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).



Figure 1

## 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 for the problem at hand starts with an arbitrary solution where the cargo space is filled randomly with packages. From this starting solution a certain amount of neighbouring solutions can be generated by changing a random package in the cargo and trying to fill the remaining empty space with packages as well. After creating the neighbours a method chooses the best next solution based on an objective function that aims to maximize the total value of the packing.

For testing purposes rotation of the packages can be allowed or prohibited. Also we can try starting the algorithm with a solution found by the previously developed greedy algorithm.

## 2.3 Genetic algorithm

The third algorithm implemented in the program is a genetic algorithm. To give a short explanation for readers without any prior knowledge about the principle of genetic algorithms it can be said that such algorithms imitate natural selection. This requires an appropriate encoding of the “genetic” information that defines an individual’s characteristics and thus its fitness. In addition methods for reproduction (involving two parent individuals and a resulting child individual) and mutation have to be provided, two key elements in the evolution of the population of individuals. Furthermore a method to select the individuals allowed to reproduce needs to be implemented. This selection method is usually based on the fitness of individuals for which a fitness evalutation method has to be chosen.

For the genetic algorithm implemented for this project a binary encoding of the “genetic” information of individuals was chosen. The chromosomes of each individual are represented by an array of 0 and 1 values, which are generated randomly for the initial population. Each of these “genes” decides whether a certain package in a certain state of rotation and a certain position in the cargo space is included in the solution. To evaluate the fitness of an individual resulting from that information (i.e. the actual packing in the cargo space), this binary string of numbers is interpreted and converted into a packing. This packing is then evaluated to calculate the fitness of that individual.

For the purpose of experimenting with the effects of changing vital methods and parameters on the performance of the genetic algorithm, several different selection methods and fitness evaluation functions have been implemented.

# 3. Assignment results

## 3.1 Using rectangular packages

Considering the results achieved so far, using mostly the greedy approximation algorithm, there is no definitive answer to the first question posed by the assingment. While it might be possible to fill the entire cargo space (especially using more sophisticated algorithms), it could not be confirmed that it is possible yet.

As for the best result achieved by any of the algorithms so far in terms of maximising the value of a single packing, the highest value achieved over X test runs was Y (just placeholders for now, no real tests done yet).

## 3.2 Using pentomino-shaped packages

Similarly, a definitive answer to the questions cannot be given yet.

However, the packing with the highest value was 912.

# 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 either maximise the value of a packing or minimise the number of gaps left in the cargo space, depending on the desired result.

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. more than 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

Due to the fact that a single result achieved by this algorithm would not necessarily be very representative due to a possible

### 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, 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, instead

### 4.3.2 Rotation

### 4.3.3 Finding and filling empty space

## 4.4 Hill climbing algorithm

### 4.4.1 Varying neighbourhoods

## 4.5 Genetic algorithm

### 4.5.1 Different generation sizes

### 4.5.2 Different selection methods

### 4.5.3 Different fitness evaluation

# 5. Conclusions

## 5.1 Greedy algorithm

## 5.2 Hill climbing algorithm

## 5.3 Genetic algorithm

## 5.4 Comparison between algorithms

# Appendix A Experiment results

## A.1 Greedy algorithm

## A.2 Hill climbing algorithm

## A.3 Genetic algorithm

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)