Solving the Container Shipment Problem with the focus on special goods (dangerous and fragile) using Genetic Algorithms

Walter Zimmer
Department of Computer Science
Cooperative State University
10 Lohrtalweg
Mosbach 74821, Germany
E-mail: walterzimmer@gmx.de

Carsten Mueller, PhD
Department of Computer Science
Cooperative State University
10 Lohrtalweg
Mosbach 74821, Germany
E-mail: carsten.mueller@itg-research.net

***Abstract*—An optimum usage of the shipping space is calculated using genetic algorithms. The objective of the optimization is the maximization of loaded container of type dangerous and the minimization of the unused shipping space. The minimization of the unused shipping space has to be preferred against the maximization of the loaded container of type dangerous. Statistical evalutations were done and the usage of various parameters was analysed.**

I. Introduction

The transport of goods is very important in the ecomomic development. Til now the solving of the container shipment problem is difficult to solve because there are many approaches. There are many port cities worldwide that have to load their ships in an efficient way. The modern global commerce reduces the cost and time of transporting goods. Today we have a lot of automated systems that load container to ships.

What we need is an algorithm which computes the right order to discharge containers [9]. This problem was firstly introduced by Gilmore and Gomory in 1965. A subset of containers of various sizes should be packed into a ship. The ship has fixed dimensions and the target is to optimize the volume of that ship. The container shipment problem is a NP-hard problem and belongs to the *packing problems* problem class. That means there is no polynomial algorithm that solves this problem in acceptable execution time [16].

One approach is to use the backtracking heuristic method to solve this problem. The volume of the packed container will be optimized and it is also possible to optimize the weight distribution. The more variations of containers there are the more difficult the problem is to solve. The mass center of all container is not necessarily near to its geometric center. In this study a weight tolerance of 5% is acceptable. To improve the stability of all containers and the ship another constraint was introduced. Before a container is placed into the ship it must be ensured that at least two third of the area below is covered by other container. In [11] is described how to improve the stability of the ship. In this study a solution of the Container Shipment Problem using Genetic Algorithm (GA) is described. GA belongs to evolutionary algorithms that are part of metaheuristics.

The organization of the paper is as follows: Section II describes the problem statement und some research approaches. An overview of the solution architecture is given in section III. A proof of concept is mentioned in section IV where a solution of a genetic algorithm is compared with a solution that was generated through a monte carlo evaluation. Section V concludes the work.

II. Problem Statement and Research Approach

A. Problem Statement

The first aim is to maximize the amount of containers or minimizing the waste of space. This can be solved by using the backtracking method combined with heuristics. The second goal is to maximize the amount of *dangerous* container that should be loaded between two container with goods type *standard*. This will be done by using a Genetic Algorithm [1]. The ship has the following dimensions (see table I)

Table I
Dimensions of the ship

length	178 m
width	33 m
height	20 m

The ship contains two halves that are seperated by a one meter wide wall. The wall is 178 m long and 20 m high. After the loading of the ship both ship sides should be balanced. A tolerance of 5% should not be exceeded. In [15, p. 238] a load balancing using a Q-learning algorithm is described. Furthermore it is not allowed that container go beyond the walls of the ship. There are three container types ($C1$, $C2$ and $C3$). Each container type could have the goods type *dangerous* or *standard*. The table II below lists the three container types:

Table II
Container types

Container type	Dimensions (length, width, height)
$C1$	6 m x 2.50 m x 2.50 m
$C2$	12 m x 2.75 m x 2.50 m
$C3$	14 m x 3.25 m x 2.50 m

B. Research Approach

There are three methods to solve the container shipment problem: *exact methods*, *heuristics* and *hybrid methods*.

Exact Methods are used to solve small problems, because there are a large number of constraints. The aim is to find the optimal (best) solution. But the disadvantage is that this method requires a lot of computational resources.

The second approach to solve this problem is the use of heuristics. The most known heuristic is called *wall building heuristic*. This method creates layers (horizontal strips) and arranges the container by reducing the problem to a 2D packing problem. There are backtracking algorithms that solve the 2D packing problem effortlessly. Another approach is to build segments (instead of layers) which could be rotated and interchanged to improve the weight distribution. Metaheuristics (e.g. tabu search) achieve good solutions in reasonable time, but they do not guarantee the best solution. This work is based on solving the container shipment problem by using Genetic Algorithms. Containers are arranged in stacks. There are 8 layers of stacks because the ship height is 20 meters and each container has a height of 2.5 meters. A chromosome consists of many stacks and each genome can be interchanged with an other genome to improve the weight distribution. The disadvantage of heuristic methods is that they do not guarantee to find the best solution. On the other hand this methods have a better execution time in contrast to exact methods.

The third approach is to use hybrid methods and combine algorithms with metaheuristics [4]. A combination of multiple approaches is always better than a single approach. A second hybrid method is the combination of heuristic methods and local search methods. It has to be said that each algorithm is better for a specific case of problem and worse for other problems.

Heuristic Backtracking methodology

The heuristic backtracking methodology consists of two phases that will be described below:

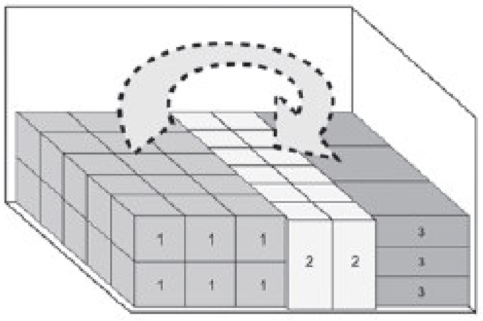
Phase I: Maximizing packed volume

To maximize the packed volume the wall building heuristics will be combined with a *backtracking search algorithm* to chose the best order. The result is a build tree solution. Container are placed recursively into the ship in such a way as to create blocks. If this algorithm would only accept optimal solutions then good solutions would be discarded. Therefore it is acceptable when small waste of space occurs [**.**]

Phase II: Optimize weight distribution

In the second phase the optimal order of container will be determined using **Genetic Algorithms**. The advantage of this project is that the height of each container is the same. Therefore eight layers fit into a ship to reach a height of 20 meters. There layers could be rotated to improve the weight distribution. Another approach is to generate blocks of containers that can easily be interchanged to optimize the weight distribution. The volume size will be the same, because blocks with the same volume are interchanged. Furthermore a set of blocks can be rotated clockwise or counterclockwise (see figure 1). The rotation direction is determined randomly.

Figure 1. Rotation of multiple blocks with different volume.



The value 1 could mean, that the blocks are rotated clockwise and the value 0 stands for counter-clockwise rotation. The fitness function of the weight optimization can decide whether the reached result is good or not. The distance between the geometric center of the ship and the container's center of mass is calculated and then compared. The ship has a high fitness value if the packed volume is hight and the container's center of mass is near the center of the ship. The below shown fitness function is used to determine the fitness value of the second optimization:

$$f = \sqrt{(x_{gc} - x_{cm})^2 + (z_{gc} - z_{cm})^2}$$

$$x_{cm} = \frac{\sum_{i=1}^{n} x_i \cdot w_i}{\sum_{i=1}^{n} w_i}; \; z_{cm} = \frac{\sum_{i=1}^{n} z_i \cdot w_i}{\sum_{i=1}^{n} w_i}$$

The geometric center of the ship is (x_{gc}, z_{gc}) and (x_{cm}, z_{cm}) is the center of mass of all container. The center of one container is (x_i, z_i) and w_i is the weight. In the end both fitness values are added and the total fitness value is calculated.

III. Solution architecture

In this section first the software architecture will be described. A class diagram shows the interaction between the used components. It improves the comprehension of the reader and also internalizes the structure of the implementation. The second part describe the Genetic Algorithm and its components. The third part shows some diagrams of the parameter analysis.

A. Software architecture

The figure 2 shows a class diagram of the used classes. The genetic algorithm is started in the `Application` class in the `main()` method. In the configuration (`Configuration.java` you can set a two boolean variables to indicate whether a already saved configuration and ship should be loaded or not. If these variables are set to `true` then the content of a CSV file will be loaded by the class `CSVLoader.java`. After the loading process a visualization of the ship will open. Otherwise a new population of ships is generated. Every ship has a set M_C of containers that are created inside a loop. After the creation of the whole population the genetic algorithm starts and begins to improve the population in small steps. In this process the strategies that were set in the configuration class are applied. They are repeated in the following order:

1) Order Selection
2) Uniform Recombination
3) Displacement Inversion Mutation
4) Elitism Selection

The recombination and mutation process will be performed randomly. An external generator of random numbers (class `MersenneTwister.java`) is used in this process. A container can have the following container types: C1, C2 or C3 (class `ContainerType`). These types as well as the dimensions of the container are put into an *enum*. Furthermore a container can be of type *dangerous/fragile* or *standard*. Here another *enum* is used (enum `GoodsType`).

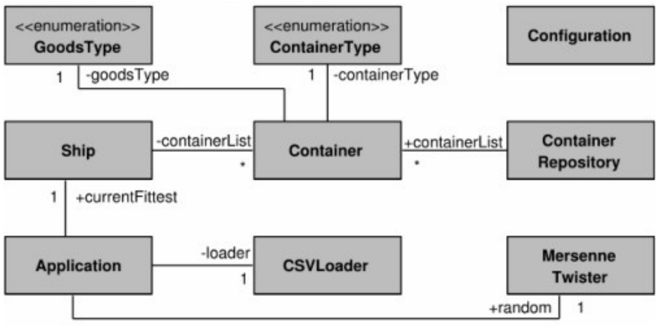


Figure 2. Software architecture

B. Genetic Algorithm

In this section the genetic algorithms are described. First a definition will be given and afterwards the components will be described. Furthermore the building block hypothesis will be mentioned.

A genetic algorithm is a technique that is used to find the most valuable individual in a generation that consists of many individuals. The individuals compete against each other which results in an increase of the fitness of the population. In figure 3 and 4 you can see that individuals converge against the maximum fitness value. In the first picture the solution candidates are spread over the whole x- and y-axis. The the second picture they accumulate at the peaks of the graph. After one generation has passed only the fittest solution candidates move to the next generation. One generations consists of a recombination and a mutation process.

Figure 3. Increase of the fitness values of individuals.

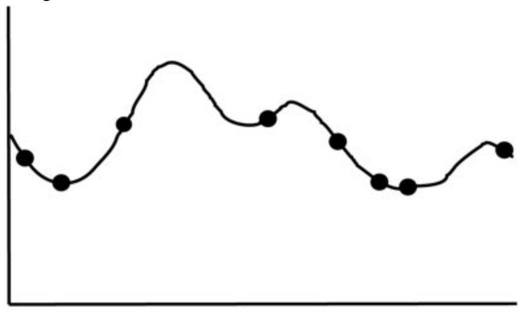
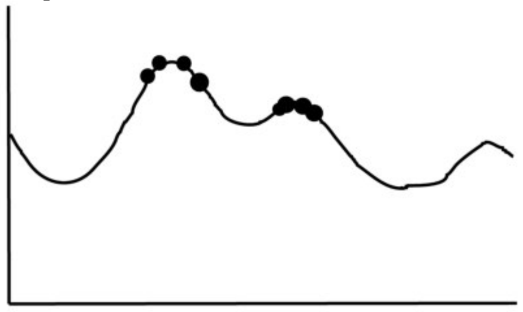


Figure 4. Increase of the fitness values of individuals.



The recombination process is applied to two or more individuals (parents) and produces two new candidates (children). The mutation process is applied only to one individual and the result is a modified individual. All solution candidates are sorted by their fitness values and after that it is checked if a better candidate was generated.

The mutation and recombination in a generation is responsible for the variety of a generation. The last process in the loop is the selection of the fittest individuals. This step improves the quality of the generation. One approach is not use choose every time only the fittest individuals. It can be of advantage to choose a low amount of weak individuals to avoid the creation of local optima and the stagnation of the fitness. In the following pseudo code the process of a generation is described:

```
BEGIN
    INITIALIZATION of a population with random
solution candidates;
    EVALUATION of each candidate;
    ITERATE until termination condition is reached
        1. SELECTION of parents;
        2. RECOMBINATION of parents;
        3. MUTATION of individuals;
        4. EVALUATION of new candidates;
        5. SELECTION of new candidates
for the next generation;
END
```

Flow chart for Genetic Algorithm is illustrated in figure 5.

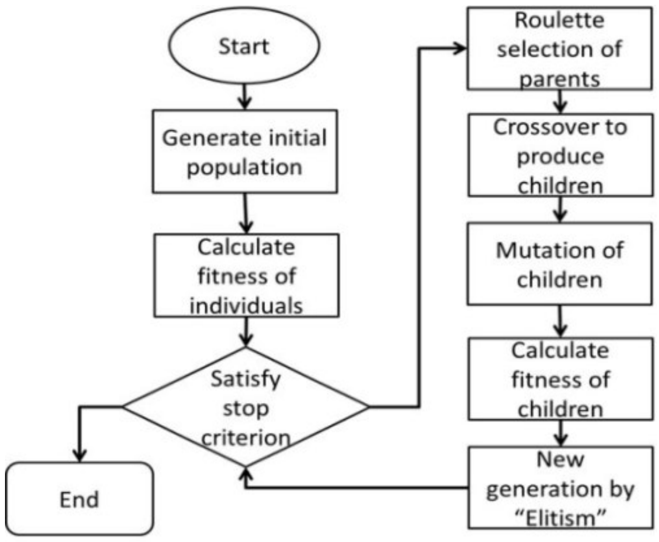

Start
Generate initial population
Calculate fitness of individuals
Satisfy stop criterion
End
Roulette selection of parents
Crossover to produce children
Mutation of children
Calculate fitness of children
New generation by "Elitism"

Figure 5. Illustration of the genetic algorithm

After the algorithm has started, the initial population is generated. The population size in this work has been set to 100. The fitness value of each individual is determined and then the termination condition will be proved. Is the termination condition satisfied a loop will be entered and the evolutionary process is started. For parent selection the 'roulette wheel selection' is chosen and the 'uniform recombination' method is used to cross both parents and generate new children. After that the children were mutated by using the 'displacement inversion mutation' that will be described in the next section. Lastly the fitness value of each individual is calculated and a new generation is chosen applying the 'elitism method'. Is the number of defined generations reached the termination condition is satisfied and the algorithm ends.

You can image the process of the genetic algorithm as a building-block-hypothesis. The chromosome blocks are used by genetic algorithms. They are shuffled to increase to increase the fitness value of that individual. With a higher fitness value that individual has a higher possibility to be chosen for pairing. This hypothesis states the improvement of individuals by assembling different blocks together. Next the components of the genetic algorithm that were used in this thesis will be described in detail:

- Individuals
- Fitness function
- Population
- Initialization
- Parent selection
- Operators (recombination and mutation operators)
- Survivor selection
- Termination condition

1) Individuals: A population consists of a set of individuals. An individual is represented by a genotype and a phenotype. The genotype determines the inner and the phenotype the outer structure of an individual. In case of saving a individual as an integer, the integer will represent the phenotype of that individual. The binary representation of the integer makes up the genotype. The evolutionary search process takes place on the genotype level. In phenotype space individuals are called solution candidates and in genotype space they are knowns as chromosomes. The components of a chromosome are the genes. Genes are also known as variables, positions or loci. An object at such a position is the value or allele. The transformation from a genotype to phenotype is called decoding. The transformation vice versa is called encoding. Each genotype has a corresponding phenotype [6].

2) Fitness function: The fitness function represents the prerequisites, that a population has to fulfil. It is used in the survival selection process to determine the fittest individuals. In the implementation of the survival function a function called elitism selection is used. This means that only the best individuals are selected for the next generation. The aim of the genetic algorithm is to optimize the volume of the ship (amount of containers that are loaded onto the ship) as well as the amount of dangerous containers. The optimization of the volume of the ship has a higher priority than the amount of dangerous container. Therefore it is weighted with the factor 3. The amount of dangerous container is weighted with the factor 2. Both fitness values are summed together to create the final fitness value of the individual. In equation 1 the calculation of the fitness function is shown.

$$f_{total\ best} = 3 \cdot V_S + 2 \cdot num_{dangerous\ container} \tag{1}$$

V_S represents the volume of the ship and $num_{dangerous\ container}$ the number of dangerous container that are loaded onto the ship. The aim is to generate individuals with high fitness values. This principle can be inverted so that the lowest fitness value is searched [5, p. 20].

3) Population: A population is made up of individuals. There can be multiply identical individuals in one population. The maximum population size is set in the configuration and the default value is 100. To speed up the genetic algorithm the population size can be set to two individuals. After each evolutionary cycle the best individuals are determined that build up the next generation. Another option is to determine the worst individuals and remove them from the population. The remaining individuals build up the next generation. The variety of a population is determined by the different individuals in a population. Another statistical value is given by the entropy that indicates that there is no order in the population [6].

4) Initialization: In the initialization phase the population is filled with randomly chosen individuals from the container repository. One approach is to use heuristics to create a start population with a high fitness value. The disadvantage of this approach is that the algorithm takes more time before the actual genetic algorithm is started. In this thesis the maximum runtime of the algorithm is set to five minutes. The right balance between a heuristic initialization and a random initialization must be found. In [6] is mentioned that an intelligent heuristic is not necessary because the genetic algorithm is capable of reaching a high fitness value after a few generations. In figure 6 you can see the increase of the fitness value of a population that depends on the time.

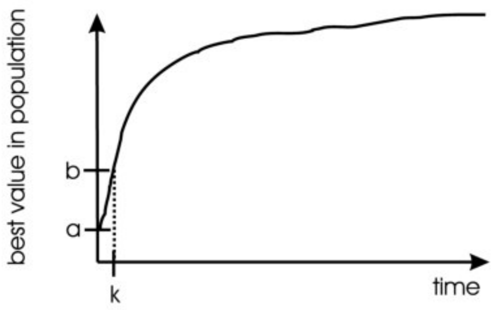


Figure 6. Visualization of the fitness value in the evolutionary process. An intelligent heuristic is not mandatory because a random generated population (level a) can reach the plateau of a population (level b) that was generated using a intelligent heuristic after *k* time is passed.

5) Parent selection: The aim of the 'parent selection' is to chose individuals for pairing. Before this method is applied all individuals of the population are sorted after their fitness values. After that always two individuals are chosen for pairing beginning with the fittest individuals. The first one will be paired with the second one and so on. If two individuals with high fitness values are paired the children will have a high fitness value too. Therefore there is the possibility to generate better individuals. Individuals with a low fitness value will not be chosen for pairing. But there will be a small probability for choosing individuals with a small fitness value to prevent a drift of the population into a local optima.

6) Crossover strategy: The recombination operator is used to create new individuals. The crossover strategy in this work is 'uniform recombination'. Given two chromosomes of individuals new chromosomes are generated by swapping each genome if the probability is below 0.5^1. The evolution process on earth has shown that a recombination of individuals has led to high structured organisms.

The below figure 7 illustrates the 'uniform recombination':

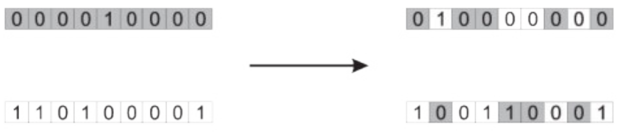


Figure 7. Uniform Recombination'.

At the first genome the probability was over 0.5 and therefore this genome was not swapped. To use this operator on the ship to exchange two series of container, the exchanged container have to be rearranged. The recombination operator is applied to the indices of container in the container repository. This list of indices is stored in the ship. Without rearranging the container after the recombination process the new fitness value can not be calculated. In [14, p. 682] is mentioned that the operators of genetic algorithms lead to an disruption of building blocks of containers. An alternative would be to use a *estimation distribution algorithms-based hyper-heuristic* (EDA_HH). First the gene distribution is estimated and then new individuals are sampled from that distribution. Finally it has to be said that the crossover strategy has to adapted to the type of the chromosome. Is the chromosome build up of bits, each bit has to be mutated in the mutation phase. This operator has to be adapted to the problem statement to use it efficiently [2].

7) Mutation strategy: The mutation operator is always applied to only one individual. Mutation of genomes is used to modify randomly chosen genomes. Displacement inversion mutation cuts a randomly chosen part of the chromosome, inverts the order and inserts this part at a randomly chosen position. In figure 8 you can see the displacement inversion mutation:

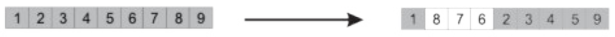


Figure 8. Displacement Inversion Mutation.

The mutation operator is important in order to avoid local optima, which are areas of the search space that appear to be optimum, but are actually just isolated by neighboring solutions of a less desirable nature [7, p. 44].

8) Survivor Selection: In contrast to 'parent selection' that is stochastic, the 'survivor selection' is deterministic. There are multiple variants to apply 'survivor selection' to the individuals.

The selection criterias are summarized below:

- Elitism selestion
- Age-based replacement
- Fitness function value only
- Age-based replacement together with fitness function value

[1]The probability range is between 0.0 and 1.0 and a random number is generated by using the `MersenneTwister`.

The first opportunity is to use the 'elitism selection' to select only the individuals with the highest fitness values. In [3, p. 15] it is advised to keep the best 10% from the previous generation. Another possiblity is to keep 2-5% of individuals in a population and to do n copies of them [7, p. 66].

The second variant determines the age of an individual. Individuals that are very young will have a higher selection rate than older individuals.

The third 'survivor selection' method is the 'random wheel selection'. The higher the fitness value of an individual the higher the probability to be chosen. Each individual has the probability p_i to reach the next generation. The below formula describes the probability p_i:

$$p_i = \frac{f_i}{\sum\limits_{j=0}^{n} f_j}$$

n is the amount of all individuals in the population and f_i stands for the fitness value of the current individual. This fitness value is divided by the sum of all fitness values. In [12, p. 119] is mentioned that using both parent and survivor selection can cause premature convergence to a local optimum. Therefore using the third selection criteria supports escaping from local optima and to track dynamic ones.

9) Termination condition: There are a lot of termination conditions that can be used in a genetic algorithm. The algorithm can terminate if a certain fitness value is reached. Is this fitness value very high, it can not be reached and the algorithm will never stop. Therefore a second termination condition is used and records the time that is passed since the algorithm was started. In this thesis a runtime of five minutes was suggested. Another method to terminate the algorithm is the achievement of a certain number of evaluations on individuals. If there is no improvement of the fitness value after x generations the algorithm will end too. The fifth method to terminate the algorithm is the lack of diversity of individuals in a population.

In the implementation of the termination conditions all opportunities are combined together with a logical `OR`. Therefore one condition is enough to abort the genetic algorithm. The last possibility to terminate the algorithm to close/kill the application (e.g. the user closes the window or presses the stop button). In table III all six termination conditions are summarized.

Table III
SUMMARY OF ALL TERMINATION CONDITIONS OF THE GENETIC ALGORITHM.

Termination condition	**Explanation**
`maxGenerations`	Max. number of generations is reached.
`maxExecutionTime`	Max. time is up.
`solutionFound`	Solution candidate found.
`noImprovementAfterXGenerations`	No improvement after x generations.
`lackOfDiversity`	A lack of diversity in population.
`forceTermination`	User forced to terminate the application.

10) Parameter: The parameter that are used in the above mentioned components of the genetic algorithm are shown in table IV. For parent selection the strategy 'Order Selection' was chosen. After that a 'Uniform Recombination' with the probability of 70% will be performed on the population. The next step of the genetic algorithm is the mutation. In this part the 'Displacement Inversion Mutation' with a probability of 25% is applied. The last section is the selection of the fittest individuals. Here the 'Elitism Selection' method is used to select the individuals with the highest fitness value.

Table IV
THE PARAMETER THAT ARE USED BY THE GENETIC ALGORITHM.

Volume of ship in [m^3]:	99.12
Volume of ship in [%]:	112,920.00
Dimensions of ship [in m]:	178.00, 33.00, 20.00
Weight difference [in %]:	0.00
Dangerous container amount/ Total container amount:	912/2352
Recombination:	Uniform Recombination
Recombination probability:	0.70
Mutation:	Displacement Inversion Mutation
Mutation probability:	0.25
Parent Selection:	Order Selection
Survival Selection:	Elitism Selection
Population size:	100
Number of needed generations:	0
Processed time [in sec]:	300
Termination condition:	max. execution time

The statistical parameter analysis was performed with the software `R`. A combination of all strategies was tested. For each component of the genetic algorithm two strategies were chosen. The impacts of the applied strategies were compared together. For the statistical parameter analysis a recombination probability between 0.4 and 0.7 was chosen. The probability of the mutation strategy was varied between 0.0001 and 0.001. Twenty different variations were created and the results were compared. A good approach is it to use 'Order Selection' together with 'Uniform Recombination' because the parameter analysis often evaluated results with this combination as significant. In [13] a process of adapting mutation operators is described. The parameter tuning is an exhaustive process and is problem specific. Tuning of parameters can lead to a higher performance and enables the algorithm to escape local optima more efficiently. [13] mention that it makes sense to combine multiple genetic operators adaptively.

IV. Proof of concept

To provide a proof-of-concept (poc) the genetic algorithm is compared with a monte carlo evaluation. The well-known Monte Carlo approach is basically an iterated generation of random solutions [10, p. 9]. In this section a complex case will be described to verify the performance of the genetic algorithm. After a lot of experimental work the practical interest of the algorithm will be shown. This section also describes how and why the genetic methods work. Ships with the dimensions of 24m length, 16m width and 5m height will be generated and the fitness values will be recorded 50 minutes long. In table V you can see the configuration and the fitness value that was found after a runtime of 50 minutes. 5543 generations were generated after 50 minutes. Afterwards the result of the genetic algorithm was compared with a monte carlo evaluation with 50000 ships that took about 50 minutes too. In table VI you can see the fitness values that were generated by the genetic algorithm. In the first column you see the corresponding generation with a 500 steps size. Furthermore the first two generations and the last generation is listed in the table.

Table V
Configuration of the genetic algorithm to provide a proof of concept.

Fitness	4530.50
Volume of ship [in %]	83.75
Volume of ship [in m^3]	1507.50
Ship dimensions [in m]	24.00, 16.00, 5.00
Weight difference [in %]	1.14
Dangerous container amount/ Total container amount	4/39
Free space volume [in %]	16.25
Free space volume [in m^3]	292.50
Recombination	UniformRecombination
Recombination probability	0.70
Mutation	DisplacementInversionMutation
Mutation probability	0.005
Parent Selection	OrderSelection
Survival Selection	ElitismSelection
Evaluate	true
Evaluation	MonteCarlo
Maximum Number Of Evaluations	50000
Population size	2
Number of needed generations	5543
Processed time [in sec]	3000
Termination conditions	maxExecutionTime

Table VI
Listing of the fitness values of the genetic algorithm with a step size of 500.

Generation	**Fitness value**
1	3,041.25
2	4,384.00
...	
3500	4,418.00
4000	4,418.00
4500	4,444.00
5000	4,530.50
5500	4,530.50
5543	4,530.50

After the first generation the fitness value increases to 4,384.00 and stays constant up to the 3500th generation. To compare the fitness values a monte carlo evaluation with 50000 ships was created (table VII).

Table VII
Result of the monte carlo evaluation with a runtime of 50 minutes.

Number of evaluations:	50,000
Maximum fitness value:	4,434.50

The maximum fitness value that was generated is 4,434.50. Therefore the genetic algorithm performs better than the monte carlo evaluation. Besides a better fitness value the genetic algorithm took less time for generating fitness values. Even after the 4500th generation (about 45 minutes of runtime) a better fitness value was found. The monte carlo evaluation took about 50 minutes and was not able to generate a higher fitness value. Therefore a proof of the performance of the genetic algorithm is given.

V. Conclusions

In this section some benchmarks will be described and also the advantages of genetic algorithms. They show the quality of the reached solutions. The computational results were obtained using an Intel i7-2720QM (@2.2 GHz) and 16 GB (@667 MHz) RAM. The *Container Shipment Problem* with only three different container types belongs to the weak heterogeneous *Container Shipment Problem*. The Genetic Algorithm was tested with *VisualVM* and *Mission Control*.

The following figure 9 shows the whole ship loaded with *standard* (dark gray) and *dangerous* (light gray) container types.

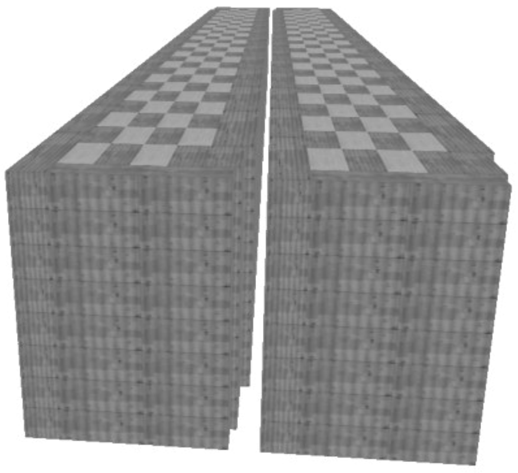


Figure 9. Visualization of the generated ship.

The ship has the below shown dimensions (table VIII) and contains two halves that are separated by a one meter wide wall. The wall is 178m long and 20m high.

Table VIII
DIMENSIONS OF THE SHIP

length:	178 m
width:	33 m
height:	20 m

Finally the advantages of genetic algorithms are summarized. There are some problems mentioned in [8] that are good candidates to solve with genetic algorithms.

- Problems where it is hard to find a solution but once a solution is found, it can be measured.
- Problems where the search space is very large, complex and poorly understood.
- Problems where a near optimal solution is acceptable.
- Problems where no mathematical analysis is available.

All four Problems fit to the *Container Shipment Problem* and therefore genetic algorithms are the best choice to solve this problem. *"Three billion years of evolution can not be wrong. It [genetic algorithm] is the most powerful algorithm there is."* [Goldberg, [7]]

REFERENCES

[1] Luiz Araujo and Placido Pinherio. *Genetic Algorithms for Semi-Static Wavelength-Routed Optical Networks*. Ed. by Olympia Dr. Roeva. INTECH Open Access Publisher, 2012. ISBN: 978-953-51-0146-8.

[2] Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, eds. *Handbook of Evolutionary Computation*. 1st. Bristol, UK, UK: IOP Publishing Ltd., 1997. ISBN: 0750303921.

[3] Eduard Burke, Nam Pham, and Rong Qu. "Learning the heuristic distribution by an evoluionary hyper-heuristic". In: (2014), p. 40. URL: http://www.slideserve.com/mikayla-osborne/learning-the-heuristic-distribution-by-an-evolutionary-hyper-heuristic.

[4] Irina Dumitrescu and Thomas Stützle. "Applications of Evolutionary Computing: EvoWorkshops 2003: EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, and EvoSTIM Essex, UK, April 14–16, 2003 Proceedings". In: ed. by Stefano Cagnoni et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. Chap. Combinations of Local Search and Exact Algorithms, pp. 211–223. ISBN: 978-3-540-36605-8. DOI: 10.1007/3-540-36605-9_20. URL: http://dx.doi.org/10.1007/3-540-36605-9_20.

[5] Agoston Eiben. *Introduction to evolutionary computing*. New York: Springer, 2003. ISBN: 9783540401841.

[6] Agoston Eiben. *Introduction to evolutionary computing*. Berlin: Springer, 2015. ISBN: 3662448734.

[7] Adam Janiak and Maciej Lichtenstein. *Advanced algorithms in combinatorial optimization*. 2011.

[8] JGAP. "Introduction to Genetic Algorithms with JGAP". In: (2009). URL: http://jcraane.blogspot.de/2009/02/introduction-to-genetic-algorithms-with.html.

[9] Marc Levinson. *The box : how the shipping container made the world smaller and the world economy bigger*. Princeton, N.J. Woodstock: Princeton University Press, 2006. ISBN: 978-0691136400.

[10] Ana Moura and José Fernando Oliveira. "An integrated approach to the vehicle routing and container loading problems". In: *OR Spectrum* 31.4 (2009), pp. 775–800. ISSN: 1436-6304. DOI: 10.1007/s00291-008-0129-4. URL: http://dx.doi.org/10.1007/s00291-008-0129-4.

[11] David Pisinger. "Heuristics for the container loading problem". In: *European Journal of Operational Research* 141.2 (2002), pp. 382–392.

[12] Kenneth Price. *Differential evolution : a practical approach to global optimization*. Berlin New York: Springer, 2005. ISBN: 978-3-540-20950-8.

[13] A. Prokopec and M. Golub. *Adaptive mutation operator cycling*. 2009. DOI: 10.1109/ICADIWT.2009.5273969. URL: http://www.zemris.fer.hr/~golub/clanci/icadiwt2009_atga.pdf.

[14] Rong Qu et al. "Hybridising heuristics within an estimation distribution algorithm for examination timetabling". In: *Applied Intelligence* 42.4 (2014), pp. 679–693. ISSN: 1573-7497. DOI: 10.1007/s10489-014-0615-0. URL: http://dx.doi.org/10.1007/s10489-014-0615-0.

[15] Sani Tijjani and Ihsan Omur Bucak. "An approach for maximizing container loading and minimizing the waste of space using Q-learning". In: *2013 The International Conference on Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE)* (2013). DOI: 10.1109/taeece.2013.6557277. URL: http://dx.doi.org/10.1109/TAEECE.2013.6557277.

[16] Gerhard Wäscher, Heike Haußner, and Holger Schumann. "An Improved Typology of Cutting and Packing Problems". In: *An Improved Typology of Cutting and Packing Problems* 24 (Jan. 16, 2006), p. 46.