

Generating timetables for the barges of the Rotterdam port using genetic algorithms

WA de Landgraaf

supervised by M. Schut

for the AI faculty of the Vrije Universiteit, Amsterdam

1 - Introduction

The port of Rotterdam is one of the major ports in the world. In 2002, the port handled over 300 million tons of goods from all over the world and the major port in the EU when it comes to throughput. The majority of goods are transported using large sea-faring ships, of which a portion use containers. In 2002, 4 million containers were transferred in Rotterdam, with a total of 50 million tons in weight.[1]

These containers are transferred in Rotterdam from large sea-faring container ships onto barges for transportation further land-inwards, at a number of terminals. In the current situation, barge operators and terminal operators both try to match their schedules in order to have an efficient continuation of the throughput of the port. The terminal operator handles the large sea ships and the separate barges at a single terminal, of which there are 15 in the Rotterdam area. The barge operator has as his goal to transfer a certain amount of containers, using a number of barges, from one terminal to another and to finally have these containers transferred to ports further inland.

The problem, however, is that the current way of making timetables gives a large number of problems. Currently, a barge visits about 5 to 6 terminals out of 8 terminals which it was assigned to, in 22 hours. A large portion of the daily schedule is taken up by waiting for terminals to process different barges and sea ships. This is further complicated by the fact that the barges need to move between the terminals and often are delayed. These delays cause a cascade of delays for further barges throughout the day, which in turn cause delays in shipments and lower the overall efficiency of the port.

The goal of this bachelor project is to investigate if this timetabling problem can be solved using a genetic algorithm. A number of simulations are run using various amounts of data in order to determine the effectiveness and hopefully open a way to improve the scheduling of barges in the port of Rotterdam.

2 – Real-world case

In the port of Rotterdam, the allocating of barges is done by the barge operator. The barge operator receives a number of terminals at which must be loaded or unloaded. After deciding which barges will handle which moves, requests are sent to the terminal operators who either accept or decline due to space/capacity restrictions.

With our application, each of the barge operators submit both the barges and the moves to the application, after which the application generates a fair and as optimal as possible planning for the barges, the barge operators and the terminals. If any conflicts show up (it is impossible to have a barge at a specific terminal within the selected period) , these are reported to both the barge operators as well as the terminal.

3 – Scope

This project was limited to the use of evolutionary algorithms, and genetic algorithms in particular, which were very suitable for the large amount of optimization necessary. Genetic algorithms have been used in other domains for scheduling and generating time tables, often very successfully.

As this project is only concerned about the theoretical usability of genetic algorithms for the case described above, only a simulation is made to solve the problem. Certain factors not directly relevant to the timetabling problem are ignored for the experiments or left for further development.

4 – Representation

The nature of the input data was fairly complex and consisted of a number of different dimensions. An overview of the data can be found in Appendix A, supplied by Martijn Schut. As this is a simulation and a proof of concept, not all data types will be used.

What is included:

- **Move** a number of containers must be loaded or unloaded at a specific terminal
- **Barge operator** has a number of barges under his control, receives a number of moves that he must fulfill
- **Barge** performs a series of moves selected by the barge operator
- **Terminal** fixed docks for loading or unloading, can handle

one or more barges at a time

Besides the input data there are also a number of constraints:

- **Sections** amount of space per terminal
- **Distances** between terminals, not used in this simulation
- **Sea ship entries** 'immutable moves'

The raw data however must be converted to a specific XML format. An example is laid out below (shortened where necessary):

```
<?xml version="1.0" encoding="iso-8859-2"?>

<harbour>
  ... [parameter and info tags]

  <subjects>
<!--      First Barge Operator -->
      <subject title="S1" barge="Anne"/>
      <subject title="S1" barge="Bella"/>
<!--      Second Barge Operator-->
      <subject title="S2" barge="Danielle"/>
      <subject title="S2" barge="Erica"/>
  </subjects>

  <building>
    <terminal id="T1">
      <capability type="capability-place">1</capability>
    </terminal>
    <terminal id="T2">
      <capability type="capability-place">1</capability>
    </terminal>
  </building>

  <timetable>
    <class name="Test Timetable BO-1"/>
      <subject title="S1" terminal="T2" min="1"/>
      <subject title="S1" terminal="T2" min="1"/>
      <subject title="S1" terminal="T1" min="1"/>
    </timetable>

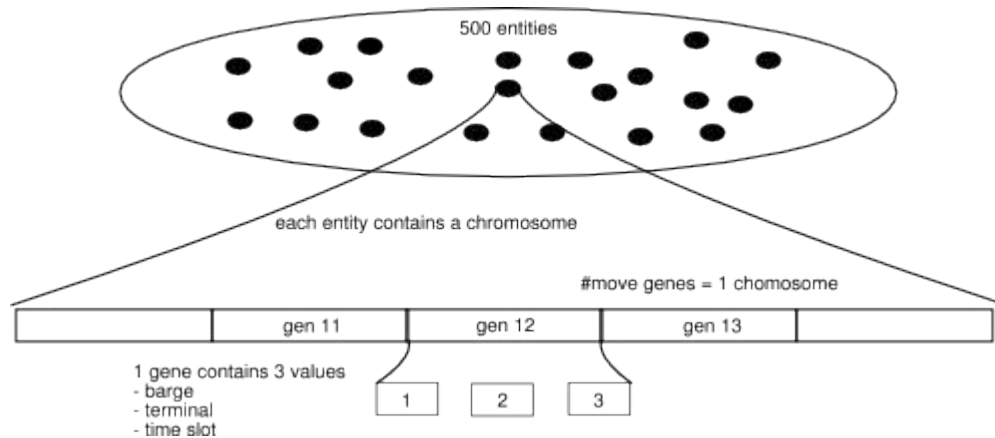
    <timetable>
      <class name="Test Timetable BO-2"/>
        <subject title="S2" terminal="T1" min="1"/>
        <subject title="S2" terminal="T1" min="1"/>
        <subject title="S2" terminal="T1" min="1"/>
    </timetable>
  </harbour>
```

This example data specifies two barge operators, each with two barges, two terminals and each barge operator has a series of 3 moves. More options are possible, but this example shows the format in which the raw data must be converted in order to be usable by the

GA. For a complete Input/Output example see Appendix B

This data is then represented in a the following way: For each of the individuals, all moves are represented in a chromosome. Each chromosome has a gen, also known as a tuple, for each move defined in the input data. This gen has the following data:

- timeslot
- terminal
- barge



Each number is an integer between 1 and the number of slots, terminals or barges. In this way, each gen describes a possible move, with each individual containing a chromosome that describes a complete series of possible moves, a possible timetable.

The output from the GA is the same XML file as the input, but with an extra section of assignment data, for example:

```
<chromo grade="18">  
  <gen period="5" day="0" terminal="T2" barge="Anne"/>  
  <gen period="5" day="0" terminal="T1" barge="Bella"/>  
  <gen period="7" day="0" terminal="T2" barge="Bella"/>  
  <gen period="9" day="0" terminal="T1" barge="Erica"/>  
  <gen period="8" day="0" terminal="T1" barge="Danielle"/>  
  <gen period="10" day="0" terminal="T1" barge="Danielle"/>  
</chromo>
```

These types of output files can then be used for generating other output formats, like HTML. In this case, a barge planning of a single day of the 6 moves would be shown, with each slot being occupied according to the time, location and barge that has been chosen to handle the move. An example HTML output table is shown below, indicating per column a time slot and per row a barge. The T indicates the terminal where the barge is at.

	1	2	3	4	5	6	7	8	9	10	11	12
0-Anne					Anne T3							
1-Bella						Bella T2	Bella T2	Bella T3				
2-Corien					Corien T2	Corien T3	Corien T1	Corien T1				
3-Danielle						Danielle T1		Danielle T2				
4-Erica							Erica T3		Erica T3	Erica T1		
5-Froukje					Froukje T1					Froukje T2		

5 – Implementation

The timetable is generated for a number of periods (hours), the experiments have been done for a single day but moves can be extended to cover a week. Due to the problem, 3 variables need to be stored in every gen to cover a move: a time, a terminal and a barge.

For this project, we started off with Tablix [2], a reasonably well-known open source timetable-generating tool for colleges. A large number of changes were necessary before the tool was able to use the above data types, but the modular structure of the application was retained so that it can be expanded for future applications.

Tablix is set up in a flexible way, so that the fitness/grading functions can easily be expanded and weights can easily be adjusted. Tablix uses an XML format for it's input and output, easing integration with other projects.

The population is set at 500 individuals for the experiments, we enable the following operators:

- **Crossover** for every generation, a new generation is made by exchanging half of every entity with the half of another entity ($\lambda = 1$)
- **Mutation** we randomly modify parts of a gen for a number of entities in every generation (12 / generation, $\mu = 0.024$) and randomly randomize all values in a gen (6 / generation, $\mu = 0.012$)

Also, local search is applied when the grades of the population remains stagnant over a large number of generations (over 100). This operator searches the immediate surroundings of the search space for a better fitness (if the time slot would be 5, search both 4 and 6 and reevaluate the fitness). Being computationally very expensive this is only used in the latter part of the GA, allowing the algorithm to fine-tune its results.

The fitness function is made up of separate grades functions, weighed differently in order to describe the most important aspects of the generated output. With the implemented fitness function a lower fitness means a better fit, thus the task is minimizing the fitness of the entities. There are 6 currently enabled:

- Sametime – mandatory for completion, avoids overlap between moves (weight of 100)
- Sametimeroom, sameroom – mandatory for completion, make sure a move is set for the right terminal (weight of 200 / 10)
- GoodBO – mandatory for completion, make sure the right barge operator is being used for the move (weight of 70)
- MaxTime – total number of time each barge is active, incl. holes between moves (weight of 1)
- Holes – total number of holes of the whole solution (weight of 6)

The weights were set and calibrated through trial and error. Using these grades, the following constraints are set:

- a barge doesn't have multiple moves in the same period
- a terminal doesn't have more barges in the same period
- a barge operators moves are assigned across only the owned barges

In addition to these constraints, the total time for each barge is optimized and the time for all moves on a whole is optimized. Holes in a barges roster gives the chromosome a higher score (that is, less optimized). Physically impossible moves are further penalized. This equation results in a grade for every chromosome in the population. The grade influences the crossover operator and lower-grade

chromosomes are removed from the population.

After a fixed number of generations, or when all mandatory fitness-modules are fulfilled, the chromosome with the best fitness is converted to the native XML format Tablix uses. Using the adapted output tools, tables and reports can be generated from the output files.

6 – Experimentation

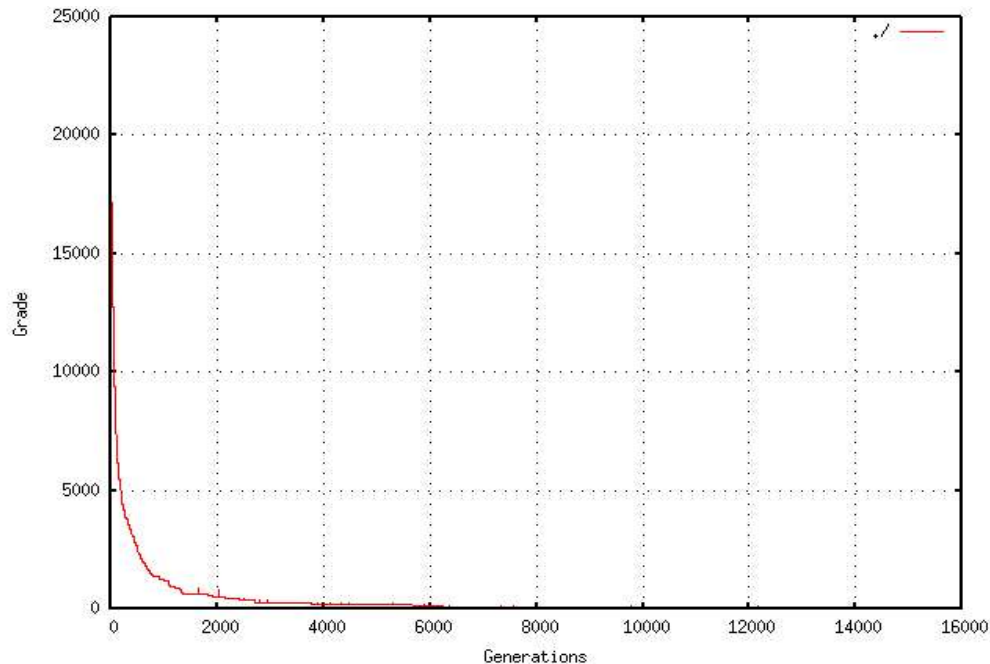
Two full-scale experiments are performed using the criteria displayed above with a number of entities comparable with that in the port of Rotterdam. The weights defined in the previous section are fixed for all trials.

These experiments have been done on a single AMD 2200 PC with 1GB of RAM running Debian GNU/Linux. A rough estimate during experiment 2 gives a CPU usage of 95% and memory usage of 0.2% of the process. It is interesting to note that Tablix uses a clustering framework so that computation could be spread out over multiple nodes.

Experiment 1 is done with:

- seven barge operators, with 3 barges each
- seven terminals
- 98 moves (4 to 5 per barge)
- 24 periods

This experiment resembles the real-life case scenario. The total runtime took about 12 minutes to complete the 15000 generations that has been set as the maximum for the run. In order to complete the first 4000 generations (which gives a satisfactory result) the algorithm took little over 3 minutes as can be seen in the results later on:

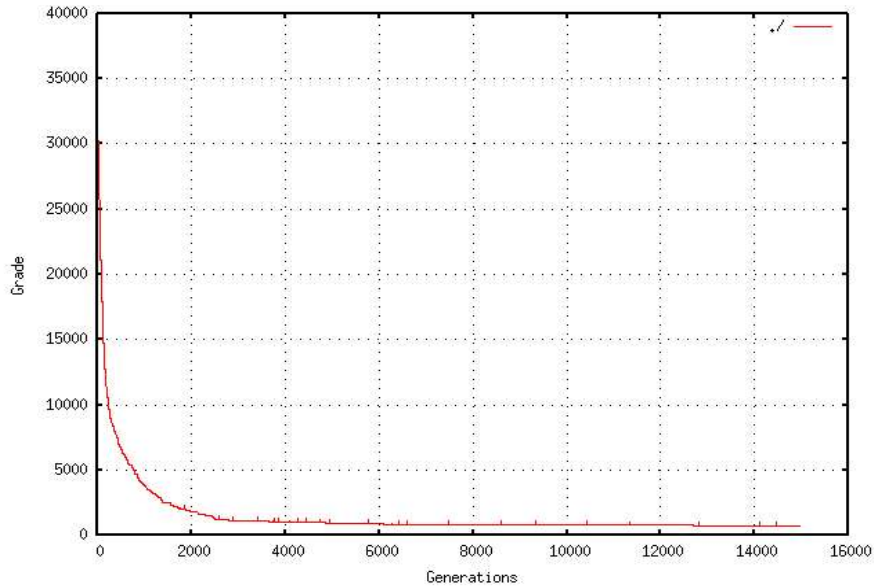


Notice that the biggest improvement happens in the first few thousand generations: if a quicker result is necessary performing a run using 2000-4000 generations would give a very reasonable result, after that time the GA is merely fine-tuning its results. The generated output table after 15000 generations can be found in Appendix C.

Experiment 2 is done with:

- seven barge operators, with 3 barges each
- seven terminals
- 147 moves (7 per barge)
- 24 periods

With this experiment we simulate a larger number of moves using the same number of barges and terminals. This shows how a busier port influences the scheduling. The time required for the 15000 generations was nearly 25 minutes, more than doubling the time required compared with the first experiment. The time required to reach the 4000 generation mark is little over 9 minutes in this experiment.



A much higher grade is the result of the increase in moves. A more gradual decline of the optimal grade is a result of a smaller amount of slots available for the barges; there are even a couple of double-bookings at terminals, as not all barges were allocated correctly within the 15000 generations.

In the following table the nature of the grades decent of the two experiments is shown more precisely, with the grade and the time in seconds after starting the algorithm.

Generation	Exp1	Exp2
1	0s - 23816	0s - 37491
100	5s - 8021	11s - 19336
500	25s - 2707	57s - 6829
1000	49s - 1294	118s - 3919
1500	73s - 670	176s - 2556
2000	98s - 541	226s - 1800
2500	122s - 385	277s - 1351
3000	146s - 311	362s - 1416
4000	194s - 305	546s - 1188
5000	243s - 239	883s - 931
14999	742s - 73	1473s - 787

Complete per-generation statistics can be downloaded for further processing [3].

6 – Conclusion

A global conclusion is that this case is a good example of a problem that can be solved using genetic algorithms. In spite of growing complexity, the application scaled reasonably well and is capable of reaching a decent grade fairly quickly. The application hasn't been rigorously fine-tuned or optimized for speed, it is possible that a reasonable part of the runtime duration could be dealt with.

It was unfortunately unable to compare a real-world timetable with a generated one, due to port authority policies. This would be a true test for the application. In this conclusion we must note that the timetables generated are merely simulations of the real-world case. There are a number of factors that have been neglected for this experiment and could be improved if further development is viable:

- Distance between terminals

Although the speed of barges has been implemented, nothing is done in the experiments with this data. This in turn causes terminals to be selected not on distance to the previous terminal of a barge but on merely the terminals that fit all the moves best. Implementing this would complicate the application considerably, however this would make real-world usability very close. It would group moves together near terminals that are close to each other.

- Docking of sea-faring container ships

It is possible to include immutable moves, however these have not been used in the above experiments for simplicity. Although a minor factor, adding these ships would cause a higher grade.

- multiple barges per terminal per timeslot

In the experiments each terminal can only handle one barge at the same time. This isn't the case in the real world, as certain terminals can handle multiple barges simultaneously. Also, nothing has been done with the number of containers for a specific move.

- other output translations

Currently only a html table is produced containing all moves, next to the default XML output. Using the XML output file, it is possible to easily add output types for terminals, barges and barge operators.

References

[1] Statistics available on www.portofrotterdam.com

[2] Tablix: <http://freshmeat.net/projects/tablix>

[3] All code, data, this paper and the generated reports can be found on:

<http://www.alextrime.org/projects/rotterdam-ga>

Appendix A – data overview

Created by Martijn Schut:

** ALGEMENE INFO **

Tijdspanne: 48 uur, van 22-04-2004 0:00 tot 23-04-2004 23:59

Aantal terminals: 9

Aantal barges: 22

** INPUT **

Barges:

SHIPID - nummer van de barge

ETA - estimated time of arrival (at brienenoord)

ETD - estimated time of departure (at brienenoord)

(Verschil tussen ETA en ETD is gemiddeld zo'n 17-24 uur.)

Terminals:

TERMINAL - naam vd terminal

(we veronderstellen altijd beschikbare kadecapaciteit en voldoende mankracht)

Moves:

MOVEID - uniek ID voor move

TERMINAL - betrokken terminal

TYPE - laden of lossen

SIZE - aantal betrokken containers in MOVE) (varieert 5-25)
(per TERMINAL zijn er ongeveer 12-18 MOVEs)

** OUTPUT **

TRIPID - uniek ID voor deze omloop

TERMINAL - te bezoeken terminal

LOAD - aantal te laden containers (varieert van 5-25)

UNLOAD - aantal te lossen containers (varieert ook van 5-25)

DURATION - tijd dat barge bij de terminal is

STARTTIME - aankomsttijd bij terminal

NEXT_TERMINAL - geplande volgende terminal

VAARTIJD - vaartijd tussen TERMINAL en NEXT_TERMINAL

Appendix B – input/output example files for the genetic algorithm

Experiment-input.xml:

```
<?xml version="1.0" encoding="iso-8859-2"?>

<harbour>
  <info>
    <title>Sample Rotterdam</title>
    <address>sample.xml</address>
    <author></author>
  </info>

  <parameters>
    <module name="sametime.so" weight="100" mandatory="yes"/>
    <module name="sameroom.so" weight="200"
      mandatory="yes"/>
    <module name="goodbo.so" weight="70" mandatory="yes"/>
    <module name="sametimeroom.so" weight="10"
      mandatory="yes"/>
    <module name="maxtime.so" weight="1" mandatory="no"/>
    <module name="teacher_holes.so" weight="6" mandatory="yes"/>
    <internal name="impossible" weight="55"/>
  </parameters>

  <!-- title is barge/timetable owner, name the unique barge descrip-->

  <subjects>
<!--      <subject title="S1" name="BO-1"/>-->
    <subject title="S1" barge="Anne" speed="20"
      eta="12" etd="17"/>
    <subject title="S1" barge="Bella" speed="30"
      eta="13" etd="18"/>
    <subject title="S1" barge="Corien" speed="15"
      eta="14" etd="19"/>
<!--      <subject title="S2" name="BO-2"/>-->
    <subject title="S2" barge="Danielle" speed="15"
      eta="14" etd="19"/>
    <subject title="S2" barge="Erica" speed="15"
      eta="14" etd="19"/>
    <subject title="S2" barge="Froukje" speed="15"
      eta="10" etd="20"/>
  </subjects>

  <building>
    <terminal id="T1">
      <capability type="capability-place">1</capability>
    </terminal>
    <terminal id="T2">
      <capability type="capability-place">1</capability>
    </terminal>
    <terminal id="T3">
      <capability type="capability-place">1</capability>
    </terminal>
  </building>
</harbour>
```

```
        </terminal>
<!--      <terminals start="1" end="40"/>-->
</building>

<timetable>
  <class name="Test Timetable BO-1" year="1"/>
    <subject title="S1" terminal="T3" min="1"/>
    <subject title="S1" terminal="T2" min="1"/>
    <subject title="S1" terminal="T1"
min="1"/>
    <subject title="S1" terminal="T2" min="1"/>
    <subject title="S1" terminal="T3"
min="1"/>
    <subject title="S1" terminal="T1"
min="1"/>
    <subject title="S1" terminal="T3"
min="1"/>
    <subject title="S1" terminal="T2"
min="1"/>
  </timetable>

<timetable>
  <class name="Test Timetable BO-2" year="1"/>
    <subject title="S2" terminal="T1"
min="1"/>
    <subject title="S2" terminal="T1"
min="1"/>
    <subject title="S2" terminal="T3"
min="1"/>
    <subject title="S2" terminal="T1"
min="1"/>
    <subject title="S2" terminal="T2"
min="1"/>
    <subject title="S2" terminal="T2"
min="1"/>
    <subject title="S2" terminal="T3"
min="1"/>
  </timetable>
</harbour>
```

Experiment-output.xml

(the data above is included, with the following additions:

```
<chromo grade="18">
  <gen period="5" day="0" terminal="T3" barge="Corien" uid="1"/>
  <gen period="5" day="0" terminal="T2" barge="Anne" uid="2"/>
  <gen period="5" day="0" terminal="T1" barge="Bella" uid="3"/>
  <gen period="6" day="0" terminal="T2" barge="Corien" uid="4"/>
  <gen period="8" day="0" terminal="T3" barge="Corien" uid="5"/>
  <gen period="7" day="0" terminal="T1" barge="Corien" uid="6"/>
  <gen period="6" day="0" terminal="T3" barge="Bella" uid="7"/>
  <gen period="7" day="0" terminal="T2" barge="Bella" uid="8"/>
  <gen period="9" day="0" terminal="T1" barge="Erica" uid="9"/>
  <gen period="8" day="0" terminal="T1" barge="Froukje" uid="10"/>
  <gen period="9" day="0" terminal="T3" barge="Froukje" uid="11"/>
  <gen period="10" day="0" terminal="T1" barge="Froukje" uid="12"/>
  <gen period="9" day="0" terminal="T2" barge="Danielle" uid="13"/>
  <gen period="8" day="0" terminal="T2" barge="Danielle" uid="14"/>
  <gen period="7" day="0" terminal="T3" barge="Erica" uid="15"/>
</chromo>
```

Appendix C – generated tables from output files

(see included PDF documents for tables of experiment 1 and 2)