Creating an Optimal NS Controller Schedule to Maximise Fine Revenue

Bas de Gooijer (2520781)

Under supervision of
Prof. dr. Rob van der Mei

July 11, 2014

NS pakken zwartrijders aan

Spoorwegen gaan controles maximaal opvoeren ✱ Personeel voelt zich onvoldoende toegerust

[illegible]

Contents

1 Introduction

The Dutch Railways (popularly known as the “NS”) is the primary railway transporter for people in the Netherlands. The NS transports 1.1 million passengers per day, spread over 4500 train rides [1]. While most of these passengers purchase a ticket, some choose to go on the train without one. This is possible because there is no ticket control when entering or leaving the train. However, NS controllers occasionally control passengers’ tickets (in the train itself) and hand out fines if a correct ticket cannot be presented. Due to the apparent lack of structural ticket control, the NS may be missing out on revenue generated by fines. It may also be seen as an encouragement for passengers to not buy a ticket, because it is not certain if they will be controlled.

While the NS mentions a general strategy to minimize illegal train journeys in their annual report [2], there is little (public) knowledge on how NS controllers should be placed to maximise fine revenue. By using security cameras and gates, and employing security officers in and around the stations, the NS aims to provide a safe travelling environment and reduce illegal train journeys. However, little attention is paid to the *route* of controllers. This seems like a missed opportunity, because an effective placement of controllers can easily be quantitatively measured by looking at the fine revenues.

In this paper we construct an optimal strategy for the routing of a set of NS controllers to maximise fine revenue. This is done by implementing two different algorithms. A formal definition of the problem is given below. Throughout this paper, we discuss strengths and weaknesses of the two algorithms, and discuss their validity.

2 Model

2.1 Parameters

When choosing the parameters for our model, attention is paid to limiting the number of variables, while also allowing the model to be flexible. For instance, adding new stations or railway connections should be easy, as well as possible changes in the train schedule. Another important issue to consider is that some trains only travel between large cities, and do not stop at any smaller stations that lie inbetween. Also, it is important to have a flexible input of all the obtainable revenue.

2.1.1 Connections c_i

A connection c_i directly connects two stations in one direction, and the connections are arbitrarily indexed c_1 to c_N. Each connection has a fixed travel time. This system means that there is no need to index the stations - the location of a controller can be described by the last connection the controller took. If we consider the layout to be a graph, we only consider the edges and *not* the vertices. To explain the system, we present the following simple train station network consisting of 5 stations:

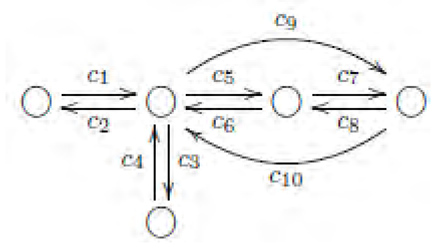


Figure 1: Simple example of a possible train network

Each connection c_i marks where a train leaves and arrives. That means there is a difference between taking c_9 and taking c_4 with c_8, because a stop is made inbetween. Note that two connecting stations each have a connection in opposite directions. This is to clearly distinguish where a controller is when going on a connection. If the controller's last connection was c_2, we know he is now in the leftmost station.

2.1.2 Adjacency Matrix A

We introduce an "adjacency" matrix A, consisting of points a_{ij}:

$$a_{i,j} = \begin{cases} t & \text{if } c_j \text{ can be taken directly after } c_i\text{: the travel } t \text{ time of } c_j \\ 0 & \text{otherwise} \end{cases}$$

Visually, one can see which connections are available to a controller, given his location. However, an algorithm cannot distinguish this. The adjacency matrix A for the above setup could for instance be:

$$A = \begin{matrix} 0 & 1 & 1 & 0 & 2 & 0 & 0 & 0 & 4 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 2 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 3 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 2 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 & 2 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 4 \\ 0 & 1 & 1 & 0 & 2 & 0 & 0 & 0 & 4 & 0 \end{matrix}$$

If we look at the first row, we know that *after* taking c_1, we can go on connection c_2, c_3, c_5 and c_9. The first two connections' travel time is 1, whereas c_5 has travel time 5 and c_9 has travel time 4. Given a total of N connections, matrix A will have $N \times N$ dimensions. Note that the j-th column gives the travel time of c_j. Also, many rows will be identical; approaching a station from c_1 and c_4 gives the same possibility of follow-up stations. The benefit of this approach is that only one matrix is required to describe which connections are available, and what their travel time is.

2.1.3 Revenue Matrix R

The revenue matrix R consists of points r_{it} which tell how much revenue can be generated by taking connection c_i at time t. Throughout this paper, time is left as a discrete unitless quantity. The dimension of R is $N \times T$ where N is the total number of connections and T is the entire period of time the trains run. A typical i-th row could look like:

$$100 \quad 0 \quad 80 \quad 0 \quad 100 \quad 0 \quad 60 \quad 0 \quad 0$$

This means over a period of 9 time units, a train runs 5 times on the i-th connection. The 0's means there is no train departing at time $t = 2, 4, 6, 8$ and 9. An assumption of R is that every train ride has revenue. The motivation behind this is that we can then capture departure times of trains *and* their revenue in one matrix.

Another important distinction is that any revenue from a connection is only counted once. If two controllers take the same connection, they cannot both take the revenue - only one of them will receive it. This makes sense because the NS is only interested in the total revenue, not in the amount of revenue generated by each controller.

2.1.4 Controller Strategy $C_k(t)$

The decision variable $C_k(t)$ is the connection the k-th controller *takes* at time t, for $k = 1, \ldots, K$ and $t = 1, \ldots, T$. The output is either the index number of the connection or 0. In case of the latter, this indicates the controller is either waiting at the station or *en route* in a train. Using the initial example from Figure 1, if the first controller waits one time unit, takes connection c_6 (which has travel time 3) and c_2 immediately afterwards, we have:

$$\begin{aligned} C_1(1) &= 0 \\ C_1(2) &= 6 \\ C_1(3) &= 0 \\ C_1(4) &= 0 \\ C_1(5) &= 2 \\ &\ldots \end{aligned}$$

Of course, this is only possible if there is a train running on c_6 and c_2 at time 2 and 5, respectively. This is checked in R. If R_{62} and R_{25} are positive then this is a possible strategy.

2.1.5 Controller Schedule S

The controller schedule is a $4 \times K$ matrix, containing information on the starting and finishing time and location of each controller, where K is the number of controllers. The first and second row indicate the first and final time a controller can take a connection. The third row contains an (optional) connection that *arrives at* the start location. The fourth row contains an (optional) connection that *departs from* the end location. If there is no scheduled location, this is indicated with a 0. This manner of specifying the start and end location may seem odd, but it makes the programming much more efficient. Consider the following two controllers, and their schedule for the layout from Figure 1:

$$S = \begin{array}{cc} \text{Controller 1} & \text{Controller 2} \\ 1 & 2 \\ 8 & 7 \\ 9 & 0 \\ 8 & 1 \end{array}$$

The first controller works from time 1 to 8. The controller also wishes to start and finish at the rightmost station. The second controller works a shorter shift (from time 2 to 7). This controller has no scheduled starting location, but does need to end at the leftmost station. The elements of S are denoted with $S_{i,k}$, as is used in the formal optimization problem below.

2.2 Formal Optimization Problem

The formal optimization problem is given below. We are given the matrices A ($N \times N$ dimension) and R ($N \times T$ dimension). For the given number of controllers K, we are given a schedule matrix S ($4 \times K$ dimension).

With our decision variable $C_k(t)$ for $k = 1, \ldots, K$ we formulate the optimization problem as follows:

$$\text{Maximise the revenue:} \quad \sum_{k=1}^{K} \sum_{t=1}^{T} R_{C_k(t),t} \text{ under the conditions:}$$

$$A_{C_k(t),C_k(t')} > 0 \quad \forall k, t \tag{1}$$

$$R_{C_k(t),t} > 0, \quad \text{if } C_k(t) \neq 0 \quad \forall k, t \tag{2}$$

$$A_{\cdot,C_k(t^{**})} \leq T + 1 - t^{**} \quad \forall k \tag{3}$$

$$C_k(t) = 0, \quad t < S_{1,k}, \ \forall k \tag{4}$$

$$C_k(t) = 0, \quad t > S_{2,k}, \ \forall k \tag{5}$$

$$A_{S_{3,k},C_k(t^*)} > 0, \quad \text{if } S_{3,k} \neq 0, \ \forall k \tag{6}$$

$$A_{C_k(t^{**}),S_{4,k}} > 0, \quad \text{if } S_{4,k} \neq 0, \ \forall k \tag{7}$$

where $C(t), C(t')$ are two consecutive non-zero values, t^*, t^{**} are the time of the first/last non-zero entry of $C_k(t)$ respectively, and $A_{\cdot,j}$ is the non-zero value in the j-th column (each column has either value 0 or one other number).

Condition (1) means that once a controller has taken a connection, the next connection must be physically attached to where the controller ended. Condition (2) means that when a controller goes on a connection, there must be an actual train riding on that connection. Condition (3) means that a controller must stay within the bounds of the time period $1 \ldots T$. For instance, it is not possible to take a connection with travel time 4 if the controller's schedule ends in 3 time units. Conditiona (4) and (5) mean that a controller can only take trains if these are within his time schedule. The final conditions (6) and (7) mean that a controller must depart and finished at his scheduled departure and end location.

2.3 Solving the Optimization Problem

2.3.1 Introducing Dynamic Programming

One of the most straight forward ways to solve such problems is to simply test all possible strategies, and find which one has the largest revenue. This is often described as a *brute force* approach. While we do have a finite number of solutions, testing all these strategies would be too time consuming. With the given objective, the worst possible scenario would require testing N^{KT} possibilities. Clearly, a more practical approach needs to be found.

Dynamic programming is a general approach for solving a complex problem by solving a series of subproblems until it finds the solution of the original problem [3]. The main reason for doing this is to reduce computation time. While its application may depend on the specific problem, two main characteristics of such a problem is that it has overlapping subproblems and an optimal substructure [4]. A problem has overlapping subproblems if the solution to one of the subproblems can be used to solve a larger subproblem. An optimal substructure exists if the globally optimal solution can be constructed from locally optimal solutions to subproblems. Now we apply this to our situation.

2.3.2 Applying Dynamic Programming to the Objective

Consider the very final moment in time that the K controllers can make a decision, ie at time T. With this single time unit remaining, we can find the optimal strategy for the controllers at any given location. This is fairly simple because we only consider one time unit, and the set of possible locations of controllers is manageable. Now consider the possibilities at time $T-1$. The travel time will be either 1 or 2 time units. In the first case, we can add the revenue made from $t = T-1$ to the optimal revenue (which was already calculated) at time $t = T$ and and the given location. In the second case, we immediately take the revenue made from the single taken connection.

In general, once we have calculated the optimal path for time $t, \dots, T$, it is relatively easy to calculate the optimal path for $t-1, \dots, T$. This is called *backward recursion.* At each timepoint, it is essential to list all possible states (where is/are the controllers(s) or are they *en route* in a train connection) and all possible moves (which connections are available). The exact formulation is given in the following section.

2.3.3 Mathematic Formulation and Pseudocode

We define a *state* x_t at time t as the connection the K controllers have last taken. This means that a controller is either waiting at the end location of the connection, or is still *en route*. The *state space* X_t is the set of all states x_t at time t.

We also define an *action* a as a set of connections the controllers could take. Let the *action space* $A(x_t)$ be the set of all possible actions for the controllers, given state x_t. Also let $x_{t+1}(a)$ be the new state at time $t+1$ when taking action a in state x_t.

Let $R(x_t, a)$ be the revenue generated from taking action a in state x_t. Finally, let the value function $V(x_t)$ be the total generated revenue from x_t to x_T.

The problem can now be written as a recursive equation:

$$\begin{aligned} V(x_t) \quad &= \quad \max_{a \in A(x_t)} \Big(R(x_t, a) + V\big(x_{t+1}(a)\big)\Big) \\ \text{subject to:} \quad & \quad t = 1, \ldots, T \\ & \quad V\left(x_{T+1}\right) = 0 \end{aligned}$$

We iterate backwards in time by solving $V(x_T), V(x_{T-1}), V(x_{T-2}) \ldots V(x_1)$ Below is a short piece of pseudocode for this problem:

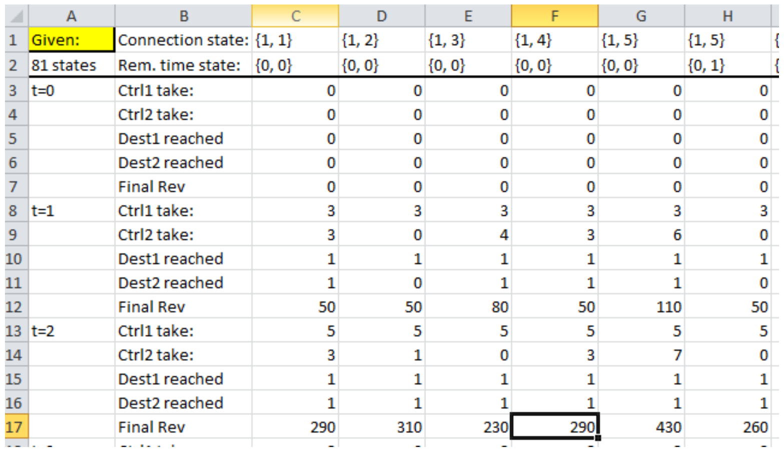
```
Data: $R, A$ and $S$
Result: Optimal strategy for all controllers
for t=T to 1 stepsize= -1 do
    for $x_t \in X_t$ do
        for $a \in A(x_t)$ do
            Determine revenue from $a$;
            Determine the state $x_{t+1}$ action $a$ will lead to;
            Look up the optimal revenue for state $x_{t+1}$;
        end
        Store the $a$ with the largest revenue for state $x_t$
    end
end
Print controller strategy, based on the stored actions $a$;
```

Algorithm 1: Dynamic Programming for Controller Strategy

In the following sections we discuss two approaches to dynamically programming this problem. The first, which we call the global algorithm, provides a guaranteed optimal solution, but is consequently much more time consuming. The second is a somewhat simplified version that performs quite fast but is not guaranteed optimal, which we call the heuristic algorithm.

2.4 Global Algorithm

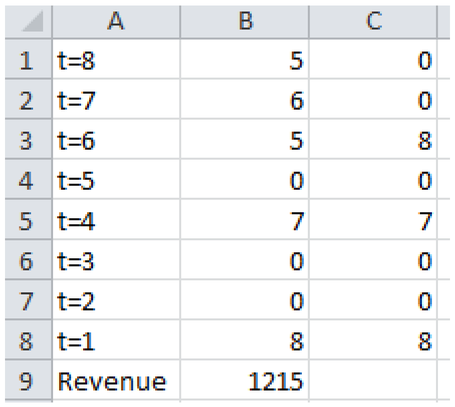
The global algorithm finds an optimal path for a set of controllers by simultaneously taking all controllers into account. First, a lookup table is generated, in which for every *remaining* time and every given state, the optimal connection is given. This may also be 0 (ie it is optimal to wait in this particular instance). The advised connection can be found in the row 'Take:'. With this advised connection, a sum of the revenues up to that point in time is given. This includes the revenue gained from the connection itself, and the revenue gained for following the rest of the optimal path. The total revenue can be found in the row 'Final Rev'. The row 'Final Dest' is explained below. The following picture is a screenshot of the first few lines of such a table.



	A	B	C	D	E	F	G	H	
1	Given:	Connection state:	{1, 1}	{1, 2}	{1, 3}	{1, 4}	{1, 5}	{1, 5}	{
2	81 states	Rem. time state:	{0, 0}	{0, 0}	{0, 0}	{0, 0}	{0, 0}	{0, 1}	{
3	t=0	Ctrl1 take:	0	0	0	0	0	0	
4		Ctrl2 take:	0	0	0	0	0	0	
5		Dest1 reached	0	0	0	0	0	0	
6		Dest2 reached	0	0	0	0	0	0	
7		Final Rev	0	0	0	0	0	0	
8	t=1	Ctrl1 take:	3	3	3	3	3	3	
9		Ctrl2 take:	3	0	4	3	6	0	
10		Dest1 reached	1	1	1	1	1	1	
11		Dest2 reached	1	0	1	1	1	0	
12		Final Rev	50	50	80	50	110	50	
13	t=2	Ctrl1 take:	5	5	5	5	5	5	
14		Ctrl2 take:	3	1	0	3	7	0	
15		Dest1 reached	1	1	1	1	1	1	
16		Dest2 reached	1	1	1	1	1	1	
17		Final Rev	290	310	230	290	430	260	

Figure 2: Output table of revenue and controller actions for every state

A Boolean value is used to check if the controller reaches their end locations (if specified). The connection and remaining time states are explained below. Once this table is constructed, the algorithm 'reads' it to find the optimal strategy for all controllers and prints out the total revenue as show in Figure 3.



	A	B	C
1	t=8	5	0
2	t=7	6	0
3	t=6	5	8
4	t=5	0	0
5	t=4	7	7
6	t=3	0	0
7	t=2	0	0
8	t=1	8	8
9	Revenue	1215	

Figure 3: Output table of revenue and optimal controller strategy

2.4.1 State Space - Connection and Remaining Time

In order to describe the state x_t, it is not sufficient to only name which connection the controllers are coming from. If only a part of the controllers can take a connection, one needs to know who those are. We also need to know the remaining travel time of all controllers that are *en route*. If the remaining travel time is 0, then that means the respective controller can proceed to a new connection. For example, say two controllers are coming from c_1 and c_5. The remaining travel time of c_1 is 1 so the remaining time will always be 0. On the other hand, c_5 has travel time 2. It could then have a remaining travel time 1 or 0. In total there are 2 states for the connection state $\{1, 5\}$, as can also be seen in Figure 3. above. Note that, because the controllers have their own specific schedule, the state $\{5, 1\}$ is not the same, and must therefor also be separately included. Given N connections, K controllers and the maximum traveltime T_{max} of all connections, the number of states is bounded by:

$$(N)^K \leq \text{number of states } \leq (N \cdot T_{max})^K$$

The left bound is in the case that the travel time of all connections is 1, whereas the right bound is the case where all connections have traveltime T_{max}.

2.4.2 Action Space

The Action Space is the set of all possible connections the controllers can take. For instance, three controllers could take $\{c_1, c_3, c_4\}$, $\{c_7, 0, c_5\}$ or

$\{c_7, c_5, 0\}$. Again, the controllers are not interchangable so the order of connections is important. Systematically creating this space is quite simple because all controllers can take any of N connections, or wait. The number of sets of connections is $(N+1)^K$. In order to find if an action a is possible for the set of controllers, several aspects must be checked. All controllers must be able to take the given connection, otherwise the set of connections is impossible, and we check the next action in the action space. As soon as one controller cannot take the given connection, the action a is deemed impossible.

2.4.3 Referring to the Next State

In terms of programming, determining the state x_{t+1} is tricky because some controllers will take connections with varying traveltimes. Others may be *en route*, so they cannot take a new connection. To solve this, we take given the connection state and remaining time state at time t, and apply the set of moves to the connection and time state at time $t+1$. An example is shown below:

State (time t)			Connection:		State (time $t+1$)	
Connection state	$\{1,5,3\}$	$+$	$\{5,0,4\}$	$\rightarrow$	Connection state	$\{5,5,4\}$
Rem. time state	$\{0,1,0\}$				Rem. time state	$\{1,0,0\}$

One particular problem is the fact that we do not know *a priori* where in the list of states, the state at time $t+1$ can be found. The ordering of the states is done systematically, but it is not trivial where to explicitly find a certain state. To solve this, all the states are indexed. Second, the algorithm simply searches through all states and stops once it finds the correct state. It then uses the index to determine which revenue value from the lookup table to take.

2.5 Heuristic Algorithm

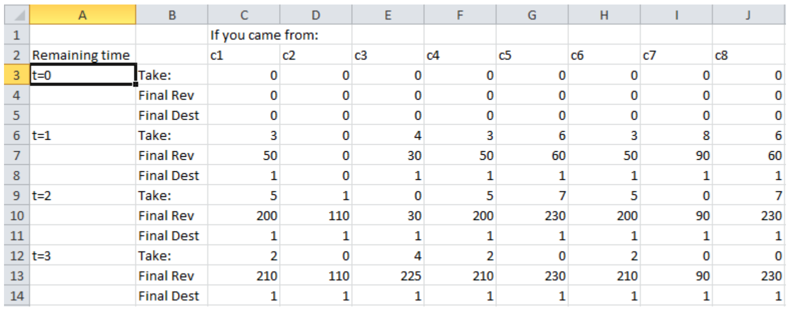
The global algorithm is able to give an optimal solution to the problem, but is very time consuming. As a consequence, the heuristic algorithm was developped, which produces a solution in the fraction of the time. However, this solution is not guaranteed optimal.

The heuristic algorithm finds the optimal course for a single given controller, and iterates this process for any additional controllers. To clarify some of the concepts, we use the same example that was shown in Figure 1.

We let R be:

$$R = \begin{bmatrix} 100 & 0 & 80 & 0 & 100 & 0 & 60 & 0 \\ 0 & 60 & 0 & 60 & 0 & 100 & 20 & 0 \\ 40 & 0 & 0 & 40 & 0 & 0 & 60 & 50 \\ 20 & 0 & 20 & 0 & 0 & 25 & 0 & 30 \\ 200 & 0 & 160 & 0 & 120 & 0 & 200 & 25 \\ 0 & 100 & 0 & 80 & 0 & 25 & 0 & 60 \\ 100 & 0 & 180 & 0 & 160 & 0 & 140 & 0 \\ 120 & 0 & 100 & 0 & 80 & 0 & 0 & 90 \end{bmatrix}$$

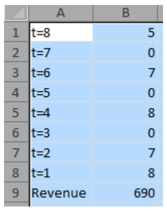
Essentially, the heuristic algorithm works the same way as the global algorithm. A lookup table is generated, and the optimal connection is found for every state. The revenue up to that point and the Boolean value to check if the final destination is reached are also included. The following picture is a screenshot of the first few lines of such a table.



	A	B	C	D	E	F	G	H	I	J
1			If you came from:							
2	Remaining time		c1	c2	c3	c4	c5	c6	c7	c8
3	t=0	Take:	0	0	0	0	0	0	0	0
4		Final Rev	0	0	0	0	0	0	0	0
5		Final Dest	0	0	0	0	0	0	0	0
6	t=1	Take:	3	0	4	3	6	3	8	6
7		Final Rev	50	0	30	50	60	50	90	60
8		Final Dest	1	0	1	1	1	1	1	1
9	t=2	Take:	5	1	0	5	7	5	0	7
10		Final Rev	200	110	30	200	230	200	90	230
11		Final Dest	1	1	1	1	1	1	1	1
12	t=3	Take:	2	0	4	2	0	2	0	0
13		Final Rev	210	110	225	210	230	210	90	230
14		Final Dest	1	1	1	1	1	1	1	1

Figure 4: Output table for the heuristic algorithm

Once the table is constructed, it is not straightforward to read what the optimal strategy is. The algorithm looks through the table to find the optimal course, and prints out which connections should be taken, given the amount of remaining time. Also, the revenue of this strategy is printed below. See below for a screenshot. For any additional controllers, the revenue the preceding controller obtained is removed from R. The algorithm then calculates the optimal course for the next controller, based on the adjusted R. This means that a new table is constructed, and then the strategy is printed next to any already printed controller strategies. Some points of interest from a programming perspective are given in the following subsections.



	A	B
1	t=8	5
2	t=7	0
3	t=6	7
4	t=5	0
5	t=4	8
6	t=3	0
7	t=2	7
8	t=1	8
9	Revenue	690

Figure 5: Table of the optimal controller strategy for heuristic algorithm

2.5.1 State Space

The algorithm always considers one controller at a time. To describe the state (consisting of one controller), we use the connection that leads to the controller's location. So if the controller is in the bottom station, we say he came from c_3. This means the state space can be described by the set $\{c_1, c_2 \ldots c_N\}$. However, some connections describe the same location. In this case, the list of truly different connections is $\{c_1, c_2, c_3, c_5, c_7\}$. However, identifying which connections are redundant would also cost time. Also, this complicates referring to the correct state x_{t+1}. Note that we do not need to consider any possible remaining travelling time in the state space. For instance, the travel time of c_5 is 2. If a controller takes c_5 at remaining time t, we only need to look at the optimal revenue at time $t+2$, given that the controller came from c_5. This is only possible in this particular instance because we only deal with one controller at a time - in the global algorithm, this simplification is no longer possible.

2.5.2 Programming Issues

In this subsection we highlight some of the challenges that were faced when programming the heuristic algorithm. Given a controller coming from a connection c_i (the state), there are several steps to see if taking a certain connection c_j is possible.

- The connections must be physically attached to each other. This can be checked using the adjacency matrix A: if A_{ij} is 0, then the connection is not possible.

- A train must actually ride c_j at time t. This is checked using the revenue matrix R. If R_{tj} is 0, then the move is not possible. However, when multiple controllers are used, they must have the possibility to go on a connection even if a previous controller has already 'claimed'.

In practice, this means that an adjusted R should be used for every next controller for which the optimal schedule is calculated.

- A controller's connection must be within his/her time schedule. Suppose a controller wishes to finish at time t_{end}. The travel time of the last connection a controller takes at time t must then be less than $t_{end} - t$. The travel time of a connection can be found in the corresponding column of A. Of course, it is not possible to go on a connection at a time before the begin time of the controller.
- A controller must start and finish according to his/her schedule. To accomodate the start location, we first run the algorithm, without any restrictions regarding the start location. We then simply take choose the optimal path for which the starting location corresponds with the correct location state (and start time).
- An end location is a little trickier. Because we are performing recursion backwards in time, we cannot simply run the algorithm and then select the correct state. For instance, the algorithm always choses the connection with the largest revenue, and may overlook the connection with a lower revenue that does reach the end location. To solve this, we must introduce an addition Boolean variable (Final Dest) that checks if the end destination has been reached. If it is false (ie value 0), the choice of connections is restricted to connections that lead to the end location. The Boolean is then changed to true if such a connection exists. If the Boolean is already true, we do not have any additional restrictions.
- If a controller is *en route*, or delibarately waits at a station, then this should always be a possible action.

3 Comparing the Heuristic and Global Algorithm

As one may expect, the heuristic algorithm does not always perform as well as the global algorithm. That is to say that the heuristic sometimes produces a controller strategy whose total revenue is less than that of the stragey from the global algorithm. However, there are a few reasons to choose for the heuristic:

- Generally, the computation time of the heuristic is much smaller than that of the global algorithm. This has become apparent from the examples below and the experience of running many other setups. If the user wants an optimal strategy for multiple controllers, the algorithm performs much slower. If a user only has one controller, the global algorithm's computation time is marginally larger than the heuristic's. The computation time is a serious issue to consider. As we see in the following examples, a large setup with several controllers can result in hours of computation time, whereas the heuristic only takes seconds to reach a solution.

- The difference between the revenue from the heuristic and from the global algorithm seems to be bounded. In other words, the heuristic does not perform arbitrarily bad compared to the global algorithm.
- In the heuristic, the optimal route of each controllers is consectutively calculated. The order that the controllers are handled has an effect on the total revenue. This is because the controllers are 'greedy': each controller takes the revenue maximising route for him/herself, although this may not be the revenue maximising route for the team of controllers. However, it seems that there is always an order of controllers that will equal the revenue of the global algorithm. This is explained in detail in section 3.3. Another formulation is that there does not seem to be an instance where the global algorithm finds a higher revenue than of all possible order of controllers in the heuristic.

Each of these aspects are handled in more detail below.

3.1 Examples where Heuristic Performs Worse

In the following examples, we demonstrate several instances where the heuristic performs worse than the global algorithm. For the sake of readability, the stations in each example have been labelled.

3.1.1 Example 1 - Different Time Schedule

In this case we have a simple triangle layout of stations: A, B and C. Each connection has travel time 1.

Let R be:

$$R = \begin{bmatrix} 20 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 20 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 20 \\ 5 & 0 & 0 \end{bmatrix}$$

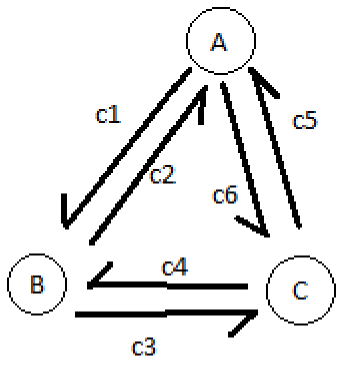


Figure 6: Station layout for Example 1

From station A there is a train that collects 20 revenue on each of the three connections (going counterclockwise). There is also a single train from A to C worth 5 revenue. It is impossible to collect all the revenue with a single controller. Consider two controllers in the following order:

	Controller 1	Controller 2
Start time	1	1
Finish time	1	3
Start location	A	A
End location	No pref.	No pref.

Both controllers are the same, except for their finishing time. The first controller can only take one connection at the start, and is then immediately finished. The heuristic will let Controller 1 collect 20 revenue from c_1. It is the most profitable connection available. The second controller will also take c_1 and complete the triangle by taking c_3 and c_5. This leads to a revenue of 40 for Controller 2. In total, the heuristic would find a strategy for the controllers with a total revenue of 60.

It is not hard to tell that the system optimum would be assigning Controller 1 to take c_6 and Controller 2 to take a trip around the entire triangle, for a total revenue of 65. This is an example of how different time schedules lead to a less than optimal controller strategy. The following example shows how location can lead to a suboptimal solution.

3.1.2 Example 2 - Different Location Schedule

Consider the following layout with stations A to D. All travel times are 1.

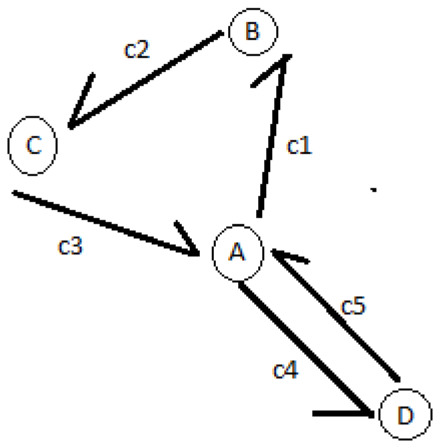


Figure 7: Station layout for Example 2

The Revenue Matrix is given by:

$$R = \begin{bmatrix} 33 & 0 & 0 \\ 0 & 33 & 0 \\ 0 & 0 & 33 \\ 50 & 0 & 0 \\ 0 & 50 & 0 \end{bmatrix}$$

If starting from station A, a controller could take the triangular route (from A to B to C) or go back and forth with (from A to D to A). The first option has a smaller revenue (99) than the second option (100). Now let the controller schedule be the following:

	Controller 1	Controller 2
Start time	1	1
Finish time	3	3
Start location	A	D
End location	No pref.	No pref.

The heuristic will let Controller 1 go from A to D to A and Controller 2 not be able to generate any revenue. This is because the revenue from c_5 has already been assigned to Controller 1. In total the heuristic gives a total revenue of 100.

The global algorithm would let Controller 1 go from A to B to C and Controller 2 go from D to A. This would give a total revenue of 149. Problems occur when the route for the personal maximum revenue of a controller, conflict with a different controller's optimal route. It is possible to enlarge the difference between the heuristic's revenue and the global algorithm's revenue, as shown below.

3.1.3 Example 3 - Largest Possible Difference in Performance

Consider the same layout and controller schedule as above, but with a different Revenue Matrix:

$$R = \begin{bmatrix} 33 & 0 & 0 \\ 0 & 33 & 0 \\ 0 & 0 & 33 \\ 1 & 0 & 0 \\ 0 & 99 & 0 \end{bmatrix}$$

The heuristic will still find a revenue of 100, because the first controller will 'claim' any possible revenue Controller 2 could have gotten. However, the global algorithm now finds a total revenue of 198. This is almost a difference of factor 2! In fact, rewriting R as:

$$\begin{bmatrix} 1-\delta & 0 & 0 \\ 0 & 49 & 0 \\ 0 & 0 & 50 \\ \delta & 0 & 0 \\ 0 & 100-\delta & 0 \end{bmatrix}$$

where δ is an infinitisemally small positive value, the global algorithm finds a revenue of $200 - \delta$. This example exploits the weakness of the heuristic algorithm to the maximum. However, the strength of the heuristic lies in its computation speed compared to the global algorithm. We disucss this in the next section.

3.2 Computation Time

In this section we compare the computation times of the heuristic and global algorithm. We see that the heuristic produces a strategy in a fraction of the time the global algorithm takes. In particular, increasing the number of controllers exponentially increases the computation time for the global algorithm, whereas the heuristic seems to grow in a slower fashion. The computation time of both algorithms depends on:

- The size of A (the number of connections).

- The length of R (the number of timesteps).
- The number of controllers.

To measure the effect of varying the parameters, a simple layout of stations was calculated for various settings:

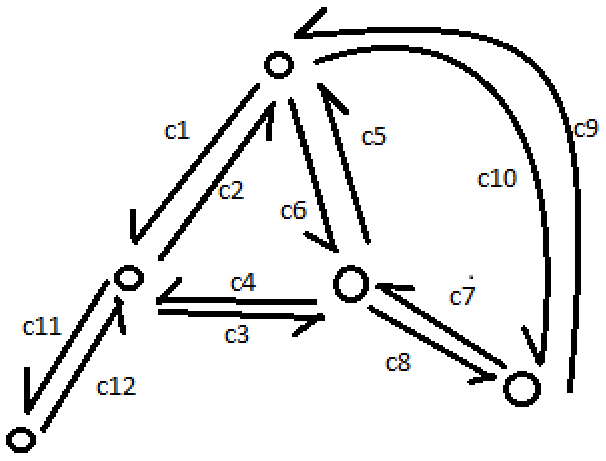


Figure 8: Station layout for measuring computation time

3.2.1 Size of A

First we only consider connections c_1 to c_6 and calculate the optimal strategy. This process is repeated with the addition of $\{c_7, c_8\}$, and then $\{c_9, c_{10}\}$ and then $\{c_{11}, c_{12}\}$. The computation time for both algorithms with varying number of controllers is recorded and graphed in Figure 5 and 6.

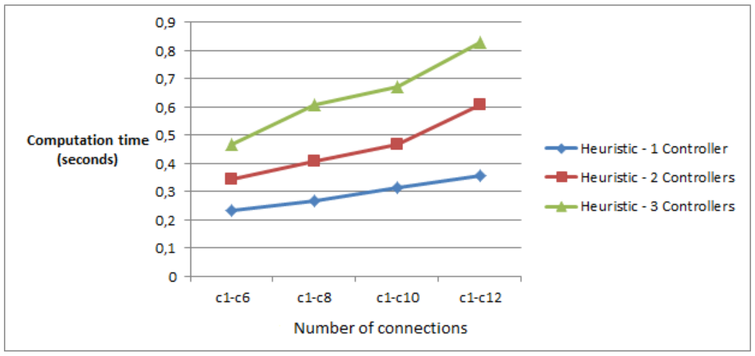


Figure 9: Computation time for number of connections (heuristic)

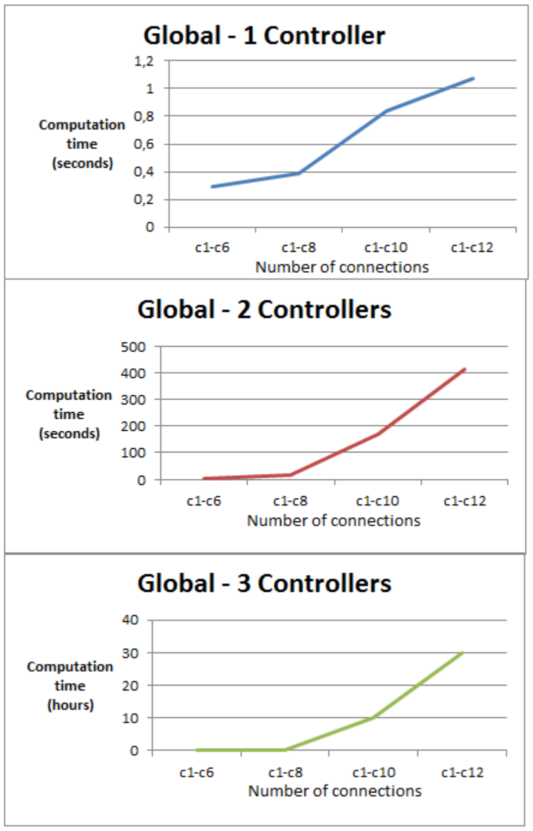


Figure 10: Computation time for number of connections (global algorithm)

An increase in the number of connections seems to have a exponential increase in the computation time. This applies to both the heuristic and global algorithm. This seems fair because each new connection creates a new state that needs to be tested *and* an extra possible connection that can be taken. The effect is particularly clear for multiple controllers in the global algorithm. Note that the computation times for the global algorithm with 3 controllers are based on estimates of the time it took to perform one timestep. This was done because it became clear that running the algorithm would be incredibly time consuming.

3.2.2 Size of R

We consider only connections c_1 to c_6. The number of timesteps are increased from 7 to 11.

$$R= \begin{array}{llllll|llllll}
20 & 0 & 25 & 0 & 0 & 20 & 0 & 0 & 0 & 20 & 0 & 25 \\
0 & 15 & 0 & 30 & 0 & 0 & 10 & 25 & 0 & 5 & 0 & 0 \\
0 & 25 & 0 & 35 & 10 & 0 & 15 & 45 & 20 & 0 & 0 & 25 \\
0 & 0 & 50 & 0 & 0 & 50 & 0 & 0 & 0 & 40 & 0 & 0 \\
55 & 0 & 0 & 0 & 45 & 0 & 0 & 0 & 30 & 5 & 55 & 0 \\
10 & 0 & 15 & 0 & 10 & 0 & 15 & 0 & 15 & 0 & 10 & 0
\end{array}$$

The left portion represents the original value of R. To test the effect of additional time steps on computation time, a new column was added one at a time. Similar to above, the computation times are recorded and graphed below (Figure 7). We see that increasing the number of timesteps has a linear increase in computation size. Because we are applying dynamic programming, it makes sense that each timestep would be similar in terms of computation size.

3.3 Conjectures on the Performance of the Heuristic

By choosing the heuristic, one can quickly have a strategy for all the controllers. However, it is not guaranteed optimal. By contrast, the global algorithm is particularly slow when the parameter sizes grow but does offer a guaranteed optimal solution. It is up to the user to decide which algorithm to use. Two additional hypotheses are listed below. It is our belief that they hold, but it has not been possible to prove or disprove them.

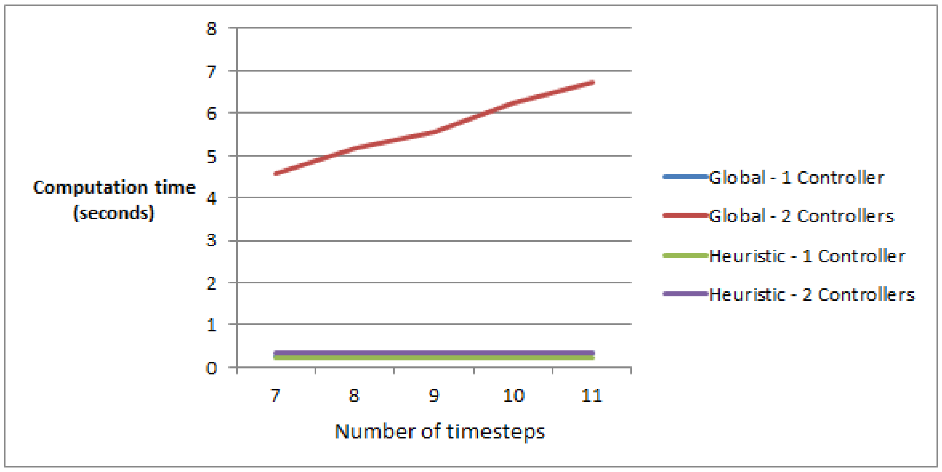


Figure 11: Computation time for number of timesteps

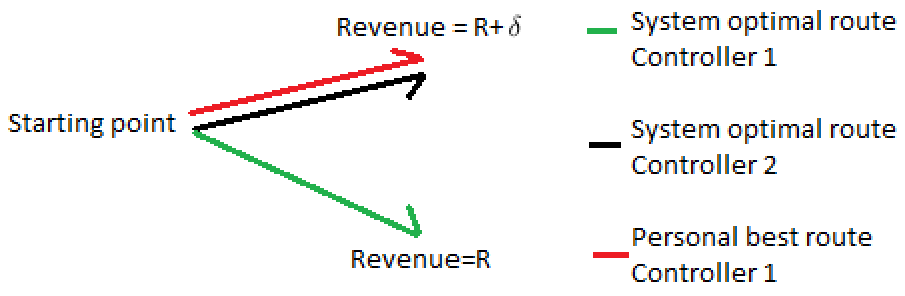
3.3.1 Bounded Optimal Value in Heuristic

Conjecture 1 *The heuristic algorithm will generate a revenue that is at least equal to the half of the optimal revenue.*

The third example offers some insight to why we believe this to hold. Although a rigorous proof cannot be given, we do offer a simplified argumentation. To start with, the hypothesis holds for 1 controller. In such a case, the heuristic mimics the approach of the global algorithm, and consequently generates the same total revenue.

First we distinguish two routes for each controller: the system optimal route, and the personal optimal route. The first is the route that a controller should take, to optimise revenue for all controllers. This may not be the most profitable route for the controller personally, but if each controller in the system takes their system optimal route, then the largest possible revenue is obtained. By definition, each controller must have a system optimal route, even if this is to stand still for the entire schedule. Each system optimal route must contain a portion that is not overlapped by any other system optimal route. If this was not the case, the route would simply be to stand still. The personal optimal route is the route that is the most profitable for a controller personally, with no consideration of what the consequences are for other controllers. This route will only differ from the system optimal route if its revenue is larger than the system optimal route's revenue.

The heuristic algorithm always looks for the personal optimal route of each controller. Now consider a situation with two controllers. The following diagram is a worst case scenario, where the first controller takes the system optimal route of the second controller:


Revenue = R+δ
Starting point
Revenue=R
System optimal route Controller 1
System optimal route Controller 2
Personal best route Controller 1

In the worst case scenario, the first controller takes the system optimal route of the second controller (black/red line). Note that this must mean that the the system optimal revenue of Controller 1 must be less than personal optimal revenue (hence the arbitrarily small term δ). If this was not the case, Controller 1 would not be directed upwards in the first place. In this worst case, Controller 2 generates 0 revenue because it cannot take any other route. For instance, the green route may not fit in the controller's time schedule. The system optimal revenue is $2R + \delta$, while the heuristic would find a revenue of $R + \delta$. Letting $\delta \to 0$ we see that the heuristic's revenue is bounded by half the global algorithm's revenue. h

In practice, we have seen that for 3 or more controllers, the heuristic's revenue has never been less than half of the optimal revenue.

If this hypothesis is true, this would be very useful for decision makers. This is because the heuristic does not perform arbitrarily bad compared to the the global algorithm. Given a revenue value from the heuristic, a user can decide whether running the global algorithm is worth finding (at most) double the heuristic's revenue.

3.3.2 Correct Order finds the Optimal Value in Heuristic

Conjecture 2 *Given all possible orders of K controllers, there exists an order which generates the optimal revenue value that the global algorithm also generates.*

The order the controllers are put into the heuristic, has a large impact on the revenue. Generally, when more restrictive controller schedules (shorter working time, hard to reach begin/end locations, etc) are put ahead of more flexible controllers, this will cause the heuristic to underperform. However, in the examples above, the heuristic can actually perform as well as the global algorithm, if the correct order of controllers is used.

To disprove the hypothesis, a situation must be created where all permutations of controllers do not generate as much revenue as the global algorithm. In this situation, a controllers must grant priority to a different controller, but also be granted priority in a later stage (or vice versa). If this is not the case (ie a controller always grants or takes priority to a constant set of other controllers) then we can construct the global optimal strategy using the heuristic. We believe that, in order for controllers to give *and* take priority, the controllers must either have different objectives or that A and R must be different for each controller. From an NS perspective, those two possibilities both seem unlikely. The controllers are employed by NS (so should not have differing objectives) and each controller has the same travel time and freedom of choice as the other.

Again, if this hypothesis holds, this would be very handy for decision makers. Considering the computation time of the global algorithm, it will likely be faster to run the heuristic for $K!$ permutations, and then keeping the best solution.

4 Conclusion

Summing up, both algorithms have their advantages for solving the optimal strategy for a set of controllers. The heuristic can offer a fast, not necessarily optimal solution, while the global algorithm offers a slow, guaranteed optimal solution.

4.1 Relevance to NS

We believe this algorithm could quite easily be implemented for the NS. Each controller can specifiy their preferences, and the input matrices R and A are flexible in use. Changes in the train schedule can quickly be applied to R and any new types of connections can easily be added to A. By varying the parameters, a user could experiment with certain policies. For instance, how much revenue is lost if all controllers worked an hour less? How much extra revenue does one extra controller generate? The algorthims could help with such types of analysis of the current system.

One important issue is the large size of the Dutch railway system. As we have seen, the computation time for the global algorithm grows exponentially with the number of connections and controllers. Calculating such a system with the global algorithm is possible, but extremely time consuming. Another assumption in the model is that the revenue on each train is deterministic. In practice, this is not the case. However, it could be considered as an expected revenue of some uknown distribution, allowing the algorithm to calculate the expected optimal revenue.

4.2 Further Research

As hinted above, modelling the revenue of each connection as a stochastic variable may be a better approximation of reality. Additionally, the model assumes that controlling a connection does not have any effect on the revenue of adjacent connections. Referring back to our first station layout, if a controller goes on connection c_1 at time t and gives all illegal passengers a fine, this will likely have an effect on the number of illegal passengers on c_5 at time $t+1$. Modelling this effect would make this a considerably more difficult problem because for each state at each time, we would need to adjust R.

A List of Symbols

c_i	A connection.
$t = 1 \dots T$	Time. May also be the *remaining time*, depending on the context.
R	Revenue Matrix.
A	Adjacency Matrix.
K	The number of controllers in the system.
x_t	State at time t as the connection the K controllers have last taken.
X_t	The set of all states x_t at time t.
a	A set of connections the controllers could take.
$A(x_t)$	The set of all possible actions for the controllers, given state x_t
$x_{t+1}(a)$	The new state at time $t+1$ when taking action a in state x_t.
$V(x_t)$	The total generated revenue from x_t to x_T.

References

[1] *Eerste week nieuwe dienstregeling NS*, published 19 December 2006, available at www.ns.nl. Retrieved 23 July 2013.

[2] *NS Jaarverslag 2012*, published 14 February 2013, available at www.ns.nl. Retrieved 22 July 2013.

[3] *Scheduling - Theory, Algorithms and Systems* M. Pinedo, New York 4th edition 2012

[4] *Introduction to Dynamic Programming*, by Jesse Farmer, published 15 November 2008, available at http://20bits.com/article/introduction-to-dynamic-programming Retrieved 22 July 2013.