EXTENDING THE SHUNTING PLAN GENERATOR OF THE DUTCH RAILWAYS WITH NON-SERVICE TRAIN TRAFFIC

Marie Klotz
University of Twente
PO Box 217, 7500 AE Enschede
the Netherlands
m.s.klotz@student.utwente.nl

ABSTRACT

Train traffic scheduling and shunting are complex logistical problems facing numerous different constraints. The Dutch railways, NS, created software to calculate the capacity of train hubs to see where future expansion of the railroad network is needed. This work describes the addition of non-service traffic trains to the mixed-integer-programming-based Instance Generator which creates traffic instances for train unit shunting problems. It investigates the influence of train density at rail hubs and the addition of non-service traffic on the feasibility and validity of instances. This paper also analyses the maximum train density at the rail hub of Heerlen, the Netherlands.

Keywords

Shunting Problems, Simulation, Optimization, Train traffic, Mixed Integer Programming, Service traffic, Railroad, Rail hubs,

1. INTRODUCTION

The travellers' branch of the Nederlandse Spoorwegen (Dutch Railways/NS) works on the implementation of Deep Reinforcement Learning for the solving of **train unit shunting problems (TUSP)**. The purpose of this is to calculate the capacity of hubs for shunting train units within the Dutch railroad network and make strategic decisions for the future expansion of the network [6]. The TUSP solver that is based on Deep Reinforcement Learning needs scenarios of train traffic as an input. Those scenarios are created by an Instance Generator which generates realistic 24h train traffic scenarios for a rail hub. The Instance Generator chooses the arrival and departure time of service trains based on **arrival** and **departure distributions** that it receives as an input. Those multinomial distributions describe the likelihood of the trains to depart or arrive within a certain time interval of the day. In this paper, the term **validity** describes whether the train traffic of an instance matches the input distributions that were given to the Instance Generator. It measures how well the generated traffic depicts the real train traffic.

The purpose of this research is to add **non-service traffic** —passenger trains and freight trains which pass through a hub without any service tasks being performed— to the Instance Generator by extending the current mixed integer programming (MIP) problem. This way, the Instance Generator will regard all traffic at a hub and allow for a more accurate calculation of the capacity. The performance of the Instance Generator is then measured by analysing the threshold of the three most relevant variables —the number of service traffic trains, the number of non-service traffic trains and the security distance between non-service traffic trains— for which an acceptable instance can be generated. Those three variables describe the **train density**. The more service and non-service traffic trains and the less security distance between trains, the higher the density at a rail hub.

Related work While there is no research specifically about the instance generation for train unit shunting problems, a lot of research about solution approaches has been performed. Van den Broek describes the mixed-integer programming algorithm that NS was using in 2016 and proposed Local Search as a new and better method to solve TUSPs [8]. The paper also gives a detailed explanation of constraints that need to be regarded at NS service sites and illustrates the complexity of TUSP.

Sajedinejad et al. describe SIMARAIL, a software that was designed for the Iranian railroad network, which uses discrete-event-driven simulation paired with optimization through the use of genetic algorithms to solve train scheduling problems [7]. The paper illustrates why scheduling and train unit shunting solutions are not universally applicable across railroad networks. The Iranian railroad network and its constraints for planning and scheduling problems differ largely from the Dutch network. Due to old tracks, daily track maintenance slots need to be planned, which negatively affects the capacity of hubs. In their scheduling problems, the Iranian software also needs to regard religious constraints that do not exist in the Netherlands. There are many other papers that describe solution approaches or sub-problems of train shunting and train scheduling problems, such as Freling et al. [3], Hassannayebi et al. [4] and D'Ariano et al. [1].

Feasibility describes whether a generated instance complies with the business rules of NS such as the defined minimum time between two trains in the same direction. No related research can be found regarding the feasibility of the generated instances as this criterion is specific to NS. As the aforementioned example of Iranian train traffic shows, railroad networks differ so much that there is no research published about this because it would not be generally applicable across countries or different rail operators.

This paper will investigate the following research question and its subquestions:

RQ1.: How does train density influence the solvability of rail hub traffic instances for train unit shunting problems?

RQ1.1: How to add non-service traffic to the mixed-integer-programming-based Instance Generator?

RQ1.2: What is the impact of adding non-service traffic on the validity of instances?

RQ1.3: What is the maximum train density that can be dealt with at the rail hub in Heerlen, the Netherlands?

RQ1.4: What is the impact of adding non-service traffic on the feasibility of generated instances?

2. BACKGROUND

2.1 Train planning problems at the Dutch Railways

The NS operates one of the busiest railroad networks in the world [8]. This network contains hubs consisting of train stations and service sites. Figure 1 shows one example of a **hub** within the Dutch railroad network. This train hub in Heerlen contains two platforms for passengers (marked in Fig. 1 with “Perron”), one platform to perform cleaning tasks on the trains (marked in Fig. 1 with “Reinigingsperron”), multiple tracks for parking trains, and a few single ended tracks.

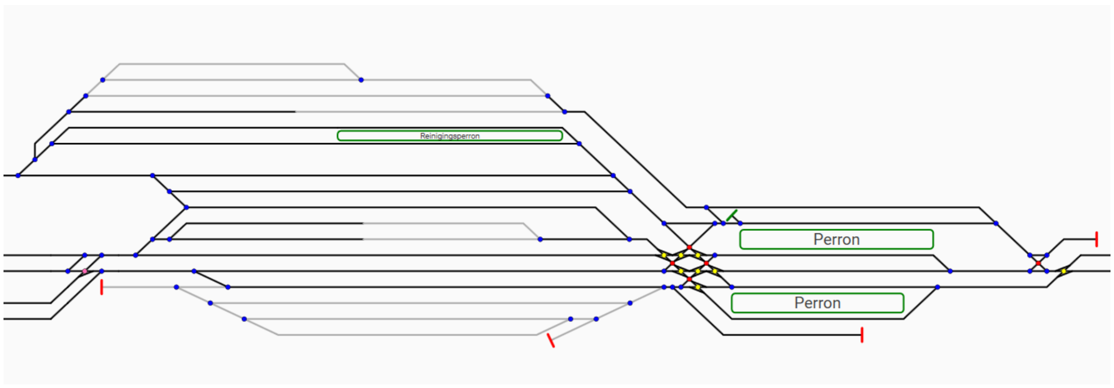

Figure 1. Schematic map of the rail hub in Heerlen, the Netherlands

The solid black lines represent electrified train tracks while the lighter grey ones are tracks which are not electrified. The blue dots indicate standard switches and the yellow dots indicate a double slip switch. Those switches allow trains to transfer from one track to another. The red dots describe crossovers. They enable trains to switch from one track to another parallel track.
For those hubs, long-term train schedules and shunting plans are being created. Shunting plans are needed to coordinate train traffic and necessary service tasks, such as cleaning and maintenance. They describe the incoming trains, the outgoing trains, the performed service tasks and the movements that every train coach makes within the hub. For those plans, a lot of constraints need to be considered, such as the traffic schedules, the available facilities to perform service tasks, the infrastructure of the rail hubs and the availability of personnel.

The number of passengers and the utilization rate of the Dutch railroad network have steadily increased over the last years [5]. Therefore, the NS needs to decide whether and where to expand their existing facilities. The Research and Development Hub Logistics of NS Reizigers created software which calculates the capacity of NS’s service sites by generating and solving realistic train unit shunting problems. The **capacity of a service site** is hereby defined as the maximum number of train coaches for which a realistic shunting plan can be constructed.

The TUSP-solving software is the software used to solve TUSP problems later in the tool chain. It makes use of three different methodologies. Currently, Local Search and Constraint Programming are already implemented, while the possibilities of Deep Reinforcement Learning are also being explored [6] [8]. The current solvers provide very different solutions to only slightly differing TUSPs. The solver based on Deep Reinforcement Learning promises to provide more general solutions. Rather than adapting to every instance, it explores the best general solution and will adapt that solution based on changes in the TUSP. Hence, it will provide similar solutions to similar instances which will be more comprehensible for human planners and allows for easier rescheduling [6].

2.2 Instance Generation of traffic scenarios

One part of the described software is the Instance Generator. As input, it receives the statistical distributions for the arrival and departure time of trains, the combination of train coaches those trains consist of, the service tasks that need to be performed, and static values, such as the infrastructure of the service hub (see Fig. 1) and information about the employee resources at the hub such as the availability of maintenance employee teams.

The Instance Generator then outputs realistic traffic scenarios for the shunting problems which shall be solved by the TUSP-solvers.
A limitation of the current implementation of the Instance Generator is that it takes only service traffic into account. **Service traffic** comprises trains which need some sort of service at the service site like cleaning or reparation or trains that need to park in the shunting yard during off-peak hours.
To generate realistic instances, NS needs to extend this generation and include non-service traffic. On their way through the hub, non-service trains block a certain path which then cannot be used by service traffic trains that want to enter or leave the shunting yard of the rail hub. Therefore, service traffic trains need to be scheduled accordingly, so that they do not interfere with the non-service train traffic.

2.3 Constraints to the addition of Non-service traffic

There are two types of constraints that need to be regarded when adding non-service traffic to the Instance Generator:

1. Route conflict constraints: Trains that enter a hub occupy a certain route within that hub by blocking it for other trains. There are specific crossing times that need to be regarded to avoid conflicts between trains that take the same path within a hub.
2. Continuity constraints: A scenario is generated for a 24h period but not all trains that enter the hub leave it at the end of the day. Therefore, some trains need to be already present

at the beginning of a scenario. Continuity between those trains needs to be ensured.

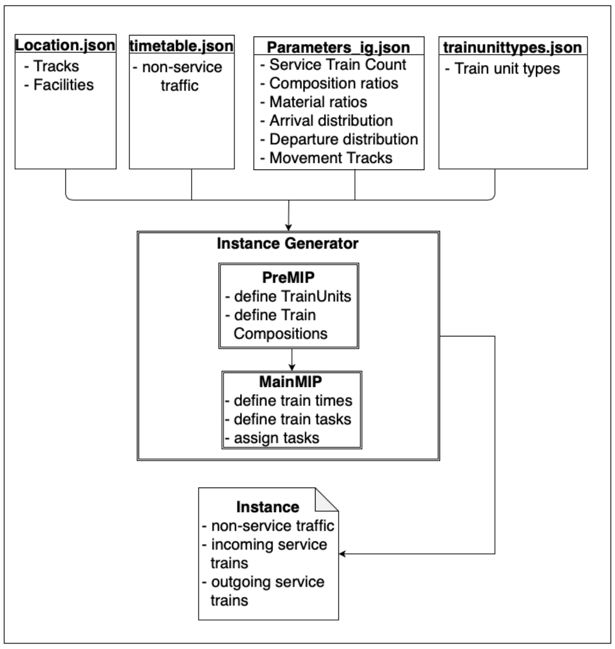

Figure 2. The Instance Generator and its in- and outputs

2.4Architecture of the Instance Generator

The current implementation of the Instance Generator is based on two mixed-integer programming (MIP) problems as illustrated in Fig. 2.

The first one, the so-called PreMIP, determines the composition of each arriving and departing service-traffic train. The PreMIP receives the possibilities for train compositions. Figure 3 displays the arrival and departure ratio for a train containing exactly one so-called “SLT4” unit. This is the data format which the PreMIP receives as input. A train can consist of several units of different types. Those ratios are given in integers. In order to determine the probability for a certain train composition to be chosen for the instance, its arrival ratio needs to be divided by the sum of the arrival ratios of all possible compositions.

```
"trainCompositions": [{
  "unitCounts": {
    "SLT4": 1
  },
  "arrivalRatio": 16062.0,
  "departureRatio": 22784.0
},
```

Figure 3. Train composition ratios.

When the arrival ratio of a train composition does not match its departure ratio, this will lead to unit shunting between arrival and departure trains. For example, if the arrival ratio of a train composition is higher than its departure ratio, this means that more arriving service trains will have this composition than departing ones.

For each of the units that can compose a train, a ratio is given that indicates the share of this unit type among all train unit types. (Fig. 4, “arrivalRatio”). Besides that, the tasks that need to be performed on the unit are being given. On a unit of type “SLT4”, an internal cleaning needs to be performed every 86400 seconds, so every day. This cleaning takes 540 seconds, as can be seen in Fig. 4.

```
"units":{ "SLT4": {
  "arrivalRatio": 28428,
  "taskTypes": [{
    "name":
"inwendige_reiniging",
    "period": 86400,
    "duration": 540,
    "exclusionGroup": 0,
    "exclusionPriority": 0
  },
}
```

Figure 4. Input information about a SLT4 train unit

The composition of a train therefore determines its length and therefore how much space it will occupy within the hub. It also indicates which tasks need to be performed on the train. The more tasks need to be performed and the longer the trains, the more difficult it will be to generate a valid instance.

```
{"nonServiceTraffic": [{
    "members": [
      "88","177","89","178",
      "115","135","71","190",
      "77","189","48","156",
      "29","28","147","15",
      "140","2","1"
    ],
    "arrival": "67940",
    "departure": "68120",
    "id": "Doorgaand-vr-3917"
  },
```

Figure 6. A non-service train within an instance

When the PreMIP has been solved successfully, the second MIP (i.e. the MainMIP, see Fig. 2) receives the train compositions. It then optimizes the movement times of all trains and selects the parking and side-tracks of all service trains. The MainMIP also determines which trains are present at the start of the day or remain present at the hub at the end of the day. After solving both, the Pre- and the MainMIP, the Instance Generator assigns tasks such as a cleaning or maintenance to the trains and then returns the instance (Fig. 5).

If the Instance Generator receives non-service traffic, it disregards it when solving the MIP problems. It will simply return the non-service traffic that it received together with the generated service-traffic. This is a problem, because service traffic trains could be scheduled on a track during a time where it is already occupied by a non-service traffic train. Therefore, the software needs to be adapted so that no instances with overlapping trains will be generated anymore. To do so, the constraints listed in subsection 2.3 need to be added to the Instance Generator so that it regards non-service traffic trains.

Figure 6 shows a non-service traffic train in the format which is used to pass them to the Instance Generator:

- “Members” consists of a list of trackpart-ids which describe the path of a non-service train through the hub.

- "Arrival" and "departure" describe the time in seconds (since 0.00h) when a train enters or leaves the hub.

- The "id" is the identifier of the train, usually consisting of the word *doorgaand*, Dutch for ongoing, two letters describing the weekday and an integer for example "vr-3988" for a train on a Friday (Dutch: vrijdag)

It is important to note that non-service traffic follows a predetermined path through the hub, while the path of the service traffic gets determined by the Instance Generator and the TUSP solver. The Instance Generator determines the parking- track, where the train parks at the hub and the side-track, the track that leads to the parking track, of the service-traffic. The rest of the path is determined when solving the shunting instance.

```
nonServiceTraffic {
  members: [88,177,89,178,115,135,71,
  190, 77,189,48,156,29,28,147,15,140,
   2, 1]
  arrival: 67400
  departure: 67520
  id: "Doorgaand-za-3929"
}
in {
  time: 58534
  id: "4000"
  sideTrackPart: 42
  parkingTrackPart: 15
  members {
    id: "2401"
    typeDisplayName: "SLT4"
    tasks {
      type {
        other: "inwendige_reiniging"
      }
      duration: 540
    }
    tasks {
      type {
        other:
"technische_controle_B"
      }
      duration: 840
    }
  }
}

out {
  time: 76107
  id: "504000"
  sideTrackPart: 42
  parkingTrackPart: 15
  members {
    id: "****"
    typeDisplayName: "SLT4"
  }
}
```

Figure 5. A returned instance

3.ADDING NON-SERVICE TRAFFIC

3.1 Route conflict constraints (RQ 1.1)

The non-service traffic needs to be added to the Instance Generator so that all traffic at a hub is being considered. The first constraints to be regarded are route conflict constraints. While passing through a hub, non-service traffic occupies track parts within the hub. This leads to two subproblems. Firstly, the physical constraint that a train track part can only be occupied by one train at a time. Secondly, the safety regulation of having minimum times in between train movements. Hence, even after a train departs from a track, it still needs to stay free for a short time. For service traffic this period varies, depending on whether the two successive movements are going into the same or in opposite directions. Due to non-service traffic not having two separate movements, arriving and departing, like service trains, this minimum time is assumed to be the same regardless of the direction. Therefore, the interval during which a path is occupied can be defined as:

$$\text{OccupationTime} = [\text{arrivalTime} - \Delta_T; \text{departureTime} + \Delta_T]$$

where Δ_T describes the security distance to be kept between a non-service traffic and a service traffic train. For security reasons, the entire path of a non-service traffic train is blocked during the entire time that it is passing through a rail hub.

3.1.1 MIP problem extension

In order to add the route conflict constraint to the MainMIP, parameters as shown in Table 1 and optimization variables as shown in Table 2 have been defined. Additionally, it is necessary to understand that service traffic trains are denoted "m-trains" where "m" stands for movement and can be either an arrival movement or a departure movement. In contrast, non-service traffic trains are called "k-trains" in the following description of constraints.

Table 1. Parameters for non-Service Traffic

Parameter	Definition
$min_k \in \mathbb{R}_0^+$	The security distance in seconds between two movements where one train is a non-service and the other is a service train.
$arrival_k \in \mathbb{N}$	Arrival time of non-service traffic train *k* in seconds since 0.00h.
$departure_k \in \mathbb{N}$	Departure time of non-service traffic train *k* in seconds since 0.00h.
$x \in \mathbb{R}_0^+$	The middle of the interval $[arrival_k; departure_k]$ which is used to describe the movement time for non-service trains.
$\Delta_k \in \mathbb{R}_0^+$	$\frac{departure_k - arrival_k}{2} + min_k$ Describing the time in seconds that a service train has to keep from the middle of the interval that a non-service train spends at the rail hub.
$u_s^k \in \{0,1\}$	Indicator whether track *s* is used by non-service train *k*.

$M \in \mathbb{R}_0^+$	A large enough constant. In the implementation the sum of the length of a day in seconds and min_k.

Table 2. Optimization variables

Optimization variable	Definition
$x_{s,t}^k \in \{0,1\}$	Boolean variable indicating whether non-service train *t* uses track *s* to make its movement.
$S_{s,t,t'}^{mk} \in \{0,1\}$	Boolean Variable indicating if service-train *t* and non-service train *t'* both use track *s*.
$\sigma_{t,t'}^{mk} \in \{0,1\}$	Boolean variable indicating if service train *t* makes its movement after non-service train *t'*.

After the introduction of the parameters and optimization variables for non-service traffic, the route conflict constraints need to be formulated as linear constraints, similar to the linear constraint for obeying minimum follow-up times between service-traffic trains. The following constraints are implemented for all service traffic trains, all train tracks and all non-service traffic trains in the MainMIP.

Variable $S_{s,t,t'}^{mk}$ indicates if service-train *t* and non-service train *t'* both use track *s*. It should be equal to one if both trains *t* and *t'* use track *s*. So, the relation between $S_{s,t,t'}^{mk}$ and $x_{s,t'}^k$ and $x_{s,t}^m$ is the following:

$S_{s,t,t'}^{mk} = 1 \; if \; x_{s,t'}^k = x_{s,t}^m = 1$ and $S_{s,t,t'}^{mk} = 0$ otherwise.

First $S_{s,t,t'}^{mk}$ is enforced to equal zero whenever $x_{s,t'}^k = 0$ or $x_{s,t}^m = 0$:

(1) $S_{s,t,t'}^{mk} \leq x_{s,t}^m$

(2) $S_{s,t,t'}^{mk} \leq x_{s,t'}^k$

(3) $x_{s,t'}^k = u_s^k$

It is then possible to force $S_{s,t,t'}^{mk}$ to equal one whenever $x_{s,t'}^k = x_{s,t}^m = 1$ with the following constraint:

(4) $x_{s,t'}^k + x_{s,t}^m \leq 1 + S_{s,t,t'}^{mk}$

The choice of constraint 3 and the use of the parameter u_s^k and the optimization variable $x_{s,t'}^k$(that are always set equal to each other) is due to the implementation of this constraint in the Instance Generator. Here, a Java function, that adds the constraints which describe whether two service traffic trains use the same track, is being reused for defining the same relationship for non-service and service traffic trains. For both trains, this function needs an optimization variable as an input, despite the fact that the arrival track of a non-service traffic train should not be optimized but is fixed. This issue could be solved by adding constraint 3.

The newly introduced variable $S_{s,t,t'}^{mk}$ can now be used to enforce the minimum time between a service train movement and a non-service train movement on the same track.

The following constraints ensure that any service train movement performed has enough security distance from all movements of non-service trains.

(5) $x - T_t^m \geq \Sigma_s \Delta_k S_{s,t,t'}^{mk} - M\sigma_{t,t'}^{mk}$

(6) $T_t^m - x \geq \Sigma_s \Delta_k S_{s,t,t'}^{mk} - M(1 - \sigma_{t,t'}^{mk})$

Note that in constraints 5 and 6, x describes the middle of the Time Interval [arrival^k; departure^k] that non-service train *k* spends at the hub. It is chosen as movement time for non-service traffic train *k* because then the distance (in seconds) that needs to be kept from that movement is equal on both sides of x. Additionally $\sigma_{t,t'}^{mk}$ describing the relation between the service-train movement T_t^m and x, can be defined unambiguously. If one would look at $arrival_k$ and $departure_k$ as two separate movements like it is being done for service trains, $\sigma_{t,t'}^{mk}$ could not be defined unambiguously for cases where the instance generator checks whether a service train could move between the arrival and departure of a non-service train.

Constraints 5 and 6 make use of a so-called big-M term. The *M* in those constraints is defined as a sufficiently large constant (based on $\sigma_{t,t'}^{mk}$), to ensure that one of the two constraints will give an upper or lower bound to the service train movement T_t^m, while the other constraint is always fulfilled. In the implementation, M equals the sum of the day length (in seconds) and min_k. So, depending on which train comes before the other, either Constraint 5 or Constraint 6 ensure that the distance Δ_k is being kept between the two trains.

The use of a big-M term is sometimes being discouraged if it is difficult to choose a good value for the constant [2]. This is no problem here, as the maximum and minimum values for all movement times are clearly bound by the number of seconds in a day and therefore a good value can be chosen for *M*.

3.1.2 The objective function

Due to the scope of this paper, it is not possible to describe the entire MainMIP with all of its constraints. In order to understand the test results and how the additional constraints impact the objective value, the objective function of the MainMIP is given as eq. (7).

Table 3. Objective function parameters

Parameter	Definition
c_{track}	Penalty coefficient for deviating from the desired number of movements on each track.
c_{time}	Penalty coefficient for deviating from the desired target times.

Table 4. Objective function optimization parameters

Optimization variable	Definition
$\varepsilon_{s,m}^{\text{track}} \in \mathbb{R}$	Captures the difference between the desired number of *m*-movements and the actual number of *m*-movements performed on track *s*.

$\boldsymbol{\varepsilon_{t,m}^{\text{time}}} \in \mathbb{R}$	Captures the difference between the target time of m-train t and the actual time at which this train makes its movement.

The objective function is a weighted sum over all 'deviation' variables given in Tables 3 and 4. It describes how much an instance differs from the arrival and departure distributions for service trains and the ratios for track usage that were given as input to the Instance Generator. The objective value can hence be seen as a measure for the validity, the closeness to reality, of an instance.

$$(7) \quad minimize \qquad Z = \Sigma_m(\Sigma_t\, c_{track}\varepsilon_{t,m}^{track} + \Sigma_s c_{time}\varepsilon_{s,m}^{time}\ )$$

Adding the non-service traffic is expected to increase the objective value because service traffic trains need to be planned around the non-service traffic trains, thus making them more likely to deviate from the desired track or target time and therefore increasing the solution to eq. (7).

3.2 Adding non-service traffic: Continuity constraints

An instance generated by the Instance Generator spans 24 hours of train traffic. A train might enter the hub at the end of the day without leaving it again. Therefore, some trains might be present in the morning without entering the hub. It is important to ensure continuity between those trains. Since an instance only spans 24 hours and instances are generated independently from each other, it is not possible to have a train appear only in the next generated instance —so to say the next morning. Therefore, it is assumed that every instance has the same train schedule and if a train does not leave at the end of the day, the same train must have remained the previous day as well. Hence, the train should be already present at the hub in the morning.

This constraint cannot be formulated as a constraint for the MainMIP because the aim is to depict reality and therefore the arrival times of non-service traffic trains should not be changed. Instead, a function was implemented to check the arrival and departure times and ensure continuity before passing the non-service traffic on to the MainMIP.

3.3 Testing

After the non-service traffic has been added, the performance of the Instance Generator will be tested and measured using the train hub of Heerlen as testing location. Comparing the Instance Generator after the implemented changes to its previous performance does not give any valuable information since the old implementation would not react on non-service traffic. Instead the different test cases will be assessed based on criteria describing the validity (RQ1.2), the train density (RQ1.3) and feasibility (RQ1.4) of an instance.

- Solvability: Does the MainMIP find a solution for the input within 120 seconds and if yes, is the solution optimal?
- Objective value of the MainMIP, see eq. (7)
- Feasibility of the instance: Does the instance pass the instance checker by complying with the NS business rules?

During all tests, unless explicitly stated differently, the following follow-up and crossing times are assumed for the train hub in Heerlen: The follow-up time between two service traffic trains in the same direction (both trains arriving or both trains departing) is assumed to be 180 seconds, the cross-over time between two service traffic trains in opposite directions (one arriving and one departing) is assumed to be 360 seconds. The time between a non-service traffic train and a service traffic train is assumed to be 360 seconds independent of the direction of the service traffic train. It is also assumed that all service traffic and non-service traffic trains use the same tracks to enter the rail hub.

4. RESULTS AND DISCUSSION

4.1 The impact of adding non-service traffic on the validity of instances (RQ 1.2)

The validity of an instance is determined by its closeness to reality. The less deviation between a generated instance and the target distributions that the Instance Generator receives, the more valid the instance. The best measure for this is the objective value of the MainMIP's objective function. An ideal instance has an objective value of 0 which indicates that it matches the input distributions. A high objective value indicates that service trains had to be scheduled at different times and different tracks than intended. There is no specific objective value which can be seen as a threshold for valid instances. Every generated instance that is feasible and complies with all constraints is acceptable but the lower the objective value the better it depicts reality. Therefore, a low objective value is desirable.

Figure 7 depicts the effect of adding non-service trains while keeping the service trains at a stable count of 10. For every datapoint 5 test runs were executed. This means that for every run, the arrival times and the departure times of the trains differ. Figure 7 shows that initially the objective value increases but then it remains constant (at an objective value of 51.5556) for all values between 9 and 24 non-service traffic trains except for 15 and 20 where slightly better solutions can be found.

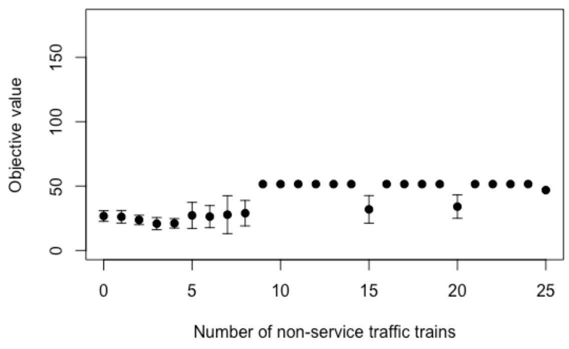

Figure 7. Increasing the number of non-service traffic trains

Figure 8 illustrates the complementary test, where the number of non-service traffic trains is kept stable at 10 while service-traffic trains are being added. It pictures a relatively constant increase of the objective value. The average objective values for more than 20 service-traffic trains are above 100 while 55.5556 was the maximum value in Fig. 7.

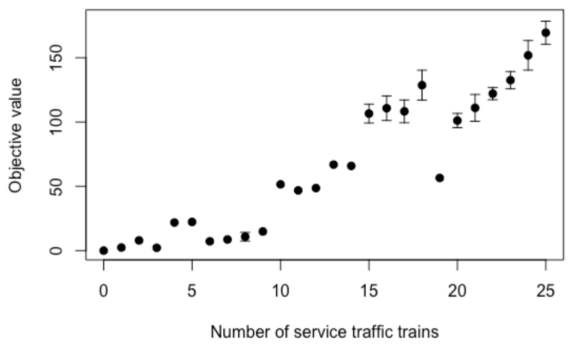

Figure 8. Increasing the number of service traffic trains

This illustrates that adding non-service trains to the instance has a much smaller impact on the validity of instances than adding more service traffic trains. This can be explained by the fact that the movement of service trains consists of arrival and departure. During both of those individual movements, security distance needs to be kept. A service train therefore blocks a train track for a longer period of time making it more difficult to schedule other trains. A non-service traffic train passes through a hub within a few minutes in one single movement. Hence, its path only has to be blocked for the Occupation time as defined in section 3.1.

4.2 The maximum train density at the rail hub Heerlen (RQ 1.3)

In order to determine the maximum train density that the train hub in Heerlen can cope with, empirical data of actual train traffic at the rail hub was used to create the input ratios and distributions for the instance generator. Based on this, 63 non-service traffic trains were assumed to pass through the hub out of which 50% use the same tracks as the service traffic trains to enter the hub. This represents the amount of non-service traffic in Heerlen on a weekday.

It was then tested how many service trains could be added to an instance before the train density would exceed the threshold where an acceptable instance for the rail hub can be found. In order to test how much the security distance, the time between non-service and service traffic trains, contributes to this maximum train density, three different security distances were tested. For every datapoint 10 test runs were executed.

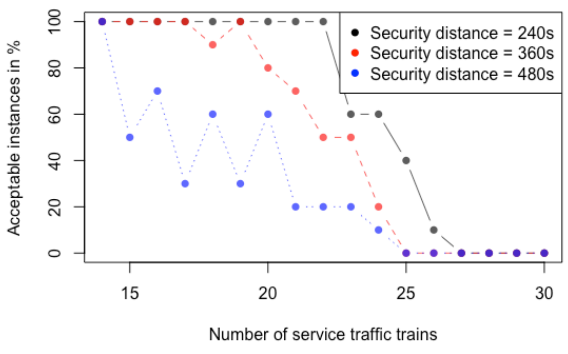

Figure 9. The maximum train density at the rail hub Heerlen

The maximum train density is reached when no valid instances can be found for a certain combination of security distance, number of non-service trains and number of service trains. Figure 9 depicts this threshold when the Instance Generator MIP solver is given 120 seconds to solve the MainMIP. It shows how the threshold is reached with 27 trains at a security distance of 240 seconds and 25 non-service trains at a security distance of 360 and 480 seconds.

It can be seen that the graphs, especially the one for a security distance of 480s, are not monotonous against the expectation that a higher number of trains should always be more difficult to solve. In reality, the rail hub does not fit 19 service traffic trains more easily within 24 hours than 18, despite the fact that the graph might lead to this assumption. Instead, the cause for the lack of monotonicity is most likely the computational time that was given to the Instance Generator in this test setting. When given more computational time, one can expect monotonically decreasing graphs and also a small shift of the threshold where acceptable instances can be found to the right. This effect is the biggest for a security distance of 480s because when more distance needs to be kept, it is more difficult to fit all service trains and the MIP becomes harder to solve.

Determining the security distance that needs to be kept between trains is a complex matter which depends on the type of train, its location within the hub and the possible conflicts with other trains that need to be avoided. Hence, choosing which value to use when generating shunting plan instances is not always straightforward. Therefore, it is interesting to consider the effect that different security distances have on the maximum train density. As seen in Fig. 8, the smallest security distance of 240 seconds allows for more trains than 360 and 480 seconds because if less distance needs to be kept between trains, more trains can fit into the rail hub. It stands out that doubling the security distance from 240 to 480 seconds, reduces the number of service trains that fit into a hub only by two trains and that no difference between the threshold for the two larger security distances could be found. Thus, these tests suggest that the security distance between non-service and service traffic trains has only a small impact on the maximum train density.

4.3 The feasibility of train instances with non-service traffic (RQ 1.4)

The feasibility of an instance is evaluated by the instance checker, a tool in the NS tool chain which tests whether instances generated by the Instance Generator comply with NS business rules e.g. for the use of facilities or human resources at a rail hub. If negative, the result of the instance checker warns which rule is being violated by the instance.

In order to test the impact of the addition of non-service traffic to the Instance Generator on the feasibility of instances, 400 test runs with (sample 1) and 400 test runs without (sample 2) added non-service trains were performed. For every test run, a valid instance was generated and checked by the instance checker. The number of service traffic trains was kept at 20. For every test run in sample 1, a number of non-service trains between 10 and 20 was drawn randomly from a uniform distribution.

As can be seen by the frequencies cross-tabulated in Table 5, the Chi-square test gives a p-value of 0.816391 meaning that there is no significant relationship between added non-service train traffic and feasibility at alpha($p<.05$). More concretely, the result of the chi-square test statistic at a sample size of 800 and with 1 degree of freedom is equal to 0.816391. *X^2 (1, N = 800) = 0.5051. p = .816391.*

Table 5. Feasibility testing results

	With non-service traffic	Without non-service traffic	Total
Instance Feasible	391	390	781
Instance not feasible	9	10	19
Total	400	400	800

This shows that the extension of the Instance Generator with non-service traffic trains did not have a negative impact on the feasibility of generated instances. Through the addition of non-service train traffic, more realistic instances can be generated without compromising on the feasibility of the generated instances.

5. CONCLUSIONS AND FUTURE WORK

This paper aimed to show the implementation of adding non-service traffic trains to the Mixed-Integer-Programming-based

Instance Generator and its effects on the validity and feasibility of instances. It also investigated the maximum train density at the rail hub of Heerlen, the Netherlands.

It was shown that the influence of non-service traffic trains on the validity of instances is low as compared to the impact of service trains on the same measure. Therefore, when using the implemented changes in order to generate instances for the TUSP solver methods of NS, one should not expect a big deviation in calculated hub capacity as compared to the previous implementation of the software. Moreover, no significant impact of the added non-service traffic on the feasibility of instances could be found.

Based on an empirical-data-based scenario it was found that under the assumption that 50% of non-service trains use the same rail infrastructure as service-trains, the maximum train density of the rail hub in Heerlen will be 25 to 27 service trains on a weekday.

The described implementation of the Instance Generator after adding non-service traffic trains can be used by the Research and Development Engineers of NS to more accurately calculate the capacity of rail hubs in the Netherlands in the future. The engineers at the NS Research and Development Logistics hub point out that adding the non-service traffic, as described in this paper, is an important step towards generating realistic instances.

In the future, one could go one step further and simulate disturbances of the non-service train traffic. This would give an even more realistic picture since trains do not always arrive according to their scheduled times. The effect of disturbances on generated train instances and a rail hub's capacity to deal with disturbances are interesting topics to investigate in the future. The Research and Development engineers are also planning to look into how non-service traffic can be adjusted to increase the capacity of rail hubs in cases where conflicts with service traffic trains arise.

Besides that, it would be interesting to repeat the tests as described in section 3.3 with a bigger rail hub than the hub in Heerlen as testing location. This could provide even better insights into the effects of the extension of the instances with non-service train traffic. The impact of security distance on maximum train density could also be investigated further by analysing more values than the three that were chosen for this research.

ACKNOWLEDGMENTS

I would like to thank the R&D Hub logistics at the Nederlandse Spoorwegen. Especially my supervisor Wan-Jui Lee for all her guidance, encouragement and support, Demian de Ruijter for the extensive explanations on the Dutch railroad network, and Bob Huisman for providing me with this research task at his department. My sincere gratitude also goes to Doina Bucur and Carlos Budde, my supervisors at the University of Twente, for all the valuable feedback, interesting discussions and advice along the way.

6. REFERENCES

[1] D'Ariano, A., Pacciarelli, D., & Pranzo, M. (2007). A branch and bound algorithm for scheduling trains in a railway network. *European Journal of Operational Research*, *183*(2), 643–657. https://doi.org/10.1016/j.ejor.2006.10.034

[2] Björkqvist, J., & Westerlund, T. (1999). Automated reformulation of disjunctive constraints in MINLP optimization. *Computers & Chemical Engineering*, *23*, S11–S14. https://doi.org/10.1016/s0098-1354(99)80004-0

[3] Freling, R., Lentink, R. M., Kroon, L. G., & Huisman, D. (2005). Shunting of passenger train units in a railway station. *Transportation Science*, *39*(2), 261–272. https://doi.org/10.1287/trsc.1030.0076

[4] Hassannayebi, E., Sajedinejad, A., & Mardani, S. (2014). Urban rail transit planning using a two-stage simulation-based optimization approach. *Simulation Modelling Practice and Theory*, *49*, 151–166. https://doi.org/10.1016/j.simpat.2014.09.004

[5] NS. (2020). Nederland duurzaam bereikbaar. Voor iedereen. Retrieved from https://www.ns.nl/binaries/_ht_1582797451971/content/assets/ns-nl/over-ns/nederland-duurzaam-bereikbaar.pdf

[6] Peer, Evertjan & Menkovski, Vlado & Zhang, Yingqian & Lee, Wan-Jui. (2018). Shunting Trains with Deep Reinforcement Learning.

[7] Sajedinejad, A., Mardani, S., Hasannayebi, E., K., S. A. R. M. M., & Kabirian, A. (2011). SIMARAIL: Simulation based optimization software for scheduling railway network. *Proceedings - Winter Simulation Conference*, 3730–3741. https://doi.org/10.1109/WSC.2011.6148066

[8] van den Broek, R. (2016). Train shunting and service scheduling: an integrated local search approach. Master's thesis, Utrecht University