DOI: 10.1111/mice.12764

ORIGINAL ARTICLE

COMPUTER-AIDED CIVIL AND INFRASTRUCTURE ENGINEERING WILEY

Optimization of annual planned rail maintenance

Menno Oudshoorn[1,2] | **Timo Koppenberg**[2] | **Neil Yorke-Smith**[1]

[1] Algorithmics group, Delft University of Technology, Delft, The Netherlands

[2] Macomi B.V., Rotterdam, The Netherlands

Correspondence
Neil Yorke-Smith, Delft University of Technology, P.O. Box 5031, 2600 GA Delft, The Netherlands.
Email: n.yorke-smith@tudelft.nl

Abstract

Research on preventative rail maintenance to date majors on small or artificial problem instances, not applicable to real-world use cases. This article tackles large, real-world rail maintenance scheduling problems. Maintenance costs and availability of the infrastructure need to be optimized, while adhering to a set of complex constraints. We develop and compare three generic approaches: an evolution strategy, a greedy metaheuristic, and a hybrid of the two. As a case study, we schedule major preventive maintenance of a full year in the complete rail infrastructure of the Netherlands, one of the busiest rail networks of Europe. Empirical results on two real-world datasets show the hybrid approach delivers high-quality schedules.

1 | INTRODUCTION

For the optimal condition of railway infrastructure it is imperative to ensure a safe and durable network, and to minimize the number of unexpected failures—causes of major disruptions to the train schedule and high costs for corrective maintenance.

As a case study, the Dutch railway network contains more than 7000 km of railway track and is one of the busiest railway networks in Europe. In 2018, a total of 165 million kilometres were driven by passenger trains, and a total of 57 billion tonne-kilometres were driven by goods trains (ProRail, 2020). The number of trains and passengers using the network is growing annually; and the demands on the European rail network are expected to keep increasing until 2040.

Much research has been done on preventative rail maintenance scheduling (Consilvio et al., 2020). However, the problems studied in the academic literature are mostly small and artificial. The methods used to solve these problems work well on small instances, but it is unclear how they would scale to a large real-world problem with complex constraints, such as in the Netherlands and other rail-heavy countries.

Against this background, we make the following contributions to the state of the art. First, we develop and benchmark three nonexact solution methods for the railway planned maintenance problem: an evolution strategy, a greedy algorithm, and a hybrid between these two. We provide insight into the performance of these algorithms in practice.

Second, we provide a study of a large-scale real-world case. The instances that come from real data require over 600 maintenance jobs to be scheduled, with more than 8000 options for each job. Further, there exist complex, noncontinuous constraints that severely limit the feasible search space. This means exact methods such as mixed integer programming (MIP) solvers, which are often used to solve the indicated small instances of related problems (Budai et al., 2006), are not suitable to solve this problem.

Third, we provide solutions to a real-world national-level maintenance scheduling problem, which are of better quality than the schedules currently being used.

This study was conducted in conjunction with ProRail, the organization that has the sole responsibility to plan and schedule the preventive maintenance for the whole of the Netherlands. ProRail is also responsible for unplanned corrective maintenance. For commercial confidentiality, all

results have been scaled by an arbitrary large constant, and the years of the datasets are redacted. The schedule developed informed ProRail in its planning for the year 2021 and beyond.

2 | RELATED WORK

Rail maintenance scheduling is a broad field where different types of problems are considered. First, the temporal granularity varies, from short to medium term maintenance planning to longer term where strategic and investment decisions are relevant (Simson et al., 2000; Sousa et al., 2019). Second, the geographic granularity varies, from taking into account individual tracks and switches to looking at a higher level network of, for example, an entire city. Third, some studies do not only consider scheduling the maintenance but also deciding what maintenance needs to be scheduled based on some kind of deterioration model (Ghofrani et al., 2020) and the trade-off between costs and chance of failure (Mohammadi et al., 2020); Memarzadeh and Pozzi (2016) also consider the value of information in gathering data about deterioration. Others assume that a list of necessary maintenance jobs is known and only look at when to schedule them. Fourth, the use of periodic and aperiodic tasks varies: both these types of tasks, as well as their combination, are considered in literature. Lastly, the literature is interested in planned (scheduled) maintenance—the focus of this article—proactive (data-driven) maintenance, and emergency (corrective) maintenance. Lopes Gerum et al. (2019), who combine predictive and planned maintenance, provide a survey. Other works study railway timetabling (Yin et al., 2019), or automated inspection (Guo et al., 2021).

Budai et al. (2006) introduce the preventive maintenance scheduling problem (PMSP) where a set of routine tasks, as well as one-time projects, must be scheduled. Similarities with the problem treated in this article are that possession and maintenance costs are being used, as well as similarity in some constraints such as "project a is combinable with project b." However, there are key differences: the PMSP does not consider locations and location conflicts, and all costs are completely independent unlike in the problem we address. Budai et al.'s problem is initially solved by an MIP solver, but small ($\pm$15 tasks) problems already could take more than 3 h. Recognizing that exact methods are inappropriate, the authors propose several greedy heuristics.

Separately, Peng et al. (2011) introduce the track maintenance scheduling problem (TMSP). Like the PMSP, a set of projects must be scheduled; the maintenance teams are also considered. Thus, the objective is to minimize the total travel costs of the maintenance teams together with the impact on railway operation. The authors present an iterative heuristic approach, and apply it to a "large-scale real-world problem" with 333 projects and 21 teams over 48 weeks. The resulting sum of travel costs and soft constraint violation penalties was a two-thirds decrease over the human scheduler's solution. Later papers on the TMSP are, for example, Xie et al. (2018), who cast it as a vehicle routing problem with time windows, and Su et al. (2019), who study part of the Dutch regional rail network. Our work does not schedule the maintenance teams, but has more complicated hard constraints and soft constraints, and a real-world case study of larger size than these authors. Another direction initiated by Peng and Ouyang (2014) is clustering track maintenance jobs into projects, with the intent of assigning projects to teams.

A sequence of authors apply various algorithmic approaches to the PMSP and variants; Soh et al. (2012) is one survey. For example, Budai-Balke et al. (2009) were among the first to test various genetic and memetic algorithms (GA, MA) on the PMSP. Adding the local search to the GA improved performance for most instances tested.

Andrade and Teixeira (2011) use two objectives: maintenance costs and train delays. The authors focus on the deterioration model and deciding whether maintenance should be performed at all. Biobjective simulated annealing is used to solve the problem. Zhang et al. (2012) use a deterioration model and consider "importance" of track segments, which is similar to the notion of affected passengers for certain subcorridors that arise in our problem.

Khalouli et al. (2016) test the PMSP model with a 2-year horizon and approximately 30 projects. Using the CPLEX MIP solver the authors manage to solve (only) 62% of instances within 3 h. Kiefer et al. (2018) also formulate their problem as an MIP model but quickly find out its runtime limitations on a larger instance, exceeding the 24-h time limit. Therefore, they develop a metaheuristic based on large neighborhood search. They test it on the Vienna tram network, the largest instance encountered in the literature.

Peralta et al. (2018) use a multiobjective approach with cost and delay as objectives. They use two multiobjective methods: AMOSA (Bandyopadhyay et al., 2008) and NSGA-II (Deb et al., 2002). They also use a nonrandom initialization; solutions are generated following certain expert heuristic rules. Such rules are also available for our case study, although we will not use them due to the relatively poor optimality of the manual schedules.

Summarizing, the literature does not explain how to address simultaneously all of the following for preventative maintenance planning: (1) realistic hard and soft constraints; (2) multiple (at least two) objectives; (3) large scale, beyond city or regional size; and (4) computational tractability within 24 h.

Besides works on the PMSP and variants, it is possible to combine scheduling train traffic with scheduling rail maintenance; among the recent works are D'Ariano et al. (2019) and Zhang et al. (2020).

Lastly, we note that other engineering maintenance scheduling problems with related characteristics have also been studied, such as road maintenance and power grid maintenance scheduling problems. Both these problems also consider maintenance scheduling in some kind of network-based structure. Typical solution approaches are again inexact: genetic algorithms (Cheu et al., 2004), metaheuristics (Froger et al., 2016), reinforcement learning based on Markov decision processes (Gao & Zhang, 2013; Medury & Madanat, 2013), and simulation–optimization approaches (Shahmoradi-Moghadam et al., 2021).

3 | PROBLEM STATEMENT

3.1 | Maintenance planning and scheduling

Maintenance and constructing new infrastructure accounts for a large part of the yearly costs rail infrastructure managers such as DB Netze Track, ProRail, and SNCF Réseau. Further, performing maintenance causes trains to be *hindered*, which causes major disruptions and possible capacity problems on detour routes (Zhang et al., 2020). There is patent incentive to create a maintenance schedule in such a way that *maintenance costs* and *unavailability* of the track are minimized, while still ensuring reliability by not decreasing the amount of maintenance that is performed. The problem addressed in this article focuses on the costs and feasibility of this planned maintenance schedule.

Unavailability of rail service is a cost to the rail users. It is accounted as cost from the perspective from the rail manager because of contractual and regulatory obligations. For example, ProRail has a business objective of minimizing unavailability to freight and passengers.

Maintenance planning and scheduling is a large operation consisting of multiple stages. ProRail's approach is typical: the smallest unit of maintenance considered is an *asset operation*, describing a piece of maintenance or new infrastructure, which has to be performed at a certain part of the network. Asset operations can vary in length and costs. For example, consider the conservation of a signal and the replacement of a "recloser": the two operations take the same amount of time—about an hour—but the costs are around € 1000 and € 10,000, respectively. Another common asset operation, the complete renewal of rail, is usually planned for 200–1500 m in a single operation and

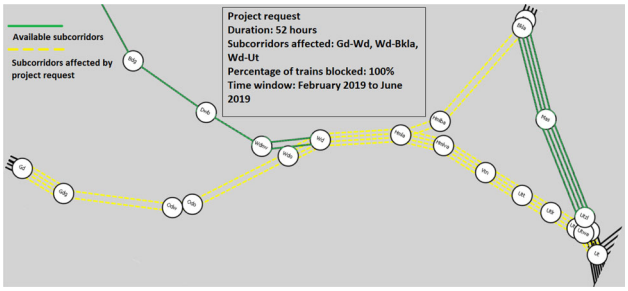


FIGURE 1 Request (in box) and affected subcorridor

FIGURE 2 Dutch railway network

can have a cost as high as half a million euros and a duration of up to 11 h.

Each year, a number of asset operations are selected to be performed, and they are clustered to *projects*. The managers of each project can then execute one or more *project requests*. A project request is a request to work on certain subcorridors of the network, for which those subcorridors *might* have to be taken out of service. A subcorridor is part of one or more corridors, which are parts of the rail network with an ongoing transport flow (Figure 1).

When the project requests for a given year are known, they can be scheduled. In the case of the Netherlands (Figure 2), this scheduling is done manually by human experts, and therefore takes a long time; usually, only a single schedule is created for each year. Due to the complexity and size of the problem together with having to deal with incomplete data, unmodeled constraint logic, and

involvement of other stakeholders, the human schedulers have difficulty making a schedule that adheres to as many constraints as possible and is even somewhat optimal in terms of costs.

In practice, the problem as given is usually overconstrained even in terms of hard constraints only, because the rail infrastructure managers, on one hand, simply specify too many requests, and on the other hand, because they are unaware of collective feasibility of the set of requests. A discussion process begins a the computed schedule. Thus a solution to the maintenance planning problem at a technical level—as we provide is this article—must be seen as part of a larger sociotechnical decision process involving discussion among multiple stakeholders. We make an important contribution to this process by providing solutions with fewer constraint violations and lower cost, obtained in much less time.

3.2 | Detailed problem description

The scheduling problem consists of a number of project requests in a given time period, usually a year. Given a list of project requests, each request must be assigned a start time, with an hour granularity. This must be done such that costs are minimized while respecting constraints. Both hard constraints, which in principle must always be adhered to despite the effect on costs, and soft constraints, which give a cost penalty when broken, are present in the problem. Scheduling *within* a project request is performed at a later stage of the planning process, closer to execution: it is out of scope of the higher level problem here.

For each project request, the following information is given: (1) the duration (in whole hours); (2) the set of subcorridors the project request affects; (3) a set of prerequisite project requests; (4) a time window indicating the period in which the project request must be scheduled; (5) the maintenance costs of the project requests, divided into three categories: personnel costs, security costs, and constant costs; (6) a percentage of passenger trains and freight trains that will be blocked by this project requests on the given subcorridor; when zero, the project request does not cause a disruption at all. Further, for each subcorridor, the following information is given: (1) the set of corridors of which it is part; (2) the number of passengers and freight trains estimated to travel through it, for any given hour in the global plan window; (3) the number of extra travel minutes each passenger can expect when the subcorridor is out of service; (4) the percentage of travelers taking replacement buses rather than a detour route by train when the subcorridor is out of service; (5) the fine that has to be paid per affected freight train when the subcorridor is out of service.

Note that we represent the maintenance actions through their length, affected area and costs: the details of the specific actions are not needed. Further, the more low-level decision of bundling the smaller parts of maintenance to the TVPs (see Section 3.2.2) is a later step in the process, not in scope here. We exclude staff scheduling from this article in order to limit the scope. However, the actual problem we tackled includes staffing constraints, which our approach successfully integrates.

3.2.1 | Problem objectives

The maintenance scheduling problem is a biobjective problem in terms of costs, together with an objective derived from the satisfaction of constraints.

Maintenance costs

The maintenance costs are the costs for actually performing the maintenance and are paid to the contracted maintenance company. There are three types of maintenance costs: security costs, personnel costs, and constant costs. The constant costs are independent of the planned moment of a project request and are therefore not optimizable.

The personnel costs are calculated per hour and subcorridor, for each project request. First, the hourly costs are computed by dividing the total personnel costs by the duration. Then, for each hour in the planned period, the hourly costs are multiplied by a given personnel multiplier for that hour. These are then all added together to find the total personnel costs. When the duration of a project request is less than 8 h, the costs are scaled up to a duration of 8 h to represent the minimum time of a personnel shift. Finally, when two short project requests on the same subcorridor are planned directly after each other, their personnel costs are scaled up together.

The security costs are costs for securing the workplace for the maintenance crew. These costs are calculated per *overlapping period* in a subcorridor. An overlapping period indicates a period in time in which at least one project request is planned at all times. For each overlapping period, the security costs are the maximum security costs of all project requests in the overlapping period. They are given as a fixed cost.

The sum of the security, personnel, and constant costs make up the first cost objective of the problem.

Availability costs

The availability costs are costs associated with the (un)availability of the railway infrastructure and the impediment experienced by passengers and freight trains. Some of the availability costs are not actual monetary costs,

but rather the impediment experienced, expressed as a cost value.

The first part of the availability costs are those for passenger impediment. They are computed per hour and subcorridor. The maximum degree of impediment of project requests scheduled at that hour and subcorridor is found. It is also given how many passengers will travel through the subcorridor at that time, and the extra travel minutes they are expected to experience. Finally, a static month multiplier is given for each month of the year. These are all multiplied to find the total extra travel minutes, which is then multiplied by a cost per extra travel minute. The costs are increased by 30% for each passenger expected to travel by replacement bus because of the relative discomfort this brings to the passenger. Finally, the costs are multiplied by the found maximum degree of impediment.

The costs for freight train impediment are computed similarly. The maximum degree of impediment for freight trains is found, and is multiplied by the expected number of affected freight trains and the fine per freight train. This is done for each hour and subcorridor. Finally, a number of alternative travel costs have to be paid. These are computed per overlapping period for each subcorridor. This can be done by computing the number of affected passengers, and a given function maps this to a number of alternative travel costs.

The sum of the impediment costs for both passengers and freight, and the alternative travel costs, make up the second cost objective of the problem.

In addition to these two explicit cost objectives, we have the third objective of maximizing satisfaction of the soft constraints, explained further below. We will aggregate the three objectives as a single weighted sum objective function representing total costs, and explore trade-offs between this cost and the number of hard constraint violations.

3.2.2 | Problem constraints

Both soft and hard constraints are present. Several constraints are conditional: they must be enforced only for requests *with hinder*, meaning the request has a degree of impediment for either passenger or freight trains larger than zero.

Conflicts

There are five different conflict types that specify combinations of subcorridors that may not be taken out of service at the same time. These types are (1) corridor conflict: trains traveling over one single corridor should experience impediment at most once; (2) passenger detour conflict: passengers must have an alternative route available so they can still get to their destination; (3) goods detour conflict: goods trains must have an alternative route available so they can still get to their destination; (4) junction conflict: certain important junctions must stay reachable; (5) border conflict: trains must be able to cross the border at least one location.

There may never be two project requests planned at the same time if there is a conflict between any of their subcorridors. A pair of requests can create only a single conflict violation. The junction conflicts are a soft constraint; the other four conflict types are hard constraints. This constraint is only enforced for project requests with hinder.

Dependencies

Dependency constraints specify combinations of a subcorridor and a time period where planning maintenance is prohibited. This constraint is only enforced for project requests with hinder.

Corridor constraints

The corridor constraints are soft constraints. They apply to overlapping periods, which are, as explained before, continuous periods in time for a given corridor or subcorridor in which at least one request is planned at all times. Only requests with hinder are used to find the overlapping periods. In these constraints, the notion of TVP is used. *TVP* stands for a "train-free period" and is used to describe an extended period during which the rail is unavailable due to one or more project requests. Due to the rules that ProRail uses for this constraint, an overlapping period only counts as a TVP if it is at least 24-h long.

The *max TVP corridor* constraint specifies the maximum number of TVPs that can be present in a corridor, in the complete global plan window. The *mintime between TVPs corridor* constraint specifies the minimum number of hours that must be between two TVPs within the same corridor.

Required time window

A project request may have a required time window that is more restrictive than the global plan window, in which the request must be planned. This is a hard constraint.

Max simultaneous requests at a subcorridor

A maximum number of project requests that may be scheduled simultaneously at a subcorridor is given. This is a soft constraint.

3.3 | Mathematical model of the problem

We can now present a complete model of the maintenance scheduling problem as a constraint programming model.

Table 1 summarizes the parameters, functions, and predicates used in the mathematical model.

Decision variables

$x_i \in \mathbb{N},\ 0 \le x_i < T_{\text{end}},\ i = 1, \dots, N$: start time of project request i in number of hours since T_{start}

Objective functions

$$\begin{aligned} f_1(\mathbf{x}) = \sum_{t=0}^{T_{\text{end}}} \sum_{sc \in SC} & \text{pbMax}(sc,t) \\ & \cdot (TR_{sc}^{t} \cdot MO_t \cdot ERM_{sc} \\ & \cdot (1 + 0.3BUS_{sc}) \cdot ermCost) \\ & + \text{atc}\big(\text{pbMax}(sc,t) \cdot TR_{sc}^{t} \cdot MO_t\big) \\ & + \text{gbMax}(sc,i) \cdot \left(FT_{sc}^{t} \cdot FINE_{sc}\right) \end{aligned} \tag{1a}$$

$$\begin{aligned} f_2(\mathbf{x}) = \sum_{i \in P} \left(COO_i + \sum_{t=x_i}^{x_i + \text{scale}(d_i,8)} PCM_t \cdot \frac{COP_i}{d_i} \right) \\ + \sum_{sc \in SC} \sum_{tvp \in getTVPs(\text{affects subcorridor }(sc))} \max_{i \in tvp} \frac{COS_i}{|SC_i|} \end{aligned} \tag{1b}$$

Equation 1a calculates the availability costs. For each hour and subcorridor, the passenger impediment costs, alternative travel costs, and freight impediment costs are computed. Equation (1b) calculates the maintenance costs. The first part is computed per request, and contains the constant costs and the possibly scaled, personnel costs. The second part is computed per overlapping period and subcorridor and finds the maximum security costs for each of them. Here, $\frac{COS_i}{|SC_i|}$ denotes the security costs of project request i per affected subcorridor.

Hard constraints

$$\begin{aligned} & \forall i \in P : requiredWindow(i) \\ & \Rightarrow \max(0, TWS_i) \le x_i \le x_i + d_i \le \min(T_{\text{end}}, TWE_i) \end{aligned} \tag{2a}$$

$$\begin{aligned} & \forall i, j \in P : hinders(i) \wedge hinders(j) \wedge conflict(i,j) \\ & \Rightarrow \neg overlap(i,j) \end{aligned} \tag{2b}$$

$$\begin{aligned} & \forall i \in P, d \in D : hinders(i) \wedge dependency(i,d) \\ & \Rightarrow \neg overlap(i,d) \end{aligned} \tag{2c}$$

Constraint 2a enforces the time window constraint. Each project request must be planned within the global plan window, and within the more restrictive time window if it is given. Constraint 2b enforces the conflict constraints. For each pair of project requests with a potential conflict, overlap is not allowed if they both cause hinder. Constraint 2c enforces the dependency constraints. Given a dependency and a project requests with hinder that potentially conflict, the two may not overlap.

Soft constraints

We model the satisfaction of the soft constraints by aggregating them into a single penalty function. Some helper functions are defined in Table 1 (bottom). Then, the equations to compute the soft constraint penalties are

$$P_{co}(\mathbf{x}) = (10,000 \cdot cOV) \tag{3a}$$

$$P_{mTVPc}(\mathbf{x}) = \sum_{c \in C} (20,000 \cdot \max TVPC(c)) \tag{3b}$$

$$\begin{aligned} P_{mtTVP}(\mathbf{x}) = \sum_{c \in C} \Bigg(\sum_{(t1,t2) \in getTVPs((getHinders() \cap affectsCorridor(c)))} & (10,000 \\ & \cdot mTV(t1,t2)) \Bigg) \end{aligned} \tag{3c}$$

$$P_{msc}(\mathbf{x}) = \sum_{sc \in SC} (2^{mOV(sc)} \cdot 20,000) \tag{3d}$$

The first penalty (3a) corresponds to the junction conflicts. (3b) arises from the max TVP corridor constraints, and (3c) from the mintime between TVPs constraint. (3d) is the maximum simultaneous requests at a subcorridor.

The total of the soft constraint penalties is defined as follows, as the unweighted sum:

$$TotalSC(\mathbf{x}) = P_{mTVPc}(\mathbf{x}) + P_{mtTVP}(\mathbf{x}) + P_{co}(\mathbf{x}) + P_{msc}(\mathbf{x}) \tag{4}$$

Goal of the optimization

When formulated with a single-objective function, the goal of the optimization is to find $\mathbf{x} = \{x_1, \dots, x_N\}$ such that

$$\chi_1 f_1(\mathbf{x}) + \chi_2 f_2(\mathbf{x}) + \chi_3 TotalSC(\mathbf{x}) \tag{5}$$

TABLE 1 Parameters, predicates, and functions for the mathematical model of the problem

Parameters	
i, j	Project request indices
$c\,sc\,t\,ty$	Corridor index/subcorridor index/time index/maintenance type index
$P\,D\,C$	Set of all projects/set of all dependencies/set of all corridors
$SC\,SC_i$	Set of all subcorridors/set of subcorridors affected by project request i
T_{start}	Start date of the global plan period
T_{end}	Number of hours in the global plan window
d_i	Duration of project request i
TWS_i	Time before which project request i may not be started
TWE_i	Time before which project request i must be finished
N:	Number of project requests
maxCor_c	Maximum number of TVPs on corridor c
$mwsc$	Maximum number of weekends affected by TVPs per subcorridor
mwc	Maximum number of weekends affected by TVPs per corridor
mpi	Minimum number of hours that should be between two TVPs in a corridor
maxSC	Maximum number of project requests that may be active at one subcorridor at any point in time
TR^t_{sc}	Number of passengers estimated to go through subcorridor sc at time t
$ermCost$	Cost of an extra minute of travel
ERM_{sc}	Number of extra minutes of travel per passenger when subcorridor sc is taken out of service
BUS_{sc}	Percentage of travelers having to take replacement buses when subcorridor sc is taken out of service
FT^t_{sc}	Number of freight trains expected to go through subcorridor sc at time t
ATC_p	Amount of alternative travel costs when p passengers are affected
$FINE_{sc}$	Fine that has to be paid per freight train when subcorridor sc is taken out of service
$COP_i\,COS_i\,COO_i$	Personnel costs/security costs/constant costs of project request i
PCM_t	Personnel cost multiplier at time t
MO_t	Month multiplier at time t
Predicates	
$overlap(i, j)$	True if project requests i and j overlap in time
$overlap(i, t_1, t_2)$	True if project request i overlaps with the time window (t_1, t_2)
$hinders(i)$	True if project request i causes a disruption by hindering (part of) the affected infrastructure
$conflict(i, j)$	True if project requests i and j would cause a conflict violation when planned in an overlapping manner
$dependency(i, d)$	True if dependency d is relevant to project request i, i.e., when project request i is planned during the specified subcorridor and time window of d, a dependency violation would arise
$requiredWindow(i)$	True if project request i has a required window which is tighter than the global plan window on at least one side
$requestActive(P, t)$	True if any request in the set P has a planned period which overlaps with time t
Functions	
$overlapping(t)$	Returns the set of all project requests whose planned period overlaps with time t
$overlapping(t, P)$	Returns the set of all project requests in P whose planned period overlaps with time t
$affectsCorridor(c)$	Returns the set of all project requests affecting corridor c
$affectsSubcorridor(sc)$	Returns the set of all project requests affecting subcorridor sc
$getTVPs(P)$	Returns a set of TVPs, i.e., overlapping periods of at least 24 h, from the requests in set P. A TVP will have a start and end date, as well as a list of requests that are part of the TVP.
$getHinders()$	Returns all project requests i for which hinders(i) is true
$timeBetween(tvp_1, tvp_2)$	Returns the time between TVPs tvp_1 and tvp_2
pbMax(sc, t)	Returns the maximum passenger blockage of project requests affecting subcorridor sc at time t
gbMax(sc, t)	Returns the maximum goods train blockage of project requests affecting subcorridor sc at time t

(Continues)

TABLE 1 (Continued)

Functions	
$scale(x, n)$	Returns the number x scaled up to the nearest multiple of n
$atc(n)$	Returns the alternative travel costs for an overlapping period in which n passengers are affected
Soft constraint penalty helper functions	
cOV	$\lvert\{(i,j) \lvert hinders(i) \wedge hinders(j) \wedge conflict(i,j) \wedge overlap(i,j)\}\rvert$
max$TVPC(c)$	$\max(0, \lvert getTVP(getHinders() \cap affectsCorridor(c))\rvert - maxCor_c)$
$mTV(t1, t2)$	$\max(0, mpi - \text{timeBetween}(t1, t2))$
$mOV(sc)$	$\max_{1 \le t \le T_{end}}(\lvert\text{overlapping}(t) \cap \text{affectsSubcorridor}(sc)\rvert) - maxAtOneLocation$

is minimized and all hard constraints are satisfied. In the case study of Section 5, the weights χ_i are set uniformly to 1, according to the preference of the domain experts.

4 | ALGORITHMS

This article approaches the rail maintenance planning problem of Section 3.3 by developing three algorithmic approaches. All three are inexact methods, in light of, first the complexity of the problem studied, and second, the desire by the stakeholders to find (only) a sufficiently good solution. Attempting to find one "optimal" solution is inappropriate due to, on one hand, the multistakeholder decision process, which is informed by the mathematical optimization—but which is not (and cannot be) incorporated into it—and on the other hand, the two objectives, costs and constraint violations, whose trade-off is ultimately a managerial decision by the rail management company.

The first algorithm is an evolution strategy: a population-based local search method. The second is a greedy algorithm, which tries to find globally (near) optimal solutions by making locally optimal choices. The third algorithm is a hybrid, which aims to combine the strengths of the first two approaches to provide solutions with better quality.

4.1 | Evolution strategy

The motivation to consider an evolution strategy (ES) is, first, their effectiveness in complex multiobjective problems, and second, the popularity of genetic and evolutionary algorithms in maintenance planning, as Section 2 surveys. A member of the family of evolutionary algorithms, an evolution strategy uses small mutations to change a population of individuals and explore the search space (Emmerich et al., 2018). Algorithm 1 shows pseudocode.

Algorithm 1: Evolution strategy

```
Initialize parent population P_μ repeat
    repeat
        Select randomly ρ parents from P_μ
        Recombine the ρ selected parents to form an offspring r
    until λ offspring are generated;
    if comma-selection is used then
        The new parent population is determined from the
          offspring population
    else
        The new parent population is determined from both the
          old parent population and the offspring population
until a termination criterion is fulfilled;
```

The termination criterion is a given number of generations. In the case study, this number was set so the algorithm terminated in approximately 24 h. This allows the manager to run the algorithm overnight.

There are three parameters that need to be set: the parent size μ, the offspring size λ, and the selection type: either offspring only (comma selection) or both parents and offspring (plus selection). Selection of the next parent population is done deterministically. Note that the fitness of individuals is used when selecting the parent. In our implementation we used plus-selection so well-performing parents will not be discarded. We experimented with crossover and found it ineffective, due to the interdependencies between elements.

4.1.1 | Mutations

With ρ being equal to 1, there is no recombination and an offspring individual is created by applying a mutation to a parent individual. Specifying good mutation(s) is arguably the most important part of creating an effective evolution strategy. Several problem-specific mutations have been created, which focus on decreasing costs and constraint violations, as well as (random) exploration of the search space.

Day and hour mutations
These mutations move a randomly selected project request to a random other day or hour, respectively. The required time window of a selected request is taken into account, but other constraints are not checked; these mutations are primarily for exploration of the search space.

Bucket mutations
Buckets are groups of project requests that are non-conflicting and affect similar subcorridors. By planning buckets in an overlapping manner, both availability and maintenance costs decrease. There are three bucket mutations:

- *Create bucket mutation*: Find two project requests that are currently not in a bucket and are eligible to be created into a bucket. Create a bucket from those two requests. The bucket is planned at the original plan moment of one of the two requests with equal chance.
- *Expand bucket mutation*: Take a random request that is not in a bucket yet and try to find an existing bucket that would accept the request. If no such bucket is found, retry with a different request until a set number of tries. If multiple eligible buckets are found, choose one at random.
- *Shrink bucket mutation*: Take a random bucket and move one request out of the bucket and plan it at a random other time. If the bucket contained only two requests, the bucket will be removed.

Fix constraint mutations
These mutations try to decrease the number of hard constraint violations, specifically through decreasing conflict and dependency violations as these are the hardest constraints to satisfy. There are separate mutations for conflicts and for dependencies; they work in a similar way. First, a project request that is currently in violation is found. Then, possible time windows where it would not cause a violation are computed. If at least one long-enough time window is found, the project request is moved to (one of those) time window(s). If no such time window is found, the project request is moved to a random other time. If possible, the starting *time* of the request is kept the same, such that only the date is changed: for example, a request planned at night is kept at night.

Each mutation is given a pre-set weight, and whenever an individual is mutated, one of the mutations is randomly selected using those weights. Future work could adaptively modify the weights, in the style of adaptive large neighborhood search.

4.1.2 | Constraint cooling

Initial tests showed that the evolution strategy was quite effective in decreasing the number of hard constraint violations. However, the cost optimizations were unsatisfactory; the algorithm became stuck in a local optimum, from which it could not escape without changing the solution in such a way that would initially increase the hard constraint violations again. Therefore a process called *constraint cooling* has been implemented, drawing on ideas from simulated annealing (Ingber, 1993). During the runtime of the algorithm, the number of allowed hard constraint violations is slowly decreased. Any solution with fewer violations than the allowed number at the time is considered feasible and is only compared on costs. The idea behind constraint cooling is that, instead of first optimizing the violations and then the costs, the two are optimized simultaneously. Initial tests showed that the final solutions had lower costs with cooling implemented.

The cooling schedule used is $C_t = C_0 \cdot \alpha^t$, with C_0 being the allowed constraints violations at the start of the algorithm, α the cooling factor, $0 < \alpha < 1$, t the number of the current generation, and C_t the allowed constraints at generation t. A minimum number of allowed violations is specified to lessen becoming stuck in a local optimum. In practice, given a maximum and minimum number of allowed violations, α is chosen such that the minimum is reached after two-thirds of the total number of generations. This means in the final one-third of the generations, the number of allowed violations is kept constant.

4.1.3 | Separation of requests with hinder

As explained in Section 3.2.2, some project requests do not have any passenger or freight impediment and therefore do not hinder the subcorridors they affect. These project requests do not cause any availability costs. Further, they are not part of the (most difficult) constraints; they do not cause conflicts or dependency violation, and are not taken into account when computing TVPs. These requests are *relatively* unimportant; almost all of the problem's difficulty is caused by the requests that do hinder. Therefore, the decision was made to separate the requests with hinder from those without. The evolution strategy is only run on the blocks with hinder, and the blocks without it are added greedily afterward. This is done by sorting these project requests descending on length, calculating the (estimated) personnel and security costs for each request for each possible plan moment and planning the request at the time which is optimal in terms of those costs.

Through this separation, the runtime of the evolution strategy is spent on optimizing the most difficult part of the problem, and no time is wasted on moving relatively unimportant project requests. Initial tests indicated almost no loss of solution quality when adding the project requests without hinder using the above greedy method. We next consider performing all the scheduling using a more advanced greedy algorithm.

4.2 | Greedy algorithm

Greedy algorithms follow the problem-solving heuristic of making a series of locally optimal choices, in the hope that this will lead to a solution that is (close to) globally optimal. The speed of a greedy algorithm is the motivation for exploring this approach. Algorithm 2 provides pseudocode

Algorithm 2: Greedy algorithm

```
1  Separate the project requests in two lists H (with hinder) and NH
     (without hinder)
2  Sort H descending on the expected affected passengers: the average
     number of affected passengers per hour for each of its locations
     times the length of the request in hours
3  Sort NH descending on length in hours
4  for each project request x ∈ H do
5      if length of x < 4 then
6          startTime ← 01:00
7      else
8          startTime ← 22:00
9      for each day d in the global plan window do
10         Plan x on d at startTime and keep track of the best result
11     Plan x on the best found result
12 for each project request x ∈ NH do
13     for each day d in the global plan window do
14         Plan x on d at 07:00 and keep track of the best result
15     Plan x on the found best result
```

for a greedy algorithm for the maintenance scheduling problem. The idea is to first schedule those requests with hinder (in order from most predicted disruption to the least) and then those without hinder (in order from the longest to shortest).

Order of project requests

Project requests with hinder are ordered by the expected impact they have, specifically on availability costs. To do this, both the length of a project request and the expected number of passengers for each of its affected subcorridors are taken into account. This way, a long project request that affects a very idle subcorridor does not get planned early on, because it would not affect that many passengers anyway. The requests are planned in reverse order of impact, so that the most impactful requests have the highest chance of finding an optimal plan moment. The requests without hinder are ordered on the length in hours, as they do not affect any passengers.

Starting time of a request

In order to decrease the runtime of the algorithm, we reduce the temporal granularity: not each hour in the global plan window is tried for each request; rather, a daily granularity is used. The start time for a request is chosen so that it provides the most potential for an optimal solution, according to the following heuristic. A request with hinder that takes fewer than 4 h is planned at 01:00, so that is finished before 05:00. The period between 01:00 and 05:00 is the least busy time, so the least passengers will be affected. A request with hinder that takes more than 4 h is always planned starting at 22:00. This way, a request which takes less than 8 h fits in a night, and a request that takes less than 56 h fits in a weekend. Finally, a request without hinder starts at 07:00, because personnel costs are the cheapest between 07:00 and 20:00, and these request do not cause availability costs so they do not need to be planned during a night or weekend.

Choosing the plan moment

Choosing the plan moment of a request is simple: while trying each possibility, we keep track of the best plan moment. Of course, hard constraint violations are more important than costs. It can occur that no possible plan moment can be found, for which the request would not create any hard constraint violations. In that case, a moment causing the least hard constraint violations is preferred.

Randomization

The greedy algorithm in its basic form is fully deterministic. This is contrary to the evolution strategy, which contains many random components. There are two main points in the algorithm in which randomization can be applied to the greedy algorithm: the order of the project requests, and the chosen plan moment. We considered three randomization methods as means to perturbation the determinization:

- *Roulette wheel selection*: Given some kind of metric per project request, select the next request to be planned by roulette wheel selection on that metric (Zhang et al., 2012). Optionally, only the first n requests are planned using this method and the remainder deterministically.
- *Randomize next request*: The requests are ordered with the deterministic order, but at each iteration, one of the top n requests is chosen to be planned next, according to a predefined distribution.
- *Randomize start time*: Instead of just keeping track of the best moment to plan a request, an ordered list of possibilities is kept. At the end of each iteration, one of the best n options is chosen using a predefined distribution.

TABLE 2 Characteristics of the two datasets

Characteristic	Year *X*	Year *Y*
Number of project requests	1033	688
With hinder	526	316
With required time window	305	484
Do not fit in required window	4	9
With prerequisites	0	0
With nonzero maintenance costs	455	530
Affected subcorridors per request	1/3.47/23	1/4.75/19
Length of requests in hours	3/26.7/576	2.33/49.5/744
Affected corridors per request	1/2.29/8	1/2.73/10
Potentially conflicting dependencies per request (with hinder)	0/41.3/308	0/159.7/1040
Potentially conflicting requests per request (with hinder)	0/181.3/467	0/117.3/277

Note: Apart from the first row, the table shows the min/mean/max.

These methods can also be combined. The effect of these randomizations are studied in Section 6.

4.3 | Hybrid greedy–ES

Initial tests showed that the evolution strategy and the greedy algorithm had complementary strengths and weaknesses. The evolution strategy was effective in minimizing costs, especially availability costs. The solutions provided by the greedy algorithm usually had slightly higher availability costs, but much lower soft constraint penalties. The motivation for combining the algorithms into a hybrid form is to combine the strengths of these algorithms to provide better solutions.

The two algorithms are combined in a multistage approach to create a iterated multilevel algorithm (Raidl et al., 2010). Pseudocode is given in Algorithm 3.

Algorithm 3: Hybrid greedy–ES algorithm

```
input : The number of stages n, n > 0, the number of project
        requests to be planned each stage {p_1, ..., p_n}, an ordered
        list of all project requests to be planned, and a number of
        generations for each stage {g_1, ..., g_n}
Using the randomized greedy algorithm, create a solution with the
  first p_1 project requests planned. Repeat this until an initial parent
  population for the evolution strategy is filled.
Run the evolution strategy on the created population for g_1
  generations.
for i = 2 to n do
    Take the resulting population from the previous evolution
      strategy for each individual in this population do
        Add the specified p_i requests using the randomized greedy
          algorithm
        Add the resulting individual to the next starting population
    Run the evolution strategy on the created population for g_i
      generations
return best solution of the final evolution strategy run
```

The hybrid algorithm works by repeatedly planning a number of requests greedily, running the evolution strategy on those solutions for a while, and adding more requests greedily to the best individuals of the evolution strategy. This is done until all requests have been planned.

5 | EXPERIMENTAL SETUP

5.1 | Datasets

Recall that our work was performed when the Year *X* Dutch rail maintenance plan had been decided, but prior to the Year *Y* plan. We received from ProRail the actual project request data for both Year *X* and Year *Y*, as well as the plan made by ProRail for Year *X*. Table 2 overviews characteristics of both datasets. Note that both are overconstrained problems: there is no solution that will satisfy all hard constraints. In practice, solutions to the mathematical model are input for the stakeholder discussion, which ultimately leads to a compromise solution and postponing (until the next year) the project requests which cause the remaining hard constraint violations.

The first part of the table shows the total number of project requests and the number of project requests with certain properties. Both datasets contain a substantial number of project requests to be planned. The Year *Y* data contains fewer requests than the Year *X* data, but in other characteristics it will become clear that despite having fewer requests, the requests in the Year *Y* data are generally longer and more impactful. Both datasets have an approximately 50/50 split between project requests which actually hinder the subcorridor and those that do not. In both datasets, there are a large number of requests that have a required window. Most of these required windows are quite large and do not limit the possibilities much. On the other hand, both datasets contain a number of project requests that do not fit in their own required

window. This means that it is impossible to find a solution with zero hard constraint violations. Due to missing data, neither dataset has any project request with prerequisites, both datasets contain some project requests that have zero maintenance costs.

Table 2 also shows the (average) impact of project requests. It can be seen that project requests in the Year *Y* data are generally longer and affect more passengers. Finally, the (potential) impact of certain constraints is shown. The number of affected corridors indicates the difficulty of the corridor constraints. The number of potentially conflicting requests and dependencies indicate the difficulty of the conflict and dependency constraints, respectively. It can be seen that in the Year *Y* data, the number of potentially conflicting dependencies is much higher on average. The number of potentially conflicting requests is relatively equal when taking the total number of requests into account.

As stated, for the Year *X* data the actual schedule created—manually—by ProRail is also available. Running this schedule through the formal model described in Section 3 reveals that the schedule has total costs of 965.2 (recall that all results have been scaled by an arbitrary large constant) and breaks 690 hard constraints. The manual schedule violates this large number of constraints due to: some unmodeled exceptions, some oversights such as missed dependencies during planning, and, most often, by conscious decision in stakeholder discussions. This solution can be used to compare the results of the different algorithm. For the Year *Y* data, at the time of our experiments, the actual schedule was not yet planned.

The Dutch Year *X* data are used as the primary dataset with which to test the algorithms. The Year *Y* data are used as a validation set, to see whether the conclusions drawn using the Year *X* data still hold. If this is the case, the conclusions have more robustness over changes in the input data.

5.2 | Experimental design

We will assess empirically the three methods proposed in Section 4 individually, and then compare them with each other. In addition, we compare with a multiobjective evolutionary algorithm from the literature, NSGA-II (Deb et al., 2002).

5.2.1 | Test of the evolution strategy

Because the evolution strategy contains many randomized components, a total of 53 runs have been performed on the Year *X* data. Further, six runs have been performed on the Year *Y* data. Each run of the algorithm was for 24 h on a 2.7 GHz CPU, 16 GB RAM machine. Table 3 shows the parameter values.

TABLE 3 Parameters for the evolution strategy

Parameter	Value
Parent size	40
Offspring size	170
Selection type	Parent and offspring
Number of parents (ρ)	1
Constraints allowed at start	1200
Constraints allowed at end	8

The parent size, offspring size, selection type, and the number of allowed constraints at the start have been set through an initial sensitivity analysis. A full parameter study is left as future research. The number of allowed constraints at the end of the algorithm is set to 8 rather than 0. This relaxation is warranted because, as shown in Table 2, there are a number of requests that do not fit in their required window and therefore *always* cause a constraint violation. Further, initial tests showed that almost all solutions had 6 or 7 constraint violations at the end. By setting the allowed constraints to 8, the slope of constraint cooling is somewhat shallower and this should allow the algorithm more time to optimize the costs while also gradually decreasing the number of constraint violations.

5.2.2 | Greedy algorithm

Both the deterministic and randomized versions of the greedy algorithm have been tested on the Year *X* data. Testing the deterministic version requires only a single run.

We study five methods of randomization: (1) roulette wheel selection, all requests; (2) roulette wheel selection, first 50 requests; (3) randomize next request, choosing one of the top 3 options with probability 0.5/0.35/0.15; (4) randomize start time, choosing one of the top 3 options with probability 0.5/0.35/0.15; (5) randomize both next request and starting time, choosing one of the top 3 options with probability 0.5/0.35/0.15. The probabilities for the last three methods are not based on parameter analysis, rather with the intuition of giving the better option more probability, but still choosing the second or third best option often enough of the time to have a significant impact on the results. Lastly, we can restrict roulette wheel to the first 50 requests only, because in the data, there are relatively few large requests with major impact, and many small requests. By taking only the first 50 requests, which will all

TABLE 4 Tested configurations of greedy–ES hybrid, Year *X* data

Run	Number of stages	Requests per stage	Runtime per stage (h)	Total costs	Violations
1	2	50/983	12/8	928.1	10
2	10	15/20/30/50/80/100/100/120/250/268	2/2.5/2.5/2.5/2/2/2/1.5/1.5/1.5	967.8	8
3	2	90/943	12/8	937.4	10
4	2	20/1013	8/12	932.8	10
5	3	20/70/943	7/7/6	934.5	10
6	2	50/983	16/4	949.8	10
7	2	50/983	4/16	961.2	10

be relatively impactful, there is still randomization but better results are expected. For each of these methods, 20 runs were performed. Each run of the greedy algorithm takes approximately 10–15 min on the same machine that was used to run the evolution strategy.

5.2.3 | Hybrid algorithm

One of the strengths, but at the same time also a weakness, of the hybrid algorithm is its customizability. By being able to fully specify the number of stages, requests per stage and runtime per stage, the algorithm is likely to be able to work well on different types of data. However, it is not obvious what the optimal values for these (hyper)parameters should be, and how they may change over a change in input data. To obtain an initial idea of the effect of changing the number of stages as well as the number of requests and runtime per stage, seven various configurations have been tested, as shown in Table 4. We discuss the results (right two columns) in the next section.

The first run is considered the baseline, and the following runs test various aspects of changing the parameters. In run 2, the number of stages is increased substantially. Runs 3 and 4 test what happens when the number of requests in stage 1 increases and decreases, respectively. Run 5 tests what happens when a single stage is added. In runs 6 and 7, the effect of changing the time distribution over the stages is tested.

The parameters for the evolution strategy part of the greedy algorithm are the similar to the parameters used in the experiments for the evolution strategy itself. For the greedy part, the "next request" randomization is used, with probabilities 0.5/0.35/0.15 to choose the best, second best, and third best request, respectively. The total runtime of evolution strategy is always set to be 20 h; the total time of the greedy algorithm is roughly 3–4 h, meaning that the total runtime will be approximately the same as the runtime of the runs of the evolution strategy.

6 | RESULTS

This section reports the results of the experiments described in Section 5. The scope is the whole of the Netherlands rail network, for Year *X* data and then for unseen Year *Y* data.

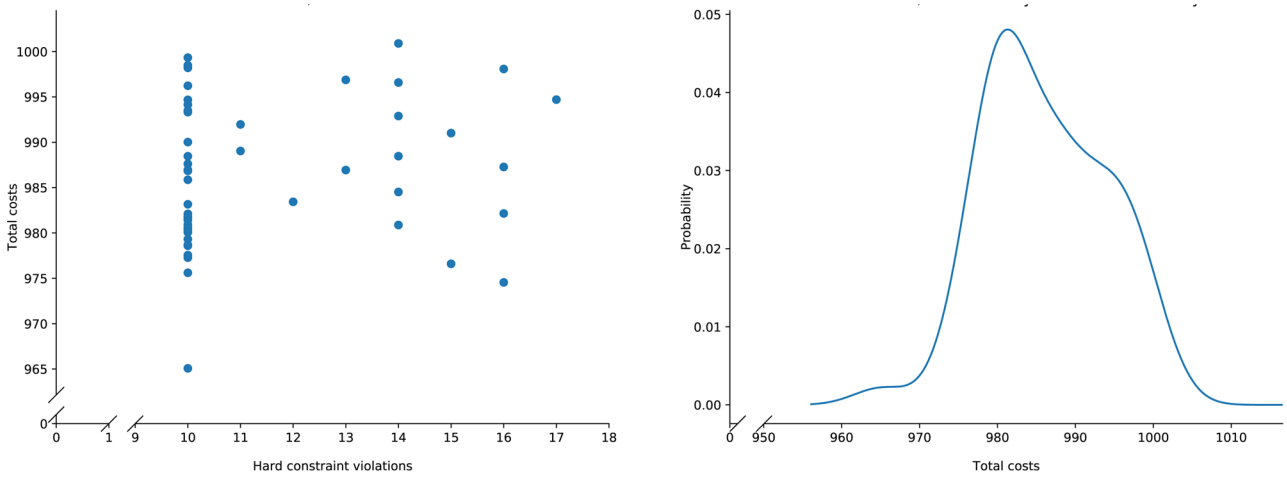
6.1 | Evolution strategy

6.1.1 | Year *X* data

As stated earlier, in total the evolution strategy was run 53 times on the Year *X* data. Figure 3a shows the solutions in terms of total costs and hard constraint violations.

A majority of the solutions have 10 hard constraint violations; 35 of 53 runs. The other solutions have 11–17 hard constraint violations. However, the relation between costs and constraints is not as expected. It might be expected that solutions that end up with slightly more constraint violations also reach lower costs because they theoretically have a bit more space for cost optimization. This relation is not visible though; the lowest cost solution is one with 10 constraint violations, and there are many solutions with 10 constraint violations and lower costs than the lowest cost solutions at higher constraint violations. It seems that the algorithm sometimes converges to a bad local optimum in terms of hard constraint violations, while the costs are in the same range as the other solutions.

Figure 3b shows a kernel density estimate of the costs of a solution of the evolution strategy. All solutions, also the ones with more than 10 hard constraint violations, have been taken into account. The average costs are 986.67 814.8 million with a standard deviation of 8.717.2 million. The distribution closely resembles a normal distribution. To confirm this, an Anderson-Darling test for normality has been performed on the data. This test returns a p-value of 0.555, and therefore does not reject the null hypothesis that the data is normally distributed under a



(a) Evolution strategy results, costs versus constraint violations

(b) Evolution strategy results, kernel density estimate of costs only

FIGURE 3 Results of evolution strategy experiments, Year *X* data

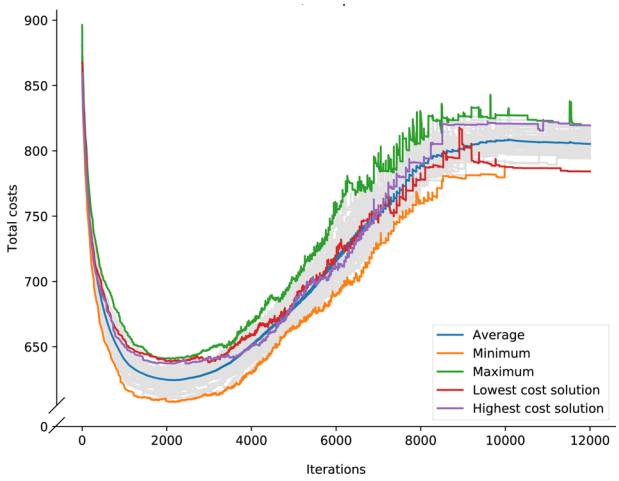


FIGURE 4 ES robustness analysis, Year *X* data

significance level of .05. Although there are no major outliers in the obtained solutions, the spread in terms of costs is still relatively large. This is a clear disadvantage of this algorithm. Doing a single run may yield an "unlucky" result, and when running the algorithm on new input data, multiple runs are required to get an idea of the average solution. Indeed, because the spread in costs is relatively high, it would be beneficial if there is a significant correlation between the quality of intermediate solutions and the end result of a single run. If so, early restarting could be implemented if an "unlucky" run is detected.

To test this hypothesis, the optimization traces of all runs are plotted in Figure 4. The behavior of the trace with the costs first going down and then up again is due to the constraint cooling process; these intermediate solutions have more hard constraint violations and their costs are not representative as a final solution. Looking at Figure 4, first it is visible that the spread in solution quality already

TABLE 5 Results of evolution strategy, Year *Y* data

Run	Total costs	Violations
1	1092.6	27
2	1101.5	27
3	1092.4	34
4	1086.7	25
5	1102.9	25
Mean	1095.2	27.6

appears early on in the algorithm: around 2000 iterations the spread seems to be almost as large as at the end of the algorithm. Unfortunately, there seems to be little correlation between the intermediate results and the end result of a run. Considering the traces for the lowest and highest cost solution, this becomes quite clear. Around 2000 iterations, the lowest cost solution almost has the highest costs. In the middle stages of the algorithm, the lines for the lowest and highest cost solution lie very close. Only at the end of the algorithm, they clearly separate. This means that is not possible to restart early on in the algorithm, and a full run is required to find out the end result.

6.1.2 | Year *Y* data

Five runs have been performed on the Year *Y* data, as shown in Table 5. Some preliminary conclusions can be drawn. First, the number of hard constraint violations in the Year *Y* results is higher than in Year *X*, even though the number of allowed constraint violations at the end of the algorithm was set to the same value. This indicates that for the Year *Y* data, it is harder to solve all hard constraints. A possible cause for this is the fact that there are many more

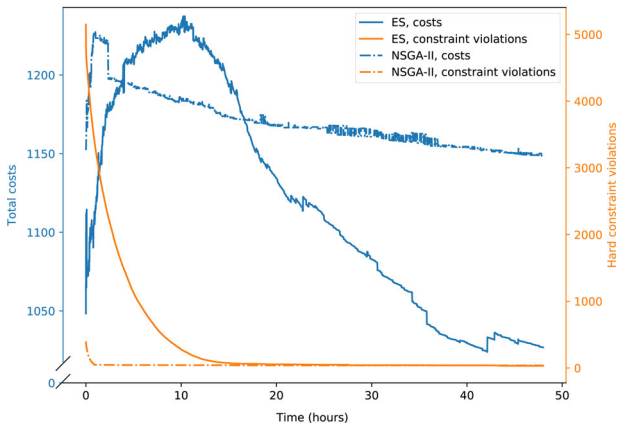


FIGURE 5 NSGA-II versus evolution strategy, Year *X* data

conflicting dependencies in the Year *Y* data compared to Year *X* as indicated in Section 5.1. Further, it is visible that there is still significant variability in the results. After seeing the results of the other algorithms, more comparative conclusions can be drawn.

6.2 | Multiobjective genetic algorithm

One algorithmic approach is to explicitly address the multiple objectives (recall Section 3.1). We examined this with the genetic algorithm NSGA-II (Deb et al., 2002). Figure 5 shows the results on Year *X* data, compared to the evolution strategy. It can be seen that whereas NSGA-II aggressively optimizes down the number of hard constraint violations, it only slowly reduces the components of the cost. By contrast, the evolution strategy including its constraint cooling process achieves a low number of constraint violations with a much lower cost. We infer that NSGA-II too quickly focuses on one part of the frontier, namely that with low violations, and does not find other interesting points that trade-off slightly more violations for much lower total cost.

We also explored another popular multiobjective algorithm, AMOSA (Bandyopadhyay et al., 2008). AMOSA augmented with constraint cooling performed better than NSGA-II (see Figure 6b, discussed next). However, it was clear that the single-objective methods operating with an aggregated objective function (4) gave much better solutions. In particular, the biobjective aspect of the costs was not important enough to warrant such a decrease in solution quality demonstrated by NSGA-II and AMOSA. The explanation is likely that the overconstrained nature of the problem means the optimization is less about two cost functions and more about a single cost function versus hard constraint violations.

TABLE 6 Greedy versus evolution strategy, Year *X* data

Algorithm	Runtime	Total costs	Violations
ES, best solution	24 h	965.1	10
ES, average	24 h	986.1	11.4
Greedy	15 min	980.5	10

TABLE 7 Greedy versus evolution strategy, Year *Y* data

Algorithm	Runtime	Total costs	Violations
ES, best solution	24 h	1086.7	25
ES, average	24 h	1095.2	27.6
Greedy	20 min	1090.4	16

6.3 | Greedy algorithm

First, the deterministic greedy algorithm has been run on both datasets. The results are shown in Tables 6 and 7 for Year *X* and Year *Y*, respectively, together with the results of the evolution strategy. It can be seen that the greedy algorithm performs very well. Not only does it run in much less time than the evolution strategy, the results are also slightly better than the average result of the evolution strategy. It does not outperform the best result(s) of the evolution strategy, so if plenty of runtime is available to do multiple runs of the evolutions strategy this yields a better solution. However, realistically this will probably not be the case, and the greedy algorithm is suitable to get a decent result in a short time.

6.3.1 | Randomized greedy

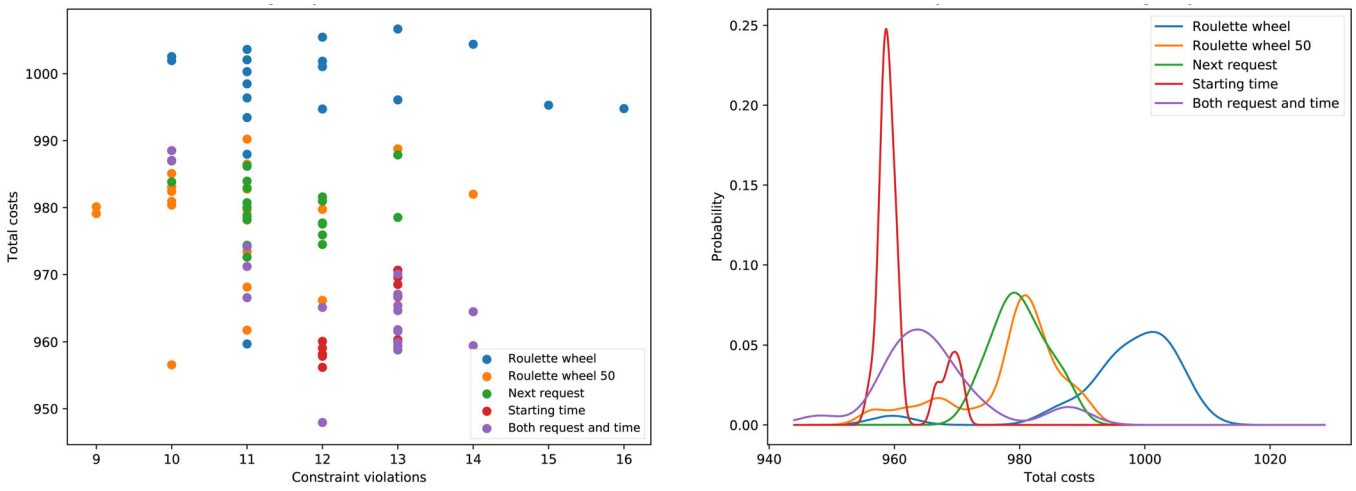
Next to the deterministic greedy algorithm, the different configurations of randomness described in Section 5.2.2 have also been tested, by running each of them for 20 times on the Year *X* data. The results can be found in Figure 7 and Table 8.

It can be seen that the "roulette wheel" method produces relatively poor results. This was expected, as explained in

TABLE 8 Results of various configurations of randomization in the greedy algorithm, Year *X* data

Configuration	Total costs	Violations
Deterministic	980.5	10
Roulette wheel	996.7/959.7	11.9/11
Roulette wheel 50	978.3/956.6	10.9/10
Next request	980.1/972.6	11.4/11
Starting time	960.8/956.2	12.6/12
Both request and time	966.1/947.9	12.4/12

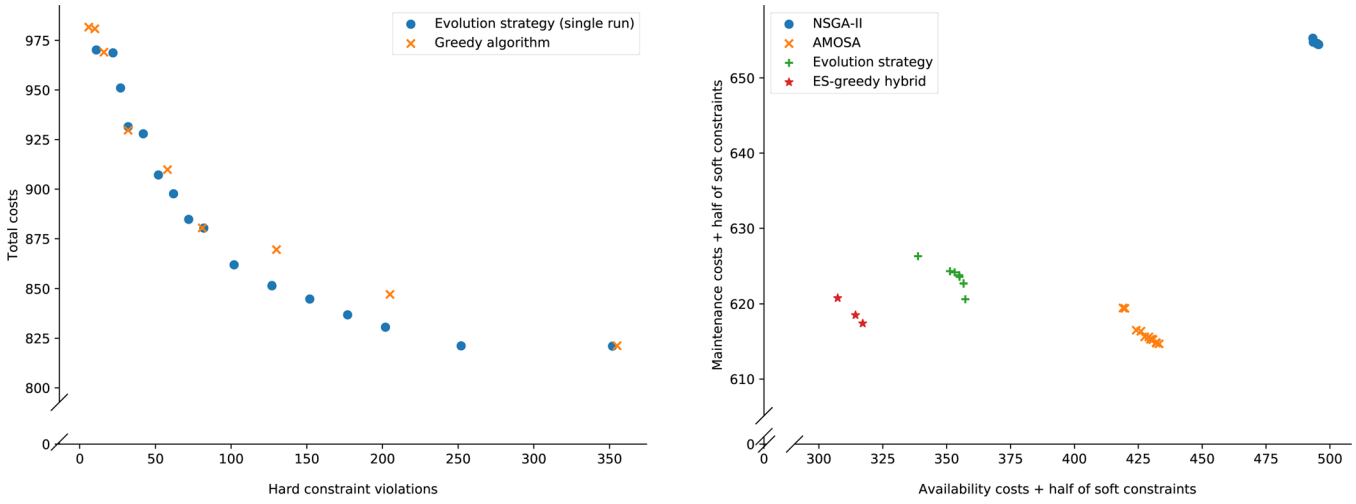
Note: For each of the nondeterministic configurations, the table reports the mean and lowest costs, respectively.



(a) Solutions of various randomization methods for the greedy algorithm

(b) Kernel density estimation, costs only

FIGURE 6 Pareto-frontiers, Year *X* data



(a) Pareto frontier costs vs. violations, ES and greedy, Year X data

(b) Pareto frontier between cost types: ES, hybrid, and two multi-objective GAs. Year X data.

FIGURE 7 Kernel density estimation of total costs for various methods of randomization in the greedy algorithm, Year *X* data

Section 5.2.2. There is essentially too much randomness for the algorithm to remain effective. The configurations for "starting time" and "both request and time" achieve the lowest cost solutions on average. However, these solutions have somewhat more constraint violations on average as well. On the contrary, "roulette wheel 50" and "next request" have slightly higher cost on average, but somewhat fewer constraint violations.

Further, all configurations yield at least one solution that has lower costs than the deterministic version, although also having one or two more hard constraint violations. Hence taking the best of greedy's rapid multiple runs is warranted.

6.4 | Hybrid algorithm

6.4.1 | Year *X* data

Recall that the hybrid algorithm was run seven times on the Year *X* data with the configurations in Table 4. The results are shown in the right two columns of the table.

The first observation is that there are multiple runs of the hybrid algorithm that provide a better solution than both the evolution strategy and (randomized) greedy algorithm, with a substantial difference. The best run of the hybrid has a cost reduction of around 37.5 31 million

TABLE 9 Tested configurations of greedy–ES hybrid, Year *Y*

Run	Stages	Requests per stage	Runtime per stage (h)	Total costs	Violations
1	2	50/638	12/8	1038.4	17
2	2	90/598	12/8	1063.2	16
3	2	20/668	8/12	1040.3	20
4	3	20/70/598	7/7/6	1078.7	17
5	2	50/638	16/4	1091.1	16
6	2	50/638	4/16	1072.9	17

compared to the best result of the evolution strategy runs. The improvement in results validates the idea to combine the strengths of the algorithms.

The second observation is that the Pareto-frontier of the evolution strategy is more encompassing than that of the greedy algorithm, and further that the two together are complementary (Figure 6a). Overall, the hybrid gives a wide spread of interesting trade-offs for the rail manager to consider.

The third observation is that, even when we focus on costs only (ignoring hard constraint violations), the hybrid has a clearly lower total cost than the multiobjective GAs, NSGA-II and AMOSA (Figure 6b).

Examining in detail the difference between the configurations of the hybrid algorithm, we observe some changes affect the solution quite heavily, and some changes that only have a little effect on the solution quality. First, the run that was deemed the "baseline" actually gave the best result. The run with 10 stages instead of 2 has much higher costs than the baseline, and although it is the only run with 8 hard constraint violations instead of 10, the cost increase does not justify this. Redistributing the time so that one stage has much more runtime than the other also causes a large cost increase; this can be seen in the results of runs 6 and 7. Changing the number of requests in the first stage only has little effect on the solution quality, provided this number stays within reasonable bounds. The difference in results between runs 1, 3 and 4 is relatively small and could be caused by variance, when considering the robustness of the evolution strategy shown in Section 6.1.

6.4.2 | Year *Y* data

Finally, the hybrid algorithm has been run on unseen the Year *Y* data with the six configurations specified in Table 9. The main goal is to check whether the conclusions drawn based on the Year *X* data, both in terms of comparison to the other algorithms and the effect of the configuration, still hold. The results are shown in the right columns of the table.

TABLE 10 Overview of results on Year *X* data

Algorithm	Runtime	Costs	Violations
ProRail schedule	6 months	965.2	690
ES, best result	24 h	965.1	10
ES, average	24 h	986.1	11.4
Greedy, deterministic	15 min	980.5	10
Greedy, best result	15 min	947.9	12
Hybrid, best result	24 h	928.1	10

It is clear that the hybrid algorithm gives the best results on the Year *Y* data as well; the best result of the hybrid has approximately 48.4 40 million fewer costs than the best result of the evolution strategy. Looking at the configurations, it can be seen that the "baseline" configuration still performs best as for Year *X*. From runs 5 and 6, it becomes clear that giving one stage much more runtime than the other gives substantially worse results, as for the Year *X* data. In run 4 where three stages were used, the results are also quite a bit worse than the baseline; this was not the case for the Year *X* data. Looking at runs 2 and 3, using 90 requests in stage 1 is worse for the Year *Y* data than for the Year *X* data, and using 20 requests gives similar results to the baseline for both datasets. It should be considered that the Year *Y* data contains fewer requests, so having 90 requests in stage 1 is a relatively higher number for Year *Y* compared to Year *X*. Overall, it looks like having under 10% of the number of requests in stage 1 and the rest in stage 2, with approximately equal runtime, gives the best results. Further research with replication, more configurations and new datasets when they become available is warranted to confirm these conclusions.

6.5 | Summary

Summarizing the results presented in this section, Tables 10 and 11 show the main results of the algorithms developed on the Year *X* and Year *Y* data, respectively. Comparing the algorithms, it is clear that the hybrid algorithm outperforms both the evolution strategy and the greedy algorithm. Only when a short runtime is required,

TABLE 11 Overview of results on Year *Y* data

Algorithm	Runtime	Costs	Violations
ES, best result	24 h	1086.7	25
ES, average	24 h	1095.2	27.6
Greedy, deterministic	15 min	1090.4	16
Hybrid, best result	24 h	1038.4	17

the greedy algorithm could be considered better than the hybrid algorithm. However, the hybrid algorithm can also be set up to run for a shorter time of evolution strategy. Further experimentation is relevant to confirm the most robust configuration of the hybrid algorithm.

7 | CONCLUSION AND OUTLOOK

This article addressed a large-scale, real-world scheduling problem of societal importance: rail maintenance scheduling at a national level. We proposed a hybrid greedy–evolutionary algorithm that robustly provides solutions of better quality than the schedules currently being used in practice. Our approach was successfully demonstrated for the complete rail network of the Netherlands, one of the busiest rail networks in Europe.

Two algorithmic approaches—an evolution strategy and a greedy meta-heuristic—provided similar solutions in terms of total costs, but were strongly differentiated in the distribution of those costs, therefore enabling the hybrid algorithm to outperform them both. All algorithms clearly outperform the manual schedule of Dutch rail infrastructure manager ProRail, with the hybrid algorithm having both lower costs and fewer constraint violations. Further, the speed of scheduling has increased drastically. This means that, in contrast to of maintenance planners such as ProRail only making a single schedule each year, the algorithms allow planners to run multiple iterations, to adjust schedules according to preferences, and, with other stakeholders, to explore different scenarios including which project requests to postpone. Thus we provide important facilitation to the overall sociotechnical planning process.

Several directions are prominent for future work. First, further testing with the hybrid algorithm. Running the algorithm multiple times with more configurations and different input data will give understanding on the effect of the hyperparameter configuration as well as the distribution of solutions for certain configurations. Second, exploring the trade-offs between the components of cost: the two explicit cost objectives and the implicit objective from managerial preferences. Observing the relatively poor performance of multiobjective GAs, a biobjective version of the hybrid is an interesting direction.

ACKNOWLEDGMENTS

The authors thank C. Versteegt, and the anonymous reviewers. This research was partially supported by TAILOR, a project funded by EU Horizon 2020 under grant agreement 952215, and received funding from ProRail.

ORCID

Neil Yorke-Smith https://orcid.org/0000-0002-1814-3515

REFERENCES

Andrade, A. R., & Teixeira, P. F. (2011). Biobjective optimization model for maintenance and renewal decisions related to rail track geometry. *Transportation Research Record*, *2261*(1), 163–170.

Bandyopadhyay, S., Saha, S., Maulik, U., & Deb, K. (2008). A simulated annealing-based multiobjective optimization algorithm: AMOSA. *IEEE Trans. on Evolutionary Computation*, *12*(3), 269–283.

Budai, G., Huisman, D., & Dekker, R. (2006). Scheduling preventive railway maintenance activities. *Journal of the Operational Research Society*, *57*, 1035–1044.

Budai-Balke, G., Dekker, R., & Kaymak, U. (2009). *Genetic and memetic algorithms for scheduling railway maintenance activities* (Econometric Institute Report). Erasmus University Rotterdam.

Cheu, R. L., Wang, Y., & Fwa, T. F. (2004). Genetic algorithm-simulation methodology for pavement maintenance scheduling. *Computer-Aided Civil and Infrastructure Engineering*, *19*(6), 446–455.

Consilvio Alice, Febbraro Angela Di, & Sacco Nicola. (2020). A Rolling-Horizon Approach for Predictive Maintenance Planning to Reduce the Risk of Rail Service Disruptions. *IEEE Transactions on Reliability*, 1–13. https://doi.org/10.1109/tr.2020.3007504

D'Ariano, A., Meng, L., Centulio, G., & Corman, F. (2019). Integrated stochastic optimization approaches for tactical scheduling of trains and railway infrastructure maintenance. *Computers and Industrial Engineering*, *127*, 1315–1335.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. on Evolutionary Computation*, *6*(2), 182–197.

Emmerich, M., Shir, O., & Wang, H. (2018). *Evolution strategies* (pp. 1–31). Cham, Switzerland: Springer.

Froger, A., Gendreau, M., Mendoza, J. E., Pinson, É., & Rousseau, L.-M. (2016). Maintenance scheduling in the electricity industry: A literature review. *European Journal of Operational Research*, *251*(3), 695–706.

Gao, H., & Zhang, X. (2013). A Markov-based road maintenance optimization model considering user costs. *Computer-Aided Civil and Infrastructure Engineering*, *28*(6), 451–464.

Ghofrani, F., Pathak, A., Mohammadi, R., Aref, A. J., & He, Q. (2020). Predicting rail defect frequency: An integrated approach using fatigue modeling and data analytics. *Computer-Aided Civil and Infrastructure Engineering*, *35*(2), 101–115.

Guo, F., Qian, Y., Wu, Y., Leng, Z., & Yu, H. (2021). Automatic railroad track components inspection using real-time instance segmentation. *Computer-Aided Civil and Infrastructure Engineering*, *36*(3), 362–377.

Ingber, L. (1993). Simulated annealing: Practice versus theory. *Mathematical and Computer Modelling*, *18*(11), 29–57.

Khalouli, S., Benmansour, R., & Hanafi, S. (2016). An ant colony algorithm based on opportunities for scheduling the preventive railway maintenance. In *Proceedings of CoDIT 2016*. IEEE.

Kiefer, A., Schilde, M., & Doerner, K. F. (2018). Scheduling of maintenance work of a large-scale tramway network. *European Journal of Operational Research*, *270*(3), 1158–1170.

Lopes Gerum, P. C., Altay, A., & Baykal-Gürsoy, M. (2019). Data-driven predictive maintenance scheduling policies for railways. *Transportation Research Part C*, *107*, 137–154.

Medury, A., & Madanat, S. (2013). Incorporating network considerations into pavement management systems: A case for approximate dynamic programming. *Transportation Research Part C*, *33*, 134–150.

Memarzadeh, M., & Pozzi, M. (2016). Integrated inspection scheduling and maintenance planning for infrastructure systems. *Computer-Aided Civil and Infrastructure Engineering*, *31*(6), 403–415.

Mohammadi, R., He, Q., & Karwan, M. (2020). Data-driven robust strategies for joint optimization of rail renewal and maintenance planning. *Omega*, *103*, 102379.

Peng, F., Kang, S., Li, X., Ouyang, Y., Somani, K., & Acharya, D. (2011). A heuristic approach to the railroad track maintenance scheduling problem. *Computer-Aided Civil and Infrastructure Engineering*, *26*(2), 129–145.

Peng, F., & Ouyang, Y. (2014). Optimal clustering of railroad track maintenance jobs. *Computer-Aided Civil and Infrastructure Engineering*, *29*(4), 235–247.

Peralta, D., Bergmeir, C., Krone, M., Galende, M., Menéndez, M., Sainz-Palmero, G. I., Bertrand, C. M., Klawonn, F., & Benitez, J. M. (2018). Multiobjective optimization for railway maintenance plans. *Journal of Computing in Civil Engineering*, *32*(3), 04018014.

ProRail. (2020). *Jaarverslag 2019*. www.jaarverslagprorail.nl/verslag

Raidl, G., Puchinger, J., & Blum, C. (2010). *Metaheuristic hybrids* (Vol. 146, pp. 469–496). Springer.

Shahmoradi-Moghadam, H., Safaei, N., & Sadjadi, S. J. (2021). Robust maintenance scheduling of aircraft fleet. *IEEE Access*, *9*, 17854–17865.

Simson, S. A., Ferreira, L., & Murray, M. H. (2000). Rail track maintenance planning: An assessment model. *Transportation Research Record*, *1713*, 29–35.

Soh, S. S., Radzi, N. H., & Haron, H. (2012). Review on scheduling techniques of preventive maintenance activities of railway. In *Proceedings of the 4th international conference on computational intelligence, modelling and simulation* (pp. 310–315). IEEE.

Sousa, N., Alçada-Almeida, L., & Coutinho-Rodrigues, J. (2019). Multi-objective model for optimizing railway infrastructure asset renewal. *Engineering Optimization*, *51*(10), 1777–1793.

Su, Z., Jamshidi, A., Núñez, A., Baldi, S., & De Schutter, B. (2019). Integrated condition-based track maintenance planning and crew scheduling of railway networks. *Transportation Research Part C*, *105*, 359–384.

Xie, S., Lei, C., & Ouyang, Y. (2018). A customized hybrid approach to infrastructure maintenance scheduling in railroad networks under variable productivities. *Computer-Aided Civil and Infrastructure Engineering*, *33*(10), 815–832.

Yin, Y., Li, D., Besinovic, N., & Cao, Z. (2019). Hybrid demand-driven and cyclic timetabling considering rolling stock circulation for a bidirectional railway line. *Computer-Aided Civil and Infrastructure Engineering*, *34*(2), 164–187.

Zhang, C., Gao, Y., Yang, L., Gao, Z., & Qi, J. (2020). Joint optimization of train scheduling and maintenance planning in a railway network: A heuristic algorithm using lagrangian relaxation. *Transportation Research Part B*, *134*, 64–92.

Zhang, L., Chang, H., & Xu, R. (2012). Equal-width partitioning roulette wheel selection in genetic algorithm. In *Proceedings of the conference on technologies and applications of artificial intelligence (TAAI'12)* (pp. 62–67). IEEE.

Zhang, T., Andrews, J., & Wang, R. (2012). Optimal scheduling of track maintenance on a railway network. *Quality and Reliability Engineering International*, *29*(2), 285–297.

How to cite this article: Oudshoorn M, Koppenberg T, Yorke-Smith N. Optimization of annual planned rail maintenance. *Comput Aided Civ Inf*. 2021;1–19. https://doi.org/10.1111/mice.12764