**Algorithmic Operations Research**

# Recoverable Robustness for Train Shunting Problems

Serafino Cicerone, Gianlorenzo D'Angelo, Gabriele Di Stefano, Daniele Frigioni et Alfredo Navarra

Volume 4, numéro 2, fall 2009

URI : https://id.erudit.org/iderudit/aor4_2art03

Aller au sommaire du numéro

Éditeur(s)

Preeminent Academic Facets Inc.

ISSN

1718-3235 (numérique)

Découvrir la revue

Citer cet article

Cicerone, S., D'Angelo, G., Di Stefano, G., Frigioni, D. & Navarra, A. (2009). Recoverable Robustness for Train Shunting Problems. *Algorithmic Operations Research*, *4*(2), 102–116.

Résumé de l'article

*Several attempts have been done in the literature in the last years in order to provide a formal definition of the notions of* robustness *and* recoverability *for optimization problems. Recently, a new model of* recoverable robustness *has been introduced in the context of railways optimization. The basic idea of recoverable robustness is to compute solutions that are robust against a limited set of disturbances and for a limited recovery capabilities. The quality of the robust solution is measured by its* price of robustness *that determines the trade-off between an optimal and a robust solution.*

*In this paper, within the recoverable robustness model, we emphasize algorithmic aspects and provide definitions of* robust algorithm *and* price of robustness *of a robust algorithm as a measure to evaluate its performance. A robust algorithm provides a solution that maintains feasibility by possibly applying available* recovery capabilities *in the case of changes to the input data. We study various settings in the context of* shunting problems*, i.e. the reordering of train cars over a hump yard. The considered shunting problems can be seen as the reordering of an integer vector by means of a set of available stacks with the further constraint that the pull operation does not involve only the element on top of a stack, but all the elements contained in the stack.*

*We provide efficient robust algorithms concerning specific shunting problems. In particular, we study algorithms able to cope with* disturbances*, as temporary and local unavailability and/or malfunctioning of key resources that can occur and affect planned operations. Various scenarios are considered, and robustness results are presented.*

*Algorithmic Operations Research Vol.4 (2009) 102–116*

# Recoverable Robustness for Train Shunting Problems

Serafino Cicerone, Gianlorenzo D'Angelo, Gabriele Di Stefano, Daniele Frigioni

Department of Electrical and Information Engineering, University of L'Aquila, Monteluco di Roio, 67040 L'Aquila, Italy

Alfredo Navarra

Department of Mathematics and Computer Science, University of Perugia, Via Vanvitelli 1, 06123 Perugia, Italy

**Abstract**

*Several attempts have been done in the literature in the last years in order to provide a formal definition of the notions of* robustness *and* recoverability *for optimization problems. Recently, a new model of* recoverable robustness *has been introduced in the context of railways optimization. The basic idea of recoverable robustness is to compute solutions that are robust against a limited set of disturbances and for a limited recovery capabilities. The quality of the robust solution is measured by its* price of robustness *that determines the trade-off between an optimal and a robust solution.*

*In this paper, within the recoverable robustness model, we emphasize algorithmic aspects and provide definitions of* robust algorithm *and* price of robustness *of a robust algorithm as a measure to evaluate its performance. A robust algorithm provides a solution that maintains feasibility by possibly applying available* recovery capabilities *in the case of changes to the input data. We study various settings in the context of* shunting problems*, i.e. the reordering of train cars over a hump yard. The considered shunting problems can be seen as the reordering of an integer vector by means of a set of available stacks with the further constraint that the pull operation does not involve only the element on top of a stack, but all the elements contained in the stack.*

*We provide efficient robust algorithms concerning specific shunting problems. In particular, we study algorithms able to cope with* disturbances*, as temporary and local unavailability and/or malfunctioning of key resources that can occur and affect planned operations. Various scenarios are considered, and robustness results are presented.*

*Key words:* robustness, disturbance, recoverability, robust algorithm, shunting, hump yard

## 1. Introduction

In practical optimization problems information is often imperfect. Input data may be subject to changes or may be completely unknown in advance. In contrast, the standard optimization theory assumes that input data are fully known in advance. Hence, it is important to develop methods that consider the unavoidable imperfection of the input data for the construction of a solution and for its prompt recovery in the case of disturbances. In this sense, what is needed is a notion of *recoverable robust solution* for an optimization problem, that is a solution that maintains feasibility by possibly applying available recovery capabilities in the case of changes to the input data.

Several attempts have been done in the literature in the last years in order to provide formal definitions of the notions of *robustness* and *recoverability* for optimization problems. For example, in the area of railway optimization, the concepts of robustness and recoverability have been studied in terms of the classical robust optimization (see, e.g., [3–5,18]) and online algorithms (see [22] for an interesting survey), respectively. In particular, a robust plan is an *a priori* plan that maintains feasibility and as much as possible of the quality of an optimal solution in the case of imperfect information. An online plan is a *real-time* plan that has to be developed when unpredictable disturbances in daily operations occur, and before the entire sequence of distur-

⋆ This work was partially supported by the Future and Emerging Technologies Unit of EC (IST priority - 6th FP), under contract no. FP6-021235-2 (project ARRIVAL). An extended abstract of this paper appeared in [9].

*Email:* Serafino Cicerone, [serafino.cicerone@univaq.it], Gianlorenzo D'Angelo, [gianlorenzo.dangelo@univaq.it], Gabriele Di Stefano, [gabriele.distefano@univaq.it], Daniele Frigioni [daniele.frigioni@univaq.it], Alfredo Navarra [navarra@dmi.unipg.it].

bances is known. The goal is to react fast, while retaining as much as possible of the quality of an optimal solution, that is, a solution that would have been achieved if the entire sequence of disturbances was known in advance.

Despite this increasing interest, a final answer to the question "what are *robustness* and *recoverability* for an optimization problem?" has not yet been given. In fact, the notion of robustness in every day life is much broader than that pursued in the area of robust optimization so far. In the most restricted sense, a robust plan stays unchanged in every likely scenario. The basic idea of robustness is given by a problem and some knowledge imperfection which one has to cope with. That is, the solution provided for a given instance of the problem must hold even though some changes in such an instance occur. This kind of robustness is not always suitable if some recovery strategies are not introduced. Moreover, in many practical applications, there might be the possibility to intervene before some scheduled operations are being performed. This suggests to study robustness and recoverability in a unified way.

Usually, modifications that may occur are restricted to some specified subset of all possible ones. It is reasonable to require that, if a disturbance occurs, then one would like to maintain as much as possible a precomputed solution taking into account some "soft" recovery strategies. Recovering should be simple and fast. Moreover, there are cases where recoverability is necessary in order to still have some useful solution for a problem. A solution that undergoes slight changes is called robust even though it could require the use of some recovery capabilities.

A first tentative of unifying the notions of *robustness* and *recoverability* into a new integrated notion of *recoverable robustness* has been done in [26] in the context of railways optimization. This new notion describes robustness with respect to (limited) recovery possibilities. It integrates robustness and recoverability as the solutions are required to be recoverable. The basic idea of recoverable robustness is to compute solutions that are robust against a limited set of scenarios and for a limited recovery. The quality of the robust solution is measured by its *price of robustness* that determines the trade-off between an optimal and a robust solution.

In this paper, based on the directions given in [26], we emphasize algorithmic aspects and provide a definition of *robust algorithm* and a definition for the corresponding *price of robustness*. The purpose is to capture useful properties that help to overcome the standard notion of robustness. Given an optimization problem $P$, a modification function $M$, and a set of available recovery strategies $\mathcal{A}$, the corresponding recoverable robustness problem $\mathcal{P}$ is a triple $(P, M, \mathcal{A})$. Given an instance $i$ of $P$, $M(i)$ is the set of the instances obtained by applying any possible disturbance to $i$. A *robust algorithm* $A_{rob}$ takes $i$ as input and outputs a feasible solution for $i$ that can be recovered in case of disturbance by using recovery capabilities in $\mathcal{A}$. In other words, given an instance $i$ of $P$ and a disturbance $j \in M(i)$, algorithm $A_{rob}$ provides a solution $s$ for $i$ that can be turned into a feasible solution for $j$ by applying some recovery strategies allowed by $\mathcal{A}$. Solution $s$ is then called a robust solution. Clearly, robust solutions provided by $A_{rob}$ can be far from the optimum. Such a distance is measured by the notion of price of robustness. In [26], the aim is to provide the best robust solution, i.e., the one that minimizes the price of robustness.

In this paper we are interested in finding efficient robust algorithms, and evaluating them by comparing the corresponding prices of robustness. In particular, we are interested in finding robust algorithms for some *shunting problems* (see, e.g. [12,23–25]), that is the scheduling of activities at a shunting yard in depots or stations. In railroad shunting yards, incoming freight trains are split up and re-arranged according to their destinations. In stations and train depots, passenger trains are parked overnight or during low traffic hours. In either case, we are given an ordering of arriving units, i.e., either cars or trains, and we have to decide how to use the tracks of the shunting yard to reorder the units according to a required departure sequence. Possible scheduling activities are limited by the fixed number of available tracks, by their length and by the way tracks may be approached. From a more general point of view, the considered shunting problems can be seen as the reordering of an integer vector by means of a set of available stacks with the further constraint that the pull operation does not involve only the element on top of a stack, but all the elements contained in the stack.

In the literature, shunting problems have been studied along two main directions. The first assumes a perfect knowledge of the incoming and outgoing sequences of units, as in the standard optimization theory, and many results have been achieved (see, e.g., [6,11,12,15,16,19,23–25]). The second looks at the shunting problem as an online problem: since the trains could accumulate lateness before arriving at the depot, the time of arrival of each train could be unpredictable. The tracks must thus be assigned online, as

the trains arrive, on the basis of departure times and previous assignments [13,14,31].

These two approaches lack in reality, since *disturbances*, concerning temporary and local unavailability and/or malfunctioning of key resources, can occur (e.g., different orders of the incoming trains/cars, new trains/cars, missing trains/cars, or faulty infrastructures like tracks). These disturbances can affect for example the planned incoming unit sequence, but it is also unlikely that we have no idea about the order of the sequence, as in the online approach. In this paper, we provide robust algorithms for the shunting over a hump yard problem which are able to cope with a limited number of disturbances, and study their price of robustness. We also study various levels of robustness according to different recovery capabilities.

### 1.1. *Related Work*

Robustness problems have been addressed in many fields of research. However, a concrete model that unifies the provided approaches is still missing. In our context, when no recovery capabilities are allowed, we talk about *strict robustness*. Fault tolerance problems fall into this setting. In fact, when studying a problem from the fault tolerance prospective, one has to cope with some possible (limited) disruptions that may occur after planning a feasible solution. That is, the solution must keep its feasibility even though the input instance may change after applying the provided solution. An example of such an approach can be found in networking problems. In [21], the problem of finding a bi-connected spanning graph of the input network is studied in order to maintain connectivity in the case an edge becomes unavailable. A similar problem is studied in [7] but for wireless networks. That is, in order to achieve the bi-connectivity or the most general $k$-connectivity, one has to opportunely tune the transmission ranges of the nodes while minimizing the power consumption. Classical networking problems like $k$-shortest paths [17,30], the most vital node [28,32] and the most vital edge [2,27] of a network also fall into the set of problems which cope with possible and limited disruptions. Apart for the robust optimization approaches previously discussed [3–5,18], other problems related to strict robustness can also be found in distributed storage and file sharing systems (see for instance [1,8]). First approaches that allow the possibility of applying also limited recovery capabilities once a disruption occurs can be found again in networking problems, like for instance in [20]. In such a paper, in fact, the idea was to pre-compute a useful graph which might be used in order to replace a possible faulty edge in order to recover the required connectivity. More classical techniques like parity check bits or RAID (Redundant Array of Independent Disks [29]) systems used for storage purposes can be also considered as examples of recoverable robust systems. In fact, they make use of redundance in order to keep safe the storage of data subject to limited malfunctions, such as the loss of one bit, or the damage of an hard disk. In such settings, once a disruption occurs, a recovery algorithm can re-build the missing information by making use of the redundant data preventively stored.

### 1.2. *Outline*

The paper is organized as follows: in Section 2., we describe our modification of the model concerning robustness for optimization problems given in [26] and introduce our notions of *robust algorithm* and *price of robustness* of a robust algorithm; in Section 3., we describe the shunting over a hump yard problem as given in [24,25]; in Section 4., we give a robust model to shunting problems arising in practical contexts, and for each problem, we provide robust algorithms and evaluate their price of robustness; finally, in Section 5., we give some conclusive remarks and discuss some open problems.

## 2. Recoverable Robustness

In this section, motivated by algorithmic issues, we describe a slightly different formulation of the notion of recoverable robustness proposed in [26]. In particular, given an optimization problem $P$, we first show how to turn $P$ into a *recoverable robustness problem* $\mathcal{P}$. Then, we formally define the notion of *robust solutions* for $P$, that is, the feasible solutions for $P$ that also solve $\mathcal{P}$. Finally, we define the concept of *robust algorithm* for $\mathcal{P}$, that is an algorithm that computes the robust solutions for $P$, and then quantify its *price of robustness*. By using the theoretically best robust algorithm for $\mathcal{P}$, we define the price of robustness of problem $\mathcal{P}$. In conclusion, we define the notions of *exact* and *optimal* robust algorithms.

In the remainder, an optimization problem $P$ is characterized by the following parameters.

- $I$, the set of instances of $P$;

- $F$, a function that associates to any instance $i \in I$ the set of all feasible solutions for $i$;
- $f$: $S \to \mathbb{R}^+$, the objective function of $P$, where $S = \bigcup_{i \in I} F(i)$ is the set of all feasible solutions for $P$.

Note that, for several optimization problems the objective function is defined to have values in $\mathbb{R}$. However, it is possible to turn any such problem into an equivalent one which satisfies our definition. Without loss of generality, from now on we consider minimization problems. Additional concepts to introduce robustness requirements for a minimization problem $P$ are needed:

- $M : I \to 2^I$ – a *modification* function for instances of $P$. This function models the following case. Let $i \in I$ be the considered input to the problem $P$, and let $s \in S$ be the planned solution for $i$. A *disturbance* is meant as a modification to the input $i$, and such a modification can be seen as a new input $j \in I$. Typically, the modification $j$ depends on the current input $i$, and this fact is modeled by the constraint $j \in M(i)$. Hence, given $i \in I$, $M(i)$ represents the set of instances of $P$ that can be obtained by applying all possible modifications to $i$. Of course, when a disturbance $j \in M(i)$ occurs, a new solution $s' \in F(j)$ has to be recomputed for $P$.
- $\mathcal{A}$ – a class of *recovery algorithms* for $P$. Algorithms in $\mathcal{A}$ represent the capability of recovering against disturbances. Since in a real-world problem the capability of recovering is limited in some way, the class $\mathcal{A}$ can be defined in terms of some kind of *restrictions*, such as feasibility or algorithmic restrictions. An element $A_{rec} \in \mathcal{A}$ works as follows: given $(i, s) \in I \times S$, an instance/solution pair for $P$, and $j \in M(i)$, a modification of the current instance $i$, then $A_{rec}(i, s, j) = s'$, where $s' \in S$ represents the recovered solution for $P$. In what follows we provide two examples for the definition of $\mathcal{A}$.

(1) Define $\mathcal{A}$ by imposing a constraint on the solutions provided by the recovery algorithms. In particular, the new (recovered) solutions computed by an algorithm must not deviate too far from the original solution $s$, according to a distance measure $d$. Formally: given a real number $\Delta \in \mathbb{R}$ and a distance function $d : S \times S \to \mathbb{R}$, each element $A_{rec}$ in such a class fulfills the following constraint:

$$\forall i \in I, \forall s \in S, \forall j \in M(i),\ d(s, A_{rec}(i, s, j)) \leq \Delta.$$

(2) Define $\mathcal{A}$ by bounding the computational power of recovery algorithms. Formally: given a function $f : I \times S \times I \to \mathbb{N}$, each element $A_{rec}$ in such a class fulfills the following constraint:

$\forall i \in I, \forall s \in S, \forall j \in M(i),\ A_{rec}(i, s, j)$ must be computed in $O(f(i, s, j))$ time.

We can now formally define how $P$ can be transformed into a recoverable robustness problem.

**Definition 1** *A* recoverable robustness problem $\mathcal{P}$ *is defined by the triple* $(P, M, \mathcal{A})$. *All the recoverable robustness problems form the class* RRP.

**Definition 2** *Let* $\mathcal{P} = (P, M, \mathcal{A}) \in$ RRP. *Given an instance* $i \in I$ *for* $P$, *an element* $s \in F(i)$ *is a* feasible solution *for* $i$ *with respect to* $\mathcal{P}$ *if and only if the following relationship holds:*

$$\exists A_{rec} \in \mathcal{A} : \forall j \in M(i),\ A_{rec}(i, s, j) \in F(j).$$

In other words, $s \in F(i)$ is feasible for $i$ with respect to $\mathcal{P}$ if it can be *recovered* by applying an algorithm $A_{rec} \in \mathcal{A}$ for each possible disturbance $j \in M(i)$. We use the notation $F_{\mathcal{P}}(i)$ to represent all the feasible solutions for $i$ with respect to $\mathcal{P}$. Formally $F_{\mathcal{P}}(i)$ is defined as:

$$F_{\mathcal{P}}(i) = \{s \in F(i) : s \text{ is a feasible solution for } i \text{ with respect to } \mathcal{P}\}.$$

A possible scenario for this situation is depicted in Figure 1. In the remainder, solutions in $F_{\mathcal{P}}(i)$ are also called *robust solutions* for $i$ with respect to the original problem $P$.

**Definition 3** *Let* $\mathcal{P} = (P, M, \mathcal{A}) \in$ RRP. *A* robust algorithm *for* $\mathcal{P}$ *is any algorithm* $A_{rob}$ *such that, for each* $i \in I$, $A_{rob}(i)$ *is a robust solution for* $i$ *with respect to* $\mathcal{P}$.

It is worth to mention that, if $\mathcal{A}$ is the class of algorithms that do not change the solution $s$, that is, if each algorithm $A_{rec} \in \mathcal{A}$ fulfills the following condition

$$\forall (i, s) \in I \times S, \forall j \in M(i),\ A_{rec}(i, s, j) = s,$$

then the robustness problem $\mathcal{P} = (P, M, \mathcal{A})$ represents the so called *strict robustness problem*. Note that, in this case, a robust algorithm $A_{rob}$ for $\mathcal{P}$ must provide a solution $s$ for $i$ such that, for each possible modification $j \in M(i)$, $s \in F(j)$. This means that, since $A_{rec}$ has no capability of recovering against possible disturbances, then $A_{rob}$ has to find solutions that "absorb" *any* possible disturbance.

Let us consider Figure 1 again. Note that, if $\bar{s}$ denotes the optimal solution for $P$ when the input instance is

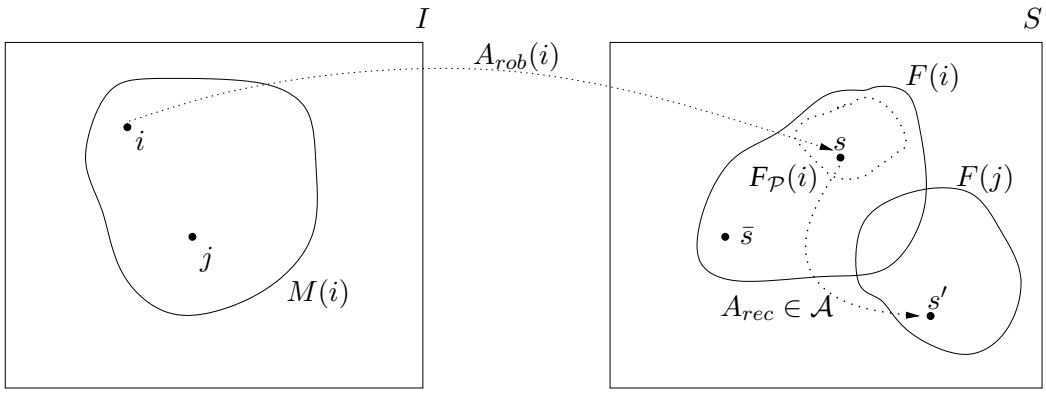


Fig. 1. A scenario for recoverable robustness problem: $I$, set of instances; $S$, set of solutions; $M(i)$, set of instances obtainable after a small modification; $F(i)$ and $F(j)$, set of feasible solutions for $i$ and $j$ respectively; $F_{\mathcal{P}}(i)$, set of recoverable solutions for $i$; $\bar{s}$, optimal non-robust solution for $i$; $s$, robust solution obtained by $A_{rob}$; $s'$, recovered solution obtained by an algorithm $A_{rec} \in \mathcal{A}$ after disturbance $j \in M(i)$.

$i$, it is possible that $\bar{s}$ is not in $F_{\mathcal{P}}(i)$; this implies that every robust solution for $i$ may be "very far" from $\bar{s}$. A "good" robust algorithm should find the best solution in $F_{\mathcal{P}}(i)$ for $P$, for each possible input $i \in I$. The following definition introduces the concepts of *price of robustness* of both a robust algorithm and a recoverable robustness problem.

**Definition 4** *Let* $\mathcal{P} \in$ RRP. *The* price of robustness *of a robust algorithm* $A_{rob}$ *for* $\mathcal{P}$ *is given by*

$$\text{POR}(\mathcal{P}, A_{rob}) = \max_{i \in I} \left\{ \frac{f(A_{rob}(i))}{\min\{f(x) : x \in F(i)\}} \right\}.$$

**Definition 5** *Let* $\mathcal{P} \in$ RRP. *The* price of robustness *of* $\mathcal{P}$ *is given by*

$$\text{POR}(\mathcal{P}) = \min\{\text{POR}(\mathcal{P}, A_{rob}) : A_{rob} \textit{ is a robust algorithm for } \mathcal{P}\}.$$

**Definition 6** *Let* $\mathcal{P} \in$ RRP *and let* $A_{rob}$ *be a robust algorithm for* $\mathcal{P}$. *Then,*

- $A_{rob}$ *is* exact *if* $\text{POR}(\mathcal{P}, A_{rob}) = 1$*;*
- $A_{rob}$ *is* $\mathcal{P}$-optimal *if* $\text{POR}(\mathcal{P}, A_{rob}) = \text{POR}(\mathcal{P})$.

Notice that, the definition of exact robust algorithm always refers to the problem $\mathcal{P}$.

## 3. Shunting Over a Hump Yard

In this section, we report the shunting over a hump yard problem as given in [23–25]. The problem is specified by an input train $T_{in}$ composed of $n$ cars and an output train $T_{out}$ given by a permutation of $T_{in}$ cars.

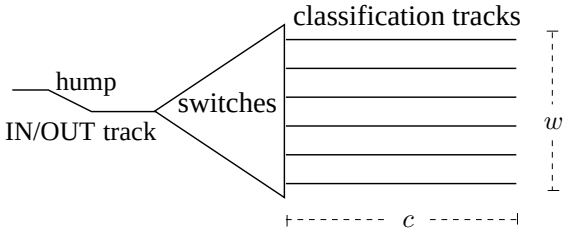


Fig. 2. Hump yard infrastructure composed of $w$ classification tracks, each of size $c$.

Each car is assigned with a unique label. The considered hump yard appears as in Figure 2. The hump yard is made of an input track where trains arrive, and a set of switches by which cars composing the incoming train can be shunted over the available classification tracks. A classification track is approached from a single side and works like a stack. The set of classification tracks is denoted by $W$, the size of $W$ is denoted by $w$, and the size of each track, i.e., the number of cars that can fit into a classification track, by $c$. Therefore, an instance of the problem is given by a quadruple $(T_{in}, T_{out}, W, c)$.

The hump yard supports a sorting operation by repeatedly doing the so called *track pull* operation which is made up of the following steps:

- Connect the cars of one classification track into a train, called *pseudotrain*;
- Pull the pseudotrain over the hump;
- Disconnect the cars in the pseudotrain;
- Push the pseudotrain slowly over the hump, yielding single cars that run down the hill from the hump towards the classification tracks;

- Control the switches such that every single car goes to a specified track.

The goal is to reorder $T_{in}$ according to $T_{out}$ by repeatedly performing the track pull operation (an example of reordering by means of track pulls can be seen in Figure 3). The cost of the reordering is measured by the number of track pulls. Notice that, at least one pull must be performed as the hump yard is used only when one has to reorder or to park a train.

As in [24], we consider three different variants of the shunting over a hump yard problem by specifying constraints for parameters $c$ and $w$. Namely,
$Sh_1$**:** $c$ bounded, $w$ unbounded;
$Sh_2$**:** $c$ unbounded, $w$ bounded;
$Sh_3$**:** $c$ and $w$ unbounded.

When convenient, we refer to $Sh$ as any of the above problems, indiscriminately, that is, when a result holds for $Sh$, then it holds for all problems $Sh_1$, $Sh_2$ and $Sh_3$.

In [24], a polynomial algorithm for each of the above problems is given. In particular, a 2-approximation algorithm for $Sh_1$, and optimal algorithms for $Sh_2$ and $Sh_3$, are provided. As in [24], we do not consider the case of $c$ and $w$ bounded.

In what follows we describe the notation used in [24,25] to represent a *shunting plan*. A shunting plan specifies (i) a sequence $S$ of $h$ track pull operations given by the tracks whose cars are pulled, and (ii) for every pulled car which track it is sent to. Note that, if one track is pulled several times, then it appears in $S$ more than once. Of course, if there is no limit on the number of tracks ($w \geq h$), then there is no need to reuse a track. Given $S$, the itinerary of a car can be described by the sequence of tracks it visits. For the task at hand, it is convenient to specify this sequence as a bit-string or code $b_1 \cdots b_h$ where the different bits stand for the pulled tracks, and there is a 1 if and only if the car visits that track. Now, if track $i$ is pulled, then the new destination of a car is given by the position of its next 1 in its code, i.e., the lowest index $j > i$ such that $b_j = 1$. A shunting plan must specify a track pull sequence $S$ and it has to associate a code to each car. The length of each code is determined by the length of $S$ and cars may share the same code.

An example is shown in Figure 3. The sequence of track pulls is given by $S = \{1, 2, 3, 4, 5\}$ from right to left among classification tracks. In the example $c = 3$ and the number of track pulls is set to 5. The set of codes of length 5 provided by a feasible solution satisfies the property that at each position at most three codes have the corresponding bit set to 1. This implements the constraint on $c$ and implies that at most eleven different codes can be generated. Cars from 11 down to 1 are associated with codes 00000, 00001, 00010, 00011, 00100, 00110, 01000, 01100, 10000, 10001, 11000, respectively. Figure 3 shows the sequence of configurations obtained after each track pull and reorder of the pulled cars according to their codes. The algorithm used in the example has been proposed in [24]. From now on, we refer to such an algorithm as $A_{out}$. It computes a shunting plan when $c$ is bounded and the input train is unknown in advance. In particular, $A_{out}$ provides $n$ different codes, one for each car in $T_{in}$. Each code specifies the route that the corresponding car has to perform among the shunting yard in order to be placed in the desired position according to $T_{out}$. In [24] it has been shown that $A_{out}$ is optimal with respect to the minimum number of track pulls. For the sake of simplicity, it is assumed that $T_{out}$ is composed on a track not used for shunting operations but that can contain the whole train.

Note that, when $T_{in}$ is known in advance, two cars might be assigned with the same code. This would imply that they will have the same relative order in $T_{out}$ as in $T_{in}$. Two cars that are consecutive in $T_{out}$ can get the same code if they are in the correct order in $T_{in}$. A maximal set of cars in $T_{out}$ that has this property is called a *chain*.

**Definition 7** *In a shunting plan, for each code $x$, a* pure chain *is the set of all cars associated with $x$.*

In practice, the number of chains in $T_{in}$ along with the hump yard structure represents the key quantity with respect to the number of track pulls that must be performed by a shunting plan in order to obtain the desired $T_{out}$. For this reason, in the remainder of the paper, we use the following further notation: $opt(k, c, w) \geq 1$ is the number of track pulls needed by an optimal shunting plan in order to manage $k$ cars/chains over a hump yard made of $w$ tracks, each of size $c$ (for $Sh_1$ and $Sh_3$, $w = \infty$, while for $Sh_2$ and $Sh_3$, $c = \infty$); $apx(k, c, w)$ is the best known approximation algorithm for the corresponding shunting problem, and $apxr$ is its approximation ratio. When it is clear by the context we skip parameters equal to $\infty$ from the previous notation. Furthermore, for every instance $i = (T_{in}, T_{out}, W, c)$ we denote by $r_i$ and $n_i$ the number of chains and cars in $T_{in}$, respectively.

## 4. Disturbances and Recoverability

In this section, we provide robust algorithms for the shunting problems described in Section 3. and evaluate

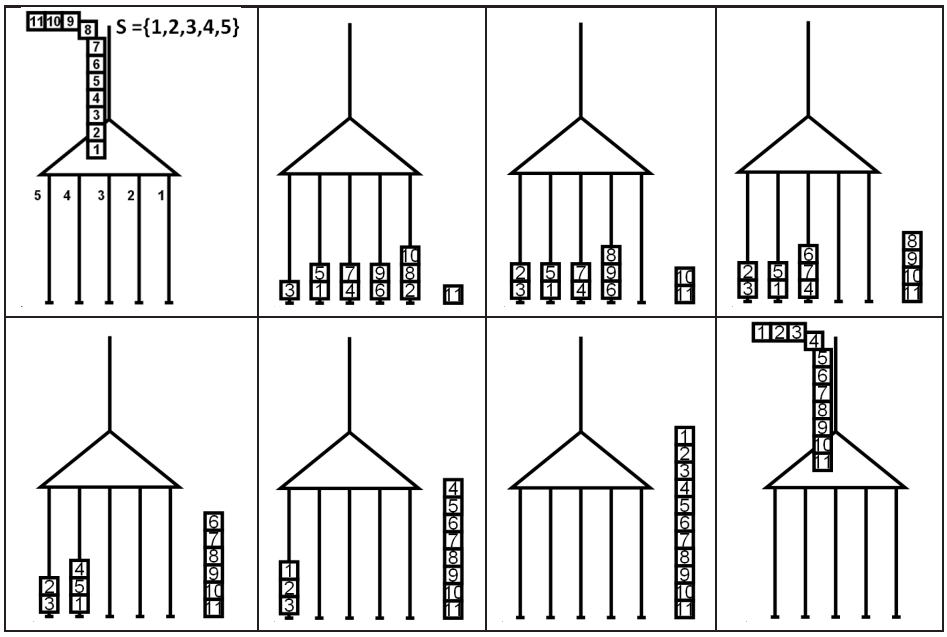


Fig. 3. Example of a shunting plan when $c = 3$ and the number of track pulls is set to 5. Cars from 11 down to 1 are associated with codes 00000, 00001, 00010, 00011, 00100, 00110, 01000, 01100, 10000, 10001, 11000 respectively. $T_{out}$ is composed outside the hump yard and the corresponding track is not shown.

their price of robustness as defined in Section 2.. For example, $Sh_1$ is defined by

- $I$: set of quadruples $(T_{in}, T_{out}, W, c)$ where train $T_{in}$ is defined as a sequence of cars and train $T_{out}$ is a permutation of $T_{in}$;
- $F(i)$: set of all feasible solutions for a given instance $i \equiv (T_{in}, T_{out}, W, c) \in I$, i.e. any sequence of track pulls combined with a set of codes (one per car) that transform $T_{in}$ in $T_{out}$ when $c$ is bounded;
- $f$: number of track pulls.

Regarding the modification function $M$, we consider four different possibilities:

$M_1$**:** it models the case in which one car arrives in an unexpected incoming position;

$M_2$**:** it models the case in which the incoming train contains one additional unexpected car;

$M_3$**:** it models the case in which the incoming train contains one car less than expected;

$M_4$**:** it models the case in which one of the classification tracks composing the hump yard may fault.

Concerning classes of recovery algorithms, we consider the following three possibilities:

$\mathcal{A}_1$**:** $\forall A \in \mathcal{A}_1$, $\forall (i,s) \in I \times S$, $\forall j \in M(i)$, $A(i,s,j) = s$, i.e., there are no recovery strategies to apply (strict robustness);

$\mathcal{A}_2$**:** $\forall A \in \mathcal{A}_2$, $\forall (i,s) \in I \times S$, $\forall j \in M(i)$, $A(i,s,j) = s'$, where $s'$ may differ from $s$ by at most one code without affecting the track pull sequence, i.e., at most one pure chain may be assigned with a new code of the same length;

$\mathcal{A}_3$**:** $\forall A \in \mathcal{A}_3$, $\forall (i,s) \in I \times S$, $\forall j \in M(i)$, $A(i,s,j) = s'$, where $s'$ may differ from $s$ by all the set of codes without affecting the track pull sequence, i.e., every pure chain may be assigned with a new code of the same length.

Note that each of the three defined classes of recovery algorithms do not affect the scheduled track pulls sequence defined by a robust shunting algorithm. This is motivated by the fact that modifying the track pulls sequence is expensive as it requires to change the switches setting or increase the number of track pulls. Recovery capabilities, instead, should be cheap operations since they cannot be planned a priori but are used during the operational phase.

For each shunting problem and for each modification function, the three different classes of recovery algorithms imply three different recoverable robustness problems $\mathcal{P}$. However, by definition, every upper bound to the price of robustness of each shunting problem with $\mathcal{A}_1$ holds for the same problem with $\mathcal{A}_2$ as well as every upper bound obtained with $\mathcal{A}_2$ holds for $\mathcal{A}_3$. Moreover, every lower bound obtained with $\mathcal{A}_3$ holds for $\mathcal{A}_2$ as

well as every lower bound obtained with $\mathcal{A}_2$ holds for $\mathcal{A}_1$.

Each Section from 4.1. to 4.4. copes with a modification function from $M_1$ to $M_4$, respectively, and analyzes the price of robustness of all possible robustness problems arising from the combination with the shunting problems and the classes of recovery algorithms.

### 4.1. *One Car With Unexpected Incoming Position*

Given an instance $i = (T_{in}, T_{out}, W, c)$ of a shunting optimization problem $Sh$, let $M_1(i)$ represent all possible instances $(T'_{in}, T_{out}, W, c)$ obtainable from $i$ by changing the position of just one car in $T_{in}$. For each of the three problems of Section 3. we study feasibility of robust shunting plans for the three different classes of recovery algorithms defined above.

The following lemma describes which practical situation a robust plan must be able to absorb/recover with respect to a car incoming at an unexpected position.

**Lemma 1** *Let $v$ be a car arriving at the hump yard in a different position than expected. At most one additional pure chain must be managed with respect to the expected case.*

**Proof.** If $v$ composed a pure chain itself, then every shunting plan is robust since the same code assigned to $v$ is valid also in the actual case. The same holds in all cases where the change in the incoming position of $v$ does not affect its relative position with respect to the pure chain it belongs to, or $v$ can be joint with some other pure chains. In any other case, one pure chain is created. In the remainder of the proof, we address these cases.

If $v$ was the first (last, resp.) car of its original chain, and it arrives after (before, resp.) some cars of that chain then it becomes a pure chain itself (see Figure 4). All the other cars of its original pure chain still compose a pure chain since their relative order do not change.

If $v$ was part (in the middle) of a pure chain then $v$ may arrive either before its original pure chain (*case a*), or in the middle but before its expected placement (*case b*), or in the middle but after its expected placement (*case c*), or after its original pure chain (*case d*), see Figure 5. If *case a* occurs, then $v$ with all cars of the original pure chain after the expected position of $v$ still compose a chain but the remaining part of the original pure chain cannot be assigned with the same code. If *case b* occurs, then the same arguments of *case a* can be applied. If *case c* occurs then the first part of the original pure chain until the expected position of $v$ plus $v$ compose a pure chain, while the remaining cars must be another pure chain. If *case d* occurs, then same arguments of *case c* still hold.

Summarizing, in all cases at most one additional pure chain is created. □

In a shunting plan, Lemma 1 is reflected in the need of at most one additional code.

**Lemma 2** *Let $\mathcal{P} = (Sh, M_1, \mathcal{A}_1)$. For every input train $T_{in}$, any robust shunting algorithm $A_{rob}$ must provide a unique code to each car of $T_{in}$.*

**Proof.** Let us assume that $A_{rob}$ is robust with respect to any possible change of one car position. Furthermore, let us assume by contradiction that $A_{rob}$ assigns the same code to two cars $v$ and $w$. Without loss of generality, let $v$ being expected before $w$ in $T_{in}$. This means that $v$ should appear before $w$ also in the outgoing train. Let us consider the disturbance where $w$ precedes $v$ in $T_{in}$. Since $A_{rob}$ associates the same code to $v$ and $w$, then $w$ will appear before $v$ also in the outgoing train. This contradicts the hypothesis that $A_{rob}$ is a robust shunting algorithm with respect to any change in the position of one car. □

Let us consider problem $Sh_1$. As mentioned in Section 3., two solutions have been proposed in [24] for this case. The first solution provides a 2-approximation of the optimum, i.e., $apxr = 2$, but it cannot be used for robustness purposes when considering $\mathcal{A}_1$ since it does not fulfill the condition of Lemma 2; The second solution, i.e., algorithm $A_{out}$ described in Section 3., turns out to be $\mathcal{P}$-optimal, when $\mathcal{P} = (Sh, M_1, \mathcal{A}_1)$.

**Theorem 1** *Let $\mathcal{P} = (Sh_1, M_1, \mathcal{A}_1)$. There exists a $\mathcal{P}$-optimal robust shunting algorithm $A_{rob}$ such that* $\text{POR}(\mathcal{P}, A_{rob}) = \max_{i \in I} \frac{opt(n_i,c)}{opt(r_i,c)}$.

**Proof.** We choose $A_{out}$ as $A_{rob}$, i.e., we have one different code for each car without considering chains. Such a solution is clearly feasible for any change in the cars order since it is completely independent on the incoming order. From Lemma 2, $\text{POR}(\mathcal{P}) \geq \max_{i \in I} \frac{opt(n_i,c)}{opt(r_i,c)}$, in fact every robust algorithm must assign a unique code to each car, hence it cannot pay less than $opt(n_i, c)$. Moreover, the solution provided by $A_{out}$ is optimal for $Sh_1$ when one unique code per car must be assigned and hence, it follows that $\text{POR}(\mathcal{P}, A_{out}) = \max_{i \in I} \frac{opt(n_i,c)}{opt(r_i,c)}$. □

Even though $A_{out}$ is $\mathcal{P}$-optimal for $\mathcal{A}_1$, i.e., $\text{POR}(\mathcal{P}, A_{rob}) = \text{POR}(\mathcal{P})$, it is not exact since in general $opt(n, c) \geq opt(r, c)$.

It is worth noting that the number of codes provided

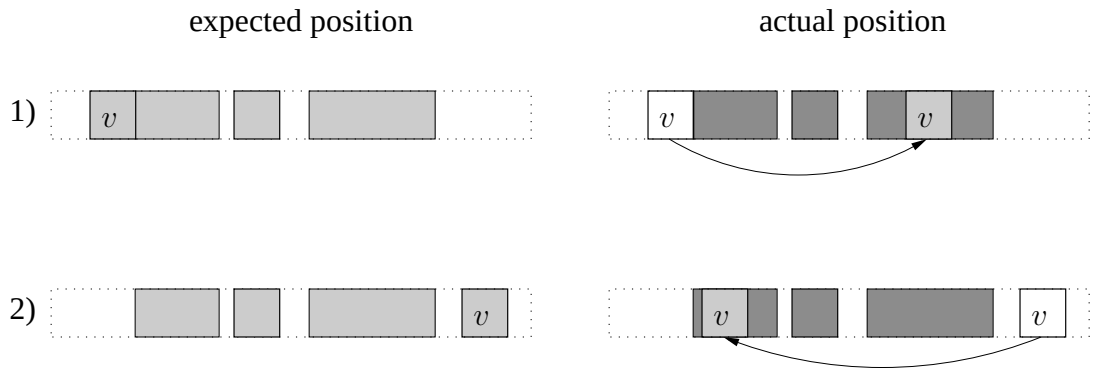


Fig. 4. A representation of the case when $v$ is the first (last, resp.) car of a pure chain and it arrives after (before, resp.) some cars of the same pure chain. The concatenation of boxes of the same gray scale represents a pure chain; the dotted box represents $T_{in}$.

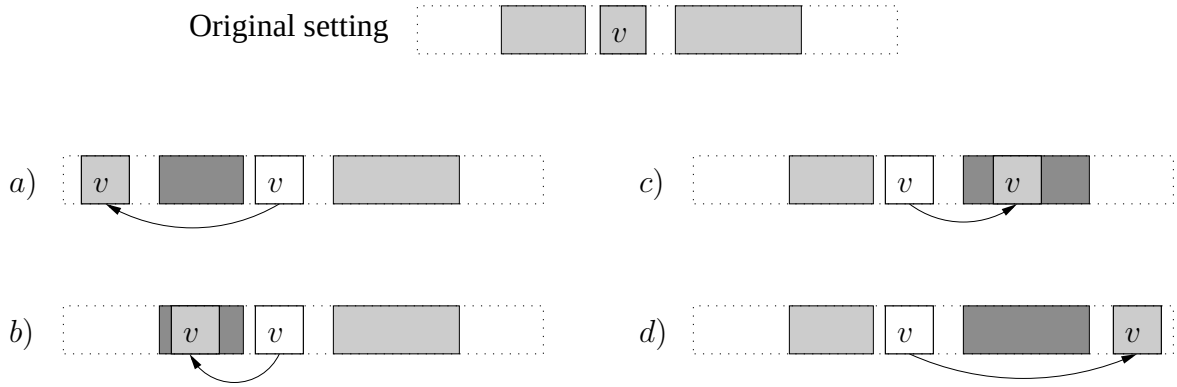


Fig. 5. A representation of cases $a$, $b$, $c$ and $d$. The concatenation of boxes of the same gray scale represents a pure chain; the dotted box represents $T_{in}$.

by the shunting algorithm $A_{rob}$ of Theorem 1 is at most $c$ times the number of codes provided by the optimal solution. In fact, we are in the case of tracks of bounded size $c$, and hence there cannot be more than $c$ cars associated with the same code. This implies that if a chain is composed by more than $c$ cars, then it must be split into more classification tracks.

**Theorem 2** *Let $\mathcal{P} = (Sh_1, M_1, \mathcal{A}_2)$. There exists a polynomial robust shunting algorithm $A_{rob}$ such that* $\text{POR}(\mathcal{P}, A_{rob}) \leq 3$.

**Proof.** The proof is structured in three parts: 1) We show that there exists a polynomial shunting algorithm for $\mathcal{P}$; 2) We show that this algorithm is robust; 3) We show that the price of robustness of this algorithm is less than or equal to 3.

(1) By Lemma 1, the change in the order of one car may produce at most one additional pure chain, hence at most one additional code is necessary to cope with such occurrence. By the 2-approximated solution proposed in [24] for $Sh_1$, the need of one additional code might imply the need of one additional track pull since it might be that codes of the original solution are already the maximum number available to manage $r_i$ chains. However, we are under $Sh_1$ assumptions, i.e., the number of tracks is unbounded. This implies that a robust algorithm must provide one additional track pull (i.e., the solution provided performs a number of track pull upper bounded by $apx(r_i, c) + 1$). This can be obtained by choosing as $A_{rob}$ an algorithm that calculates codes as in [24] for $Sh_1$ and then adding one bit (initially set to zero) corresponding to the new pull. Clearly, the proposed algorithm is polynomial since we add just one track pull operation to the 2-approximated solution proposed in [24], which is polynomial.

(2) In order to prove that $A_{rob}$ is robust we need to show that the modification of at most one code as

defined by $\mathcal{A}_2$ is enough in order to make the solution provided by $A_{rob}$ feasible with respect to any disturbance defined by $M_1$. Let $v$ be the car subject to disturbance $M_1$. From Lemma 1, if $M_1$ occurs, then the pure chain containing $v$ is split in at most two pure chains denoted as *top* and *bottom*. Without loss of generality, let us assume that $v$ belongs to *bottom*. Then an algorithm in $\mathcal{A}_2$ simply assigns the same code as planned by $A_{rob}$ to $v$ and *bottom*, and the same code but with the first bit set to one to *top*. By construction, the first pulled track contains *top*. This implies that the number of cars composing *top* is less than $c$, otherwise they could not have been associated with the same code by $A_{rob}$. Once the first pull has been performed, *top* will be placed above *bottom*, since their codes differ by just the first bit. Note that, there can be other cars between *bottom* and *top*, but this does not influence the solution since codes exactly determine the outgoing order of the cars. Hence the expected pure chain has been rebuilt and the shunting plan continues as was originally scheduled by the 2-approximation algorithm in [24].

(3) Since by point 2) $A_{rob}$ is robust with respect to $M_1$ and $\mathcal{A}_2$, by point 1) it follows that $\text{POR}(\mathcal{P}, A_{rob}) = \max_{i\in I}\frac{apx(r_i,c)+1}{opt(r_i,c)} \leq 2 + \max_{i\in I}\frac{1}{opt(r_i,c)} = 3$. □

As already said, every upper bound for $\mathcal{A}_2$ holds for $\mathcal{A}_3$. Up to now, no better upper bound for $\mathcal{A}_3$ has been found than that of $\mathcal{A}_2$.

Let us consider problems $Sh_2$ and $Sh_3$. As mentioned in Section 3., for both these cases, polynomial optimal algorithms have been proposed in [24]. If we consider $\mathcal{A}_1$, then for both $Sh_2$ and $Sh_3$, arguments similar to those of Theorem 1 can be applied, and the following theorem can be shown.

**Theorem 3** *Let $\mathcal{P}=(Sh_2,M_1,\mathcal{A}_1)$ ($\mathcal{P}=(Sh_3,M_1,\mathcal{A}_1)$ resp.). There exists a $\mathcal{P}$-optimal robust shunting algorithm $A_{rob}$ such that $\text{POR}(\mathcal{P}, A_{rob}) = \max_{i\in I}\frac{opt(n_i,w)}{opt(r_i,w)}$ ($\text{POR}(\mathcal{P}, A_{rob}) = \max_{i\in I}\frac{opt(n_i)}{opt(r_i)}$ resp.).*

If we consider $\mathcal{A}_2$, then in both $Sh_2$ and $Sh_3$, for non-trivial plans we do not need to use one additional track since any track is big enough to contain the whole train. Hence, there is always enough space to wait for the missing car/chain. The only exceptions arise when the number of track pulls required by the optimal shunting plan is too small in order to restore the expected car positions. For instance, this happens when $T_{in} \equiv T_{out}$. By applying arguments similar to those of Theorem 2, we can show the following theorem.

**Theorem 4** *Let $\mathcal{P}=(Sh_2,M_1,\mathcal{A}_2)$ ($\mathcal{P}=(Sh_3,M_1,\mathcal{A}_2)$, resp.). There exists a polynomial robust shunting algorithm $A_{rob}$ such that $\text{POR}(\mathcal{P}, A_{rob}) = \max_{i\in I}\frac{opt(r_i,w)+1}{opt(r_i,w)} = 1+\max_{i\in I}\frac{1}{opt(r_i,w)} = 2$ ($\text{POR}(\mathcal{P}, A_{rob}) \leq 1 + \max_{i\in I}\frac{1}{opt(r_i)} = 2$, resp.).*

Concerning the price of robustness of the problem, the following theorem holds.

**Theorem 5** *Let $\mathcal{P} = (Sh, M_1, \mathcal{A}_2)$. Then $\text{POR}(\mathcal{P}) \geq 2$.*

**Proof.** By Lemma 1, the change in the order of one car might imply the need of at most one additional code which in turn implies the need of one additional track pull. However, in order to be robust with respect to the considered disturbance, such additional track pull must be planned a priori by any robust algorithm $A_{rob}$ since every algorithm in $\mathcal{A}_2$, by definition, affects only codes. This implies $\text{POR}(\mathcal{P}) \geq 1 + \max_{i\in I}\frac{1}{opt(r_i,c)} = 2$ for $Sh_1$, $\text{POR}(\mathcal{P}) \geq 1 + \max_{i\in I}\frac{1}{opt(r_i,w)} = 2$ for $Sh_2$ and $\text{POR}(\mathcal{P}) \geq 1 + \max_{i\in I}\frac{1}{opt(r_i)} = 2$ for $Sh_3$. □

By Theorems 4 and 5, the following corollary can be stated.

**Corollary 1** *Let $\mathcal{P}=(Sh_2,M_1,\mathcal{A}_2)$ ($\mathcal{P}=(Sh_3,M_1,\mathcal{A}_2)$ resp.). There exists a robust shunting algorithm that is $\mathcal{P}$-optimal.*

### 4.2. *One New Car*

Given an instance $i = (T_{in}, T_{out}, W, c)$ of the shunting optimization problem $Sh$, let $M_2(i)$ represent all possible instances $(T'_{in}, T'_{out}, W, c)$ obtainable from $i$ by adding one unexpected car $v$ that was not scheduled in the original train but has to be considered in the actual shunting. For all problems $Sh_1$, $Sh_2$, $Sh_3$, $v$ should be assigned, in general, with a new code. Again, this might reflect the need of one further track pull.

**Theorem 6** *Let $\mathcal{P} = (Sh, M_2, \mathcal{A}_1)$. No robust shunting algorithm exists.*

**Proof.** In order to have a robust shunting plan with $\mathcal{A}_1$, the unexpected car $v$ should be assigned a priori by any robust algorithm $A_{rob}$ with a code independent of its outgoing placement. On the other hand, each code exactly determines the outgoing position of the corresponding car with respect to all other cars, and the claim holds. □

If we consider $\mathcal{A}_2$ or $\mathcal{A}_3$, then it is possible to find a robust shunting plan. In particular, according to the

incoming position of $v$, it might be enough to assign it with the same code of some already existent pure chain. If $v$ has to be placed at the end of the outgoing train, then it may also happen that there are some spare codes available and the problem is easily solvable. If no codes are available (this happens if the size of the codes is already minimized according to the number of cars) or the incoming position of $v$ does not allow the merge with an existent pure chain, then we need some recovery strategy. Again, the strategy must be as less "invasive" as possible.

**Theorem 7** *Let $\mathcal{P} = (Sh_1, M_2, \mathcal{A}_2)$. There exists a polynomial robust shunting algorithm $A_{rob}$ such that* $\text{POR}(\mathcal{P}, A_{rob}) = \max_{i \in I} \frac{opt(n_i+1, c-1)+1}{opt(r_i, c)}$.

**Proof.** The proof is structured in three parts: 1) We show that there exists a polynomial shunting algorithm for $\mathcal{P}$; 2) We show that this algorithm is robust; 3) We evaluate the price of robustness of the proposed algorithm.

(1) We choose as $A_{rob}$ the following modification of $A_{out}$. We consider tracks of size $c-1$ instead of $c$ and code $00\ldots0$ assigned to the new possible car. These choices imply the need of additional track pulls. Moreover, we add one bit, initially set to zero, in the rightmost position of each code. By [24] the algorithm is polynomial.
(2) By point 1) $A_{rob}$ provides a set of codes representing non-consecutive integers. This implies that wherever a new car has to be considered, there always exists an available code which an algorithm in $\mathcal{A}_2$ can use to replace code $00\ldots0$. Moreover, the constraint on $c$ is preserved by having considered $c-1$ instead of $c$. Hence, $A_{rob}$ is robust with respect to $M_2$ and $\mathcal{A}_2$.
(3) Since by point 2) $A_{rob}$ is robust, by point 1) it follows that $\text{POR}(\mathcal{P}, A_{rob}) = \max_{i \in I} \frac{opt(n_i+1, c-1)+1}{opt(r_i, c)}$. □

In order to better understand the intuition behind the proof of Theorem 7, we make use of the following example. Let $T_{in}$ be composed of 10 cars plus at most one new car, $T_{out}$ be the reverse permutation of $T_{in}$, and $c-1=3$. As in the example of Figure 3, 5 track pulls are enough to realize the shunting plan and the available codes are: 00000, 00001, 00010, 00011, 00100, 00110, 01000, 01100, 10000, 10001, 11000 that must be assigned to the possible new car and to cars from 10 to 1, respectively. For instance, if the new car must be inserted between cars 2 and 1, then we have many available codes (namely, 10010, 10011, 10100, 10101, 10110, 10111). In this case, an algorithm in $\mathcal{A}_2$ can, for instance, change 00000 in 10100. Nevertheless, if the new car must be inserted between 10 and 9, then we do not have available codes since there is no code available between 00001 and 00010. The new car may get code 00001 if it arrives after car 10 or code 00010 if it arrives before both cars 10 and 9. If the new car arrives before car 10 but after car 9 then we get in trouble since there is no way to insert it between 9 and 10 without changing other codes. In order to cope with this case we can consider a different set of codes representing non-consecutive integers. The new set of codes will be given by 000000, 000010, 000100, 000110, 001000, 001100, 010000, 011000, 100000, 100010, 110000. Now we have available codes in between any pair.

**Theorem 8** *Let $\mathcal{P}=(Sh_2,M_2,\mathcal{A}_2)$ ($\mathcal{P}=(Sh_3,M_2,\mathcal{A}_2)$ resp.). There exists a polynomial robust shunting algorithm $A_{rob}$ such that* $\text{POR}(\mathcal{P}, A_{rob}) = \max_{i \in I} \frac{opt(n_i+1, w)+1}{opt(r_i, w)}$ *(*$\text{POR}(\mathcal{P}, A_{rob}) = \max_{i \in I} \frac{opt(n_i+1)+1}{opt(r_i)}$*, resp.).*

**Proof.** In $Sh_2$, similarly to proof of Theorem 7, we use one different code for each car and preliminarily assign code $00\ldots0$ to the new car. Again, by scheduling one additional initial track pull, all codes will be not consecutive with respect to their integer representation. As a consequence, between two codes provided by $A_{rob}$ there is always a code available that an algorithm in $\mathcal{A}_2$ can use to replace code $00\ldots0$. The theorem then follows by observing that the algorithm proposed in [24] for $Sh_2$ is optimal. Similar arguments hold for $Sh_3$. □

**Lemma 3** *Let $\mathcal{P} = (Sh, M_2, \mathcal{A}_2)$. Any robust shunting algorithm $A_{rob}$ cannot assign the same code to four different cars.*

**Proof.** Assume by contradiction that four cars $u$, $v$, $w$ and $z$ arriving at the hump yard in this order get the same code in $A_{rob}$. Let $y$ be an unexpected new car that must be inserted after $v$ and before $w$, arriving at the hump yard before $u$. As $y$ arrives before $u$ and must be inserted after $v$, the two codes associated with $u$ and $v$ must be different from the two codes associated with $w$ and $z$, since otherwise the four cars $u$, $v$, $w$ and $z$ would always move all together and there would not be any possibility to insert $y$ in the middle. This implies that at least two codes must be changed in order to obtain the desired configuration. Since $\mathcal{A}_2$ allows to change at most one code, the lemma follows. □

The following corollary is a direct consequence of Lemma 3.

**Corollary 2** *Let* $\mathcal{P}=(Sh_1,M_2,\mathcal{A}_2)$ *(*$\mathcal{P}=(Sh_2,M_2,\mathcal{A}_2)$, $\mathcal{P}=(Sh_3,M_2,\mathcal{A}_2)$ *resp.).* $\text{POR}(\mathcal{P}) \geq \max_{i\in I}\frac{opt(\frac{n_i+1}{3},c)}{opt(r_i,c)}$ *(*$\text{POR}(\mathcal{P}) \geq \max_{i\in I}\frac{opt((n_i+1)/3,w)}{opt(r_i,w)}$ *and* $\text{POR}(\mathcal{P}) \geq \max_{i\in I}\frac{opt((n_i+1)/3)}{opt(r_i)}$*, resp.).*

Let us now consider the class of recovery algorithms $\mathcal{A}_3$.

**Theorem 9** *Let* $\mathcal{P}=(Sh_1,M_2,\mathcal{A}_3)$ *(*$\mathcal{P}=(Sh_2,M_2,\mathcal{A}_3)$, $\mathcal{P}=(Sh_3,M_2,\mathcal{A}_3)$ *resp.). There exists a polynomial robust shunting algorithm* $A_{rob}$ *such that* $\text{POR}(\mathcal{P},A_{rob}) = \max_{i\in I}\frac{apx(r_i+1,c)}{opt(r_i,c)}$ *(*$\text{POR}(\mathcal{P},A_{rob}) = \max_{i\in I}\frac{opt(r_i+1,w)}{opt(r_i,w)}$ *and* $\text{POR}(\mathcal{P},A_{rob}) = \max_{i\in I}\frac{opt(r_i+1)}{opt(r_i)}$*, resp.).*

**Proof.** $A_{rob}$ simply computes a set of codes for the expected cars by considering that, if a new unexpected car $v$ arrives, then one additional pure chain might be created. In this case, any algorithm in $\mathcal{A}_3$ is able to reassign all codes inserting $v$ in the desired position. □

**Theorem 10** *Let* $\mathcal{P}=(Sh_1,M_2,\mathcal{A}_3)$ *(*$\mathcal{P}=(Sh_2,M_2,\mathcal{A}_3)$, $\mathcal{P}=(Sh_3,M_2,\mathcal{A}_3)$ *resp.), then* $\text{POR}(\mathcal{P}) \geq \max_{i\in I}\frac{opt(r_i+1,c)}{opt(r_i,c)}$ *(*$\text{POR}(\mathcal{P}) \geq \max_{i\in I}\frac{opt(r_i+1,w)}{opt(r_i,w)}$ *and* $\text{POR}(\mathcal{P}) \geq \max_{i\in I}\frac{opt(r_i+1)}{opt(r_i)}$*, resp.).*

**Proof.** The proof simply follows by observing that the new unexpected car, according to its required position, may constitute itself a pure chain. The need of one further code is then necessary. □

From Theorem 9 and Theorem 10 the following corollary holds.

**Corollary 3** *Let* $\mathcal{P}=(Sh_2,M_2,\mathcal{A}_3)$ *(*$\mathcal{P}=(Sh_3,M_2,\mathcal{A}_3)$ *resp.). There exists a robust algorithm that is* $\mathcal{P}$*-optimal.*

### 4.3. *One Missing Car*

If we consider $M_3$, i.e., one missing car, then there is no change to operate in the scheduled shunting plan since cars order is preserved. This implies that any feasible shunting algorithm $A_{rob}$ is robust even though $\mathcal{A}_1$ is considered. The price of robustness in each case is then given by the corresponding best known approximation ratio.

### 4.4. *One Unavailable Track*

Given an instance $i=(T_{in},T_{out},W,c)$ of the shunting optimization problem $Sh$, let $M_4(i)$ represent all possible instances $(T_{in},T_{out},W',c)$ obtainable from $i$ if at most one track may become unavailable before the scheduled shunting plan is run, i.e. $|W'|=|W|-1$.

**Theorem 11** *Let* $\mathcal{P}=(Sh,M_4,\mathcal{A}_1)$*. No robust shunting algorithm exists.*

**Proof.** The possible unavailable track is not known in advance. A robust shunting algorithm should be able to cope with the malfunctioning of any track by avoiding to move cars in the unavailable one. It means that, in general, a strict robust plan should avoid every track. □

**Theorem 12** *Let* $\mathcal{P}=(Sh_2,M_4,\mathcal{A}_2)$*. There exists no robust shunting algorithm when the input instance is composed of* $r$ *chains with* $r>w+1$.

**Proof.** Having $r$ chains implies the need of at least $r$ codes. Since a track may become unavailable, and $\mathcal{A}_2$ allows to change only one code, then each track must be visited at most by one pure chain. This implies that there cannot be two different pure chains whose codes have a bit set to 1 in the same position. It follows that at most $w+1$ codes are available (including code $00\ldots0$). Therefore, if $r>w+1$, then there are not enough codes available to manage $r$ chains. □

**Theorem 13** *Let* $\mathcal{P}=(Sh_1,M_4,\mathcal{A}_2)$ *(*$\mathcal{P}=(Sh_3,M_4,\mathcal{A}_3)$ *resp.). There exists a polynomial robust shunting algorithm* $A_{rob}$ *which outputs a set of codes* $C$ *such that* $\text{POR}(\mathcal{P},A_{rob})=|C|+1$.

**Proof.** For $Sh_1$, the structure of the proof proceeds in three parts: 1) We show that there exists a polynomial shunting algorithm; 2) We show that this algorithm is robust; 3) We evaluate the price of robustness of the proposed algorithm.

(1) We choose as $A_{rob}$ the following algorithm. We consider tracks of size $\frac{c}{2}$ instead of $c$ and add one bit, initially set to zero, in the rightmost position of each code. We assign one different track to each pure chain (this is feasible since $w$ is unbounded) and preserve an empty track as the first to be pulled out.

(2) If a disturbance occurs, then any algorithm in $\mathcal{A}_2$ can change the code of the chain supposed to visit the unavailable track. The change must be done as follows: first, the considered pure chain is parked in the first track; after the pull of such a track, the contained pure chain is moved on top of another pure chain and merged with it. The constraint on $c$ is preserved since $A_{rob}$ considers pure chains of maximum length $\frac{c}{2}$. Hence, $A_{rob}$ is robust with respect to $M_4$ and $\mathcal{A}_2$.

(3) Since by point 2) $A_{rob}$ is robust, by point 1) we need one additional track pull, it follows that

$\text{POR}(P, A_{rob}) = \max_{i \in I} \frac{|C|+1}{opt(r_i,c)} = |C| + 1$ with $|C|$ being the number of codes generated by $A_{rob}$.

Similar arguments hold for $Sh_3$ but the one concerning tracks size constraint. In particular, if $C'$ is the set of codes generated by a robust algorithm $A_{rob}$ in this case, then $\text{POR}(\mathcal{P}, A_{rob}) = \max_{i \in I} \frac{|C'|+1}{opt(r_i)} = |C'| + 1$. □

**Theorem 14** *Let $\mathcal{P} = (Sh_1, M_4, \mathcal{A}_3)$ ($\mathcal{P} = (Sh_2, M_4, \mathcal{A}_3)$, $\mathcal{P} = (Sh_3, M_4, \mathcal{A}_3)$ resp.). There exists a polynomial robust shunting algorithm $A_{rob}$ such that $\text{POR}(\mathcal{P}, A_{rob}) = \max_{i \in I} \frac{apx(r_i,c)+1}{opt(r_i,c)} \leq 3$ ($\text{POR}(\mathcal{P}, A_{rob}) = \max_{i \in I} \frac{opt(r_i,w-1)+1}{opt(r_i,w)}$, and $\text{POR}(\mathcal{P}, A_{rob}) = \max_{i \in I} \frac{opt(r_i)+1}{opt(r_i)} \leq 2$, resp.).*

**Proof.** In any considered case, the corresponding algorithm proposed in [24] is applied with the only addition of one track pull. Since $\mathcal{A}_3$ allows to reassign any code, the only attention that $A_{rob}$ must pay is to preserve one track available (let such a track be the first scheduled for the pull) for possible substitution when the disturbance occurs. The original set of codes is then enough to manage the new situation since the same number of tracks is available. □

**Theorem 15** *Let $\mathcal{P} = (Sh, M_4, \mathcal{A}_2)$ ($\mathcal{P} = (Sh, M_4, \mathcal{A}_3)$ resp.), then $\text{POR}(\mathcal{P}) \geq 2$.*

**Proof.** In order to cope with the removal of one track among the available ones, all the cars planned to move on such a track by an optimal solution must be redirected to one or more other tracks. Each pull of the faulty track must be then reproduced by at least one other track, and it cannot be completely absorbed by other track pulls due to the optimality of the solution. Such a pull must also be planned a priori since both $\mathcal{A}_2$ and $\mathcal{A}_3$ cannot affect pulls order. This implies that at least one additional pull with respect to an optimal solution must be scheduled a priori by any robust algorithm. It then follows $\text{POR}(\mathcal{P}) \geq 1 + \max_{i \in I} \frac{1}{opt(r_i,c)} = 2$ for $Sh_1$, $\text{POR}(\mathcal{P}) \geq 1 + \max_{i \in I} \frac{1}{opt(r_i,w)} = 2$ for $Sh_2$ and $\text{POR}(\mathcal{P}) \geq 1 + \max_{i \in I} \frac{1}{opt(r_i)} = 2$ for $Sh_3$. □

From Theorem 14 and Theorem 15 the following corollary holds.

**Corollary 4** *Let $\mathcal{P} = (Sh_3, M_4, \mathcal{A}_3)$. There exists a $\mathcal{P}$-optimal robust shunting algorithm.*

## 5. Conclusion

In this paper we have studied the shunting of train cars in railways systems from the recoverable robustness point of view as defined in [26]. Robustness by itself is a not well defined property for optimization problems when recovery strategies are available and/or necessary. We have focused our attention on the definition of robust algorithms. An algorithm is said to be robust according to some allowed recovery strategy, and against some specified disturbances, if it provides a solution which is valid also if a disturbance occurs by possibly applying available recovery strategies. We also provide a measure for the price of robustness for a robust algorithm as the ratio between its performances and the performances of an optimal algorithm both applied on the expected input (without disturbances). The definition turns out to capture interesting properties among our evaluations on different shunting problems and scenarios. The proposed robust algorithms show how robustness heavily affects performances. Some algorithms that are optimal (in the robust meaning) with respect to some disturbances may become even unfeasible in other contexts. Another central issue concerns the available recovery capabilities. Intuitively, the more available recovery strategies are powerful, the less is the price of robustness for a robust algorithm. However, we have shown that there are cases where increasing recovery capabilities does not affect obtained results. Tables 2, 3 and 4 summarizes the obtained results for all the considered robustness problems arising from $Sh_1$, $Sh_2$ and $Sh_3$, respectively.

This paper gives more insight in the complex field of robust optimization. Many other applications related or not to shunting problems can be studied by following our approach.

One interesting future work would be that of considering multiple disturbances as in the robustness model proposed in [10]. Another interesting future work would be also to study the dual of robust algorithms, i.e., recovery algorithms. What would be the design of a recovery algorithm once fixed the power/capabilities of a class of robust algorithms?

## References

[1] L. Alvisi, S. Rao, and H.M. Vin. Low-overhead protocols for fault-tolerant file sharing. In *Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS)*, pages 452–461. IEEE, 1998.

Table 1

| | | Shunting Problem $Sh_1$ | | |
|---|---|---|---|---|
| Modifications | | $\mathcal{A}_1$ | $\mathcal{A}_2$ | $\mathcal{A}_3$ |
| $M_1$ | $\text{POR}(\mathcal{P})$ | $\geq \max_{i\in I} \frac{opt(n_i,c)}{opt(r_i,c)}$ | $\geq 2$ | $\geq 2$ |
| | $\text{POR}(\mathcal{P}, A_{rob})$ | $\max_{i\in I} \frac{opt(n_i,c)}{opt(r_i,c)}$ | 3 | 3 |
| $M_2$ | $\text{POR}(\mathcal{P})$ | indef. | $\geq \max_{i\in I} \frac{opt((n_i+1)/3,c)}{opt(r_i,c)}$ | $\geq \max_{i\in I} \frac{opt(r_i+1,c)}{opt(r_i,c)}$ |
| | $\text{POR}(\mathcal{P}, A_{rob})$ | no solution | $\max_{i\in I} \frac{opt(n_i+1,c-1)+1}{opt(r_i,c)}$ | $\max_{i\in I} \frac{apx(r_i+1,c)}{opt(r_i,c)}$ |
| $M_3$ | $\text{POR}(\mathcal{P})$ | 1 | 1 | 1 |
| | $\text{POR}(\mathcal{P}, A_{rob})$ | 2 | 2 | 2 |
| $M_4$ | $\text{POR}(\mathcal{P})$ | indef. | $\geq 2$ | $\geq 2$ |
| | $\text{POR}(\mathcal{P}, A_{rob})$ | no solution | $\lvert C\rvert + 1$ | 3 |

*Price of Robustness for $Sh_1$*

Table 2

| | | Shunting Problem $Sh_2$ | | |
|---|---|---|---|---|
| Modifications | | $\mathcal{A}_1$ | $\mathcal{A}_2$ | $\mathcal{A}_3$ |
| $M_1$ | $\text{POR}(\mathcal{P})$ | $\geq \max_{i\in I} \frac{opt(n_i,w)}{opt(r_i,w)}$ | $\geq 2$ | $\geq 2$ |
| | $\text{POR}(\mathcal{P}, A_{rob})$ | $\max_{i\in I} \frac{opt(n_i,w)}{opt(r_i,w)}$ | 2 | 2 |
| $M_2$ | $\text{POR}(\mathcal{P})$ | indef. | $\geq \max_{i\in I} \frac{opt((n_i+1)/3,w)}{opt(r_i,w)}$ | $\geq \max_{i\in I} \frac{opt(r_i+1,w)}{opt(r_i,w)}$ |
| | $\text{POR}(\mathcal{P}, A_{rob})$ | no solution | $\max_{i\in I} \frac{opt(n_i+1,w)+1}{opt(r_i,w)}$ | $\max_{i\in I} \frac{opt(r_i+1,w)}{opt(r_i,w)}$ |
| $M_3$ | $\text{POR}(\mathcal{P})$ | 1 | 1 | 1 |
| | $\text{POR}(\mathcal{P}, A_{rob})$ | 1 | 1 | 1 |
| $M_4$ | $\text{POR}(\mathcal{P})$ | indef. | indef. | $\geq 2$ |
| | $\text{POR}(\mathcal{P}, A_{rob})$ | no solution | no solution | $\max_{i\in I} \frac{opt(r_i,w-1)+1}{opt(r_i,w)}$ |

*Price of Robustness for $Sh_2$*

Table 3

| | | Shunting Problem $Sh_3$ | | |
|---|---|---|---|---|
| Modifications | | $\mathcal{A}_1$ | $\mathcal{A}_2$ | $\mathcal{A}_3$ |
| $M_1$ | $\text{POR}(\mathcal{P})$ | $\geq \max_{i\in I} \frac{opt(n_i)}{opt(r_i)}$ | $\geq 2$ | $\geq 2$ |
| | $\text{POR}(\mathcal{P}, A_{rob})$ | $\max_{i\in I} \frac{opt(n_i)}{opt(r_i)}$ | 2 | 2 |
| $M_2$ | $\text{POR}(\mathcal{P})$ | indef. | $\geq \max_{i\in I} \frac{opt((n_i+1)/3)}{opt(r_i)}$ | $\geq \max_{i\in I} \frac{opt(r_i+1)}{opt(r_i)}$ |
| | $\text{POR}(\mathcal{P}, A_{rob})$ | no solution | $\max_{i\in I} \frac{opt(n_i+1)+1}{opt(r_i)}$ | $\max_{i\in I} \frac{opt(r_i+1)}{opt(r_i)}$ |
| $M_3$ | $\text{POR}(\mathcal{P})$ | 1 | 1 | 1 |
| | $\text{POR}(\mathcal{P}, A_{rob})$ | 1 | 1 | 1 |
| $M_4$ | $\text{POR}(\mathcal{P})$ | indef. | $\geq 2$ | $\geq 2$ |
| | $\text{POR}(\mathcal{P}, A_{rob})$ | no solution | $\lvert C\rvert + 1$ | 2 |

*Price of Robustness for $Sh_3$*


[2] Y.P. Aneja, R. Chandrasekaran, and K.P.K. Nair. Maximizing residual flow under arc destruction. *Networks*, 38(4):194–198, 2001.

[3] H. G. Bayer and B. Sendhoff. Robust Optimization - A Comprehensive Survey. *Computer Methods in Applied Mechanics and Engineering*, 196(33-34):3190–3218, 2007.

[4] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. *Mathematical Programming: Special Issue on Robust Optimization*, volume 107. Springer, Berlin, 2006.

[5] D. Bertsimas and M. Sim. The price of robustness. *Operations Research*, 52(1):35–53, 2004.

[6] U. Blasum, M.R. Bussieck, W. Hochstättler, C. Moll, H.-H. Scheel, and T. Winter. Scheduling trams in the morning. *Mathematical Methods of Operations Research*, 49(1):137–148, 1999.

[7] G. Calinescu and P. Wan. Range assignment for biconnectivity and k-edge connectivity in wireless ad hoc networks. *Mobile Networks and Applications*, 11(2):121–128, 2006.

[8] G. Chockler, R. Guerraoui, I. Keidar, and M. Vukolic. Reliable distributed storage. *IEEE Computer*, 42(4):60–67, 2009.

[9] S. Cicerone, G. D'Angelo, G. Di Stefano, D. Frigioni, and A. Navarra. Robust algorithms and price of robustness in shunting problems. In *Proceedings of the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS)*, pages 175–190. Schloss Dagstuhl, 2007.

[10] S. Cicerone, G. Di Stefano, M. Schachtebeck, and A. Schöbel. Dynamic algorithms for recoverable robustness problems. In *Proceedings of the 8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS)*. Schloss Dagstuhl, 2008.

[11] S. Cornelsen and G. Di Stefano. Track assignment. *Journal of Discrete Algorithms*, 5(2):250–261, 2007.

[12] E. Dahlhaus, P. Horak, M. Miller, and J. F. Ryan. The train marshalling problem. *Discrete Applied Mathematics*, 103(1-3):41–54, 2000.

[13] M. Demange, G. Di Stefano, and B. Leroy-Beaulieu. On the online track assignment problem. Technical Report ARRIVAL-TR-0028, ARRIVAL Project, December 2006.

[14] M. Demange, G. Di Stefano, and B. Leroy-Beaulieu. Online Bounded Colorings. In *Proceedings of 4th Latin-American Algorithms, Graphs and Optimization Symposium (LAGOS)*, 2007.

[15] G. Di Stefano and M.L. Koči. A graph theoretical approach to the shunting problem. *Electr. Notes Theor. Comput. Sci.*, 92:16–33, 2004.

[16] G. Di Stefano, Jens Maue, Maciej Modelski, A. Navarra, Marc Nunkesser, and John van den Broek. Models for rearranging train cars. Technical Report ARRIVAL-TR-0089, ARRIVAL Project, 2007.

[17] D. Eppstein. Finding the k shortest paths. *SIAM Journal on Computing*, 28(2):652–673, 1999.

[18] M. Fischetti and M. Monaci. Robust optimization through branch-and-price. In *Proceedings of the 37th Annual Conference of the Italian Operations Research Society (AIRO)*, 2006.

[19] R. Freling, R. M. Lentink, L. G. Kroon, and D. Huisman. Shunting of passenger train units in a railway station. *Transportation Science*, 39(2):261–272, 2005.

Received 24-11-2008; revised 11-06-2009; accepted 31-08-2009

[20] A. Galluccio and G. Proietti. Polynomial time algorithms for 2-edge-connectivity augmentation problems. *Algorithmica*, 36(4):361–374, 2003.

[21] N. Garg, V. Santosh, and A. Singla. Improved approximation algorithms for biconnected subgraphs via better lower bounding techniques. In *Proceedings of the 4th annual ACM-SIAM Symposium on Discrete algorithms (SODA)*, pages 103–111. SIAM, 1993.

[22] M. Groetschel, S. O. Krumke, and J. Rambau. Online optimization of complex transportation systems. In *Online Optimization of Large Scale Systems*, pages 705–730. Springer, 2001.

[23] R. S. Hansman and U. T. Zimmermann. Optimal sorting of rolling stock at hump yard. In *Mathematics - Key Technology for the Future: Joint Project Between Universities and Industry*, pages 189–203. Springer, 2008.

[24] R. Jacob. On shunting over a hump. Technical Report 576, Institute of Theoretical Computer Science, ETH Zürich, 2007.

[25] Riko Jacob, Peter Marton, Jens Maue, and Marc Nunkesser. Multistage methods for freight train classification. In *Proceedings of the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS)*, pages 158–174. Schloss Dagstuhl, 2007.

[26] C. Liebchen, M. Lüebbecke, R. H. Möhring, and S. Stiller. Recoverable robustness. Technical Report ARRIVAL-TR-0066, ARRIVAL Project, 2007.

[27] E. Nardelli, G. Proietti, and P. Widmayer. A faster computation of the most vital edge of a shortest path. *Information Processing Letters (IPL)*, 79(2):81–85, 2001.

[28] E. Nardelli, G. Proietti, and P. Widmayer. Finding the most vital node of a shortest path. In *Proceedings of the 7th International Computing and Combinatorics Conference (COCOON)*, volume 2108 of *LNCS*, pages 278–287. Springer, 2001.

[29] D. Patterson, G. Gibson, and R. Katz. *A Case for Redundant Arrays of Inexpensive Disks (RAID)*. University of California Berkley, 1988.

[30] L. Roditty. On the k-simple shortest paths problem in weighted directed graphs. In *Proceedings of the 18th annual ACM-SIAM symposium on Discrete algorithms*, pages 920–928. SIAM, 2007.

[31] T. Winter and U. Zimmermann. Real-time dispatch of trams in storage yards. *Annals of Operations Research*, 96:287–315(29), 2000.

[32] Y.Chen, A. Hu, K. Yip, J. Hu, and Z. Zhong. Finding the most vital node with respect to the number of spanning trees. In *IEEE International Workshop on Neural Networks for Signal Processing*, volume 2, pages 1670–1673, 2003.