Optimal Algorithms for Train Shunting and Relaxed List Update Problems

Tim Nonner[1] **and Alexander Souza**[2]

1 **IBM Research Zurich, Switzerland,** tno@zurich.ibm.com
2 **Apixxo AG, Switzerland,** alex.souza@apixxo.com

Abstract

This paper considers a Train Shunting problem which occurs in cargo train organizations: We have a locomotive travelling along a track segment and a collection of n cars, where each car has a source and a target. Whenever the train passes the source of a car, it needs to be added to the train, and on the target, the respective car needs to be removed. Any such operation at the end of the train incurs low shunting cost, but adding or removing truly in the interior requires a more complex shunting operation and thus yields high cost. The objective is to schedule the adding and removal of cars as to minimize the total cost. This problem can also be seen as a relaxed version of the well-known List Update problem, which may be of independent interest.

We derive polynomial time algorithms for Train Shunting by reducing this problem to finding independent sets in bipartite graphs. This allows us to treat several variants of the problem in a generic way. Specifically, we obtain an algorithm with running time $\mathcal{O}\left(n^{5/2}\right)$ for the uniform case, where all low costs and all high costs are identical, respectively. Furthermore, for the non-uniform case we have running time of $\mathcal{O}\left(n^3\right)$. Both versions translate to a symmetric variant, where it is also allowed to add and remove cars at the front of the train at low cost. In addition, we formulate a dynamic program with running time $\mathcal{O}\left(n^4\right)$, which exploits the special structure of the graph. Although the running time is worse, it allows us to solve many extensions, e.g., prize-collection, economies of scale, and dependencies between consecutive stations.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.1.6 Optimization, G.2.1 Combinatorics

Keywords and phrases Train shunting, optimal algorithm, independent set, dynamic programming

Digital Object Identifier 10.4230/OASIcs.ATMOS.2012.97

1 Introduction

This paper considers a Train Shunting problem where we are given a set of n cars and a set of stations, and each car has a *source* station and a later *target* station. Moreover, we have a locomotive which visits the stations in a predefined order, and once the locomotive passes the source of a car, it needs to be added to the current train configuration. On the other hand, once its target is passed, it needs to be removed. Any such action is called a *shunting operation*. Adding or removing a car at the end of the train is called an *outer* shunting operation and incurs *low* cost, but adding or removing truly in the interior requires a more complex *inner* shunting operation and thus yields *high* cost. The objective is to schedule the adding and removal of cars as to minimize the total shunting cost. This problem actually originated from a discussion at the Deutsche Bahn AG. Thus, even though it is simple and stated cleanly mathematically, it has a concrete practical application. We will explain later that we can also think of Train Shunting as a relaxed version of the well-known List Update problem.

12th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'12).
Editors: Daniel Delling, Leo Liberti; pp. 97–107

OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Contributions. We derive polynomial time algorithms for Train Shunting by reducing this problem to finding an independent set in bipartite graphs. Our approach, described in Section 2, works as follows: We first observe that any two cars exclude each other from using a cheap outer shunting operation if they *overlap*, a term which will be explained later. The key observation is then that these overlap dependencies can be captured in a bipartite constraint graph, and our main theorems states that any maximal (with respect to inclusion) independent set in this graph corresponds to a set of cars that can be served with cheap outer shunting operations, and vice versa. The proofs show how to convert any maximal independent set into a solution for Train Shunting algorithmically in $\mathcal{O}\left(n^2\right)$ running time. It is well-known that the weighted and unweighted Independent Set problem is polynomially solvable in the case of bipartite graphs.

This allows us to treat several variants of the problem in a generic way, see Section 3. Specifically, we obtain an algorithm with running time $\mathcal{O}\left(n^{5/2}\right)$ for the uniform case, where all low costs and all high costs are identical, respectively. Furthermore, for the non-uniform case we have running time of $\mathcal{O}\left(n^3\right)$. Both versions translate to a symmetric variant, where it is also allowed to add and remove cars at the front of the train at low cost. In addition, in Section 4, we formulate a dynamic program with running time $\mathcal{O}\left(n^4\right)$, which exploits the special structure of the graph. Although the running time is worse, it allows us to solve many extensions, for instance, economies of scale, dependencies between consecutive stations, and price-collection. Specifically, in the economies of scale variant, we provide a discount if many inner shunting operations are performed at the same station. Dependencies between stations, for example, occur if we want to avoid that two consecutive stations need to perform inner shunting operations. Finally, in the price-collection variant, we get paid for transporting cars, and the goal is find the best tradeoff between profit and shunting operation cost.

Related work. Shunting problems are usually considered in the context of a single *hump-yard* or *shunting-yard* which serves as a central facility to rearrange trains. Specifically, a standard hump-yard has a single *incoming* track where the trains arrive and leave and several *classification* tracks. When a car wants to change its current classification track, it always needs to be pulled first to the central incoming track via a *pull-out* operation, and then pushed back to the destination classification track via a *roll-in* operation. For a more detailed description, we refer to the survey of Gatto et al. [8]. This is the most basic variant of this problem, but there are many variants depending on the allowed shunting operations or topologies [12]. For instance, the case of two incoming tracks, also called a *marshalling-yard*. Di Stefano and Koci [5] use a graph coloring approach to deal with this topology, including overnight operations. This is also the paper the most related to ours, but their constraint graph still differs significantly and they do not provide customized algorithms as our quite flexible dynamic program. For an NP-hard version of the problem, Beygang, Dahms, and Krumke [2] gave lower bounds on the optimal objective value and derived approximation algorithms for offline and online variants. However, all these variants consider *hub-and-spoke-systems* where a central facility is used to perform all shunting operations [18, 3]. A multistage variant was considered by Jacob et al. [14], where an encoding of classification schedules was introduced, which allows characterizing train classifcation methods as classes of schedules and yields simpler and more precise analysis of well-known classification methods.

By contrast, our problem considers shunting from a more global perspective since the evolution of a train is treated from its origin to its destination. Already in 2006, a similar perspective was taken in a seminal operations research paper by Kroon et al. [16], in which the construction of the dutch timetable is explained. A novel property of this timetable is that it features robustness in spite of growing passenger and cargo demands. One key

challenge to achieving this was to be able to treat train scheduling problems in a global manner. The robustness of timetables continues to be an active area of research, e.g., by Cicerone et al. [4].

Relations to list update. In the List Update problem, we are given a linked list, which supports the operations *put* and *get*. A call of Put(i, x) stores a data item x at position i in the list, yielding access cost of i. A call of Get(x) returns and removes x (if present) from the list, at access cost equal to the position of x. Recall that this classical List Update problem was introduced in a seminal paper by Sleator and Tarjan [21]. It was shown by Ambühl et al. [1] that the offline List Update problem is NP-hard. Being a fundamental problem in computer science, many variants have been proposed for the List Update problem: For example, Martinez and Roura [17] and Munro [19] defined the MRM model with an alternative cost model. There, it is allowed that the ordering of the elements preceeding the one which is accessed can be changed free of charge. Golynski and López-Ortiz [10] were able to give a polynomial time algorithm for the MRM list update model.

The Relaxed List Update problem, which we introduce here, features the following cost model: An access at the head of the list encounters *low* cost $c \geq 0$. Otherwise, we are charged a *high* cost of $c' > c$. That is, the access is either cheap, if it is at the head of the list, or expensive, if it is not. Observe that each sequence of put/get operations translates directly into an instance of Train Shunting with uniform low cost c and high cost $c' > c$. This model is more crude than the classical one, but it has the desireable property that the corresponding offline problem can be solved in low order polynomial time as we show in this paper.

Preliminaries. We denote the set of cars by $J = \{1, 2, \dots, n\}$ and the set of stations by $I = \{1, 2, \dots\}$ which are passed by the locomotive in this order. For each car $j \in J$, we associate two *events*, the *source* s_j and the *target* t_j, which induces the set $S = \{s_1, \dots, s_n\}$ of sources and the set $T = \{t_1, \dots, t_n\}$ of targets. Moreover, for each event $e \in S \cup T$, there is an associated station $i(e)$ where this events takes place such that $i(s_j) < i(t_j)$ for each car j. A solution for the problem is a *schedule* which defines the change of the train configuration at each event e, from which we can derive the exact positions where to add cars over time. If a car j is added to the exact end of the train, then only an *outer* shunting operation with low cost $c_j \geq 0$ is required, but otherwise, if j is to be added to some other position, and hence the train needs to be split, an *inner* shunting operation with high cost $c'_j > c_j$ is necessary. Similarly, removing a car from the end of the train results in an outer shunting operation with low cost c_j, but removing it from any other position requires high cost c'_j. Let $w_j := c'_j - c_j$ denote the additional cost of an inner shunting operation. The goal is hence to find a schedule that minimizes the additional cost due to inner shunting operations. We refer to the case where all c'_j and c_j are identical as Uniform Train Shunting, and we may then assume that $c'_j = 1$, $c_j = 0$, and hence $w_j = 1$.

2 Train Shunting and Bipartite Independent Sets

In this section, we give a construction reducing Train Shunting to finding independent sets in bipartite graphs. To this end, let us assume for the moment that we have a strict ordering $<$ of the events $S \cup T$ which defines the temporal order in which these events are executed. Clearly, this strict ordering needs to be *consistent* with the natural ordering of the stations such that $e < e'$ implies $i(e) \leq i(e')$ for any pair of events e, e'. We will expose in Lemma 5 an algorithm how to find an optimal such ordering. Using this, if $s_j < s_{j'} < t_j < t_{j'}$, then we say that the cars j, j' are *overlapping*, and otherwise, we say that they are *independent*.

Now that we are given the ordering of the shunting events, it turns out that only the overlapping pairs of cars j, j' are problematic: Either the addition of car j' or the removal of car j can not be a (cheap) outer shunting operation. Therefore, we have to decide which car encounters an outer and which an inner shunting operation. Motivated by this observation, we define the following bipartite graph $G = (S \cup T, E)$ with vertices $S \cup T$ and edges E, which is called the *constraint graph* of the Train Shunting problem and can trivially be constructed in $\mathcal{O}\left(n^2\right)$ time. For any two overlapping cars j, j', we add the edge $\{s_{j'}, t_j\}$ to the set E of edges of the graph G.

For a graph $G = (V, E)$ with vertices V and edges E, a subset $U \subseteq V$ is called an *independent set* in G, if $u, v \in U$ imply $\{u, v\} \notin E$. An independent set U is called *maximal* if it is maximal with respect to inclusion. It is called *maximum*, if it is maximal with respect to total weight.

Now we formally define the solution of Train Shunting as follows: Any sequence C over elements of J is called a *configuration*, and the left-most element in C is called the *end*. Using this, a solution $\mathcal{C}$ defines a configuration $C(e)$ for each event e such that $j \in C(e)$ for $s_j \leq e < t_j$ and $j \notin C(e)$ for $e < s_j$ and $e \geq t_j$.

The following two theorems establish an equivalence between solutions for Train Shunting and independent sets in bipartite graphs.

▶ **Theorem 1.** *Let $\mathcal{C}$ be any solution of the* Train Shunting *problem and let U be the events having outer shunting operations. Then U is an independent set in G.*

Proof. Let $s_{j'}, t_j \in U$ and assume $\{s_{j'}, t_j\} \in E$. Then there is an overlapping pair of cars j, j', i. e., with $s_j < s_{j'} < t_j < t_{j'}$. In the solution $\mathcal{C}$, both events $s_{j'}$ and t_j have low cost since they are both members of U. When the event t_j occurs, i.e., the removal of car j, the car is at the end. However, at event $s_{j'}$, the car j' is added at the end, while car j is still in the configuration. Thus, car j can not be at the end at event t_j, which is a contradiction. ◀

▶ **Theorem 2.** *Let U be a maximal independent set in G. Then there is a solution $\mathcal{C}$ for* Train Shunting *such that exactly the events U have outer shunting operations.*

The proof of the theorem follows immediately from Lemma 3 and Lemma 4 and is organized as follows: The graph G and the independent set U induce an acyclic digraph H. This digraph yields an ordering π of J. Finally, π yields a solution $\mathcal{C}$, such that exactly the events in the independent set U yield an outer shunting operation.

Let U be any independent set in G. We construct $H = H(U) = (W, A)$ as follows: Associate a vertex in H with every car, i.e., $W = J$. For an event e, let $j = j(e)$ be the *associated* car, i.e., $e = s_j$ or $e = t_j$. Let j be any car and consider the events e' with $s_j < e' < t_j$ and associated car $j' = j(e')$. If $e' = s_{j'}$ and $s_{j'} \in U$, add the edge (j, j') to H, but if $e' = t_{j'}$ and $t_{j'} \in U$, add the edge (j, j') to H. The graph H can be constructed in $\mathcal{O}\left(n^2\right)$ time.

▶ **Lemma 3.** *Let U be any independent set in G. Then the induced digraph H is acyclic and can be constructed in $\mathcal{O}\left(n^2\right)$ time.*

Proof. Assume, for sake of contradiction, that H contains a directed cycle C, say. First observe that any car j with $s_j, t_j \notin U$, by construction of H, does not have any incoming edges $(\cdot, j)$ in H. Thus, such a car can not occur in a cycle. Hence we may assume that each car j in a cycle satisfies $s_j \in U$ or $t_j \in U$ (or both).

Now let $C = (j_1, \ldots, j_k)$ be any directed cycle in H. We may assume that car j_1 has the smallest source. Since the edge (j_k, j_1) must exist in H we must have $s_1 < s_k < t_1 < t_k$ and

$t_1 \in U$. This implies $s_k \notin U$ since the graph G contains the edge $\{s_1, t_k\}$ as the cars 1 and k overlap. Therefore $t_k \in U$. The graph H must also contain the edge (j_{k-1}, j_k). Hence we must have $s_{k-1} < t_k < t_{k-1}$ by construction. Thus we have $s_1 < s_{k-1} < t_1 < t_{k-1}$ or $s_k < s_{k-1} < t_k < t_{k-1}$. As $t_1, t_k \in U$ we can not have $s_{k-1} \in U$ since the car $k-1$ overlaps with car 1 or k. Hence $t_{k-1} \in U$. We continue this reasoning and deduce $t_k < t_{k-1} < \cdots < t_2$, $t_k, \ldots, t_2 \in U$, and $s_k, \ldots, s_2 \notin U$. Since the edge (j_1, j_2) must exist in H we must have $s_1 < s_2 < t_1 < t_k < t_2$ and $s_2 \in U$, which is a contradiction. ◀

▶ **Lemma 4.** *Let U be any maximal independent set in G and H be the induced acyclic digraph. Then H induces a solution $\mathcal{C}$ for* Train Shunting *such that exactly the events U yield an outer shunting operation and which can be constructed in $\mathcal{O}\left(n^2\right)$ time.*

Proof. First we construct an ordering π on J. Recall from Lemma 3 that the digraph H is acyclic, and moreover recall that a vertex without outgoing edges is called a sink. Every directed acyclic graph has a sink. Initially, π is the empty sequence. Let j be a sink in H, append j to π, and remove j from H. Continue analogously removing sinks until H is empty. This takes running time of $\mathcal{O}\left(n^2\right)$.

Let $I = (e_1, \ldots, e_{2n})$ be the sequence of events with $e_1 < e_2 < \ldots < e_{2n}$. For the sequence $I_k = (e_1, \ldots, e_k)$, the set of *active* cars A_k is the set of cars j such that $s_j \in I_k$ but $t_j \notin I_k$. For any set $J' \subseteq J$ define by $\pi(J')$ the subsequence of π induced by J'. We define the following family of configurations $\mathcal{C}$: $C(e_k) = \pi(A_k)$ for $k = 1, \ldots, 2n$.

Now we have to show that exactly the events in U incur an outer shunting in $\mathcal{C}$. Let $s_j \in U$ be the source of some car j. Let A be the set of active cars at event s_j, i.e., the cars j' with $s_{j'} < s_j < t_{j'}$. By construction, the graph H contains the edge (j', j) and hence j precedes j' in π. In particular, at the event s_j, the sequence $\pi(A)$ begins with the car j and the car incurs an outer shunting. Analogously, if $t_j \in U$. Now let $s_j \notin U$. Since U is a maximal independent set, the vertex s_j in G is incident with an edge $\{s_j, t_{j'}\}$ and $t_{j'} \in U$. We hence have $s_{j'} < s_j < t_{j'} < t_j$. By construction, H contains the edge (j, j') and j' precedes j in π. In particular, at the event s_j, the car j' is a member of the active set A of cars and hence $\pi(A)$ does not begin with car j. Hence the car j incurs an inner shunting in the configuration at event s_j. Analogously, if $t_j \notin U$. ◀

Having established the correspondence between independent sets and solutions for Train Shunting, we are in the position to remove the assumption that we have an explicit ordering $<$ over the events. Assume that there are events e, e' with associated stations $i(e) = i(e')$. In this case we are free to choose the ordering of the events, provided that this ordering is consistent. There are two ways to break the tie: Either $e < e'$ or $e' < e$. However, the two orderings may yield different optimal objective values and it is questionable which one to choose. The following result gives the answer.

▶ **Lemma 5.** *An optimal consistent ordering can be constructed using the following rule: Whenever we need to break a tie for two events, break the tie such that the two corresponding cars become independent. This ordering can be computed in $\mathcal{O}(n^2)$ time.*

Proof. Observe that the two constraint graphs of two orderings that break any fixed tie have the property that one is a subgraph of the other. Hence any independent set in the supergraph is also one in the subgraph. Further, it is easy to see that the claimed ordering yields the smallest number of edges in a constraint graph which is consistent with the input instance, hence proving the claim. ◀

3 Generic Algorithms

Theorems 1 and 2 give rise to a polyomial time algorithm Generic, which solves Train Shunting optimally: Construct the constraint graph G, find a maximum independent set U in G, construct the graph H, and deduce the sequence π, which yields an optimal solution $\mathcal{C}$.

▶ **Corollary 6.** Generic *solves* Train Shunting *optimally and can be implemented to run in time* $\mathcal{O}\left(n^2 + \tau(n)\right)$*, where* $\tau(n)$ *denotes a polynomial time for finding a maximum independent set.*

Proof. The optimality of the algorithm follows from Theorems 1 and 2. For the running time, we have already argued that all the steps, except for the construction of the independent set U, can be done in $\mathcal{O}\left(n^2\right)$ time. For general graphs, it is NP-complete to decide on the existence of an independent set of a certain size or weight, see Garey and Johnson [7]. However, it is well known that, for bipartite graphs, maximal independent sets with minimal weight can be constructed in polynomial time, e.g., with the Ellipsoid method, see Grötschel, Lóvász, and Schrijver [11]. This is due to the fact that the constraint matrix in the canonical integer linear program is the incidence matrix of a bipartite graph. Such matrices are totally unimodular and it is well known that the extreme point solutions of the linear programming relaxation are integral. Thus the Ellipsoid algorithm already yields that Generic has polynomial running time. ◀

Depending on the weight structure of the constraint graph, we can obtain low order polynomial running times by using combinatorial algorithms.

3.1 Uniform Cost

Recall that in Train Shunting with uniform cost, which is identical to Relaxed List Update, we may assume that $w_j = 1$. Thus, the reduction derived in Section 2 gives an unweighted bipartite graph G, and hence the goal is to simply find an independent set of maximum size.

▶ **Corollary 7.** Generic *solves* Uniform Train Shunting *optimally and can be implemented to run in time* $\mathcal{O}\left(n^{5/2}\right)$.

Proof. The problem can be reduced to finding a minimum cut in the following auxiliary network $G' = (S \cup T \cup \{u, v\}, A)$ with the following set A of arcs: For each edge $\{s, t\} \in E$ with $s \in S$ and $t \in T$ introduce (s, t) and add (u, s) for all $s \in S$ and (t, v) for all $t \in T$. All capacities are equal to one. Now, a minimum $u - v$-cut (X, Y) with $u \in X$ and $v \in Y$ in G' corresponds to a maximum independent set $U = (S \cap Y) \cup (T \cap X)$ in the original graph. In the present special case, the algorithm Even-Tarjan [6] finds a solution in time $\mathcal{O}\left(n^{5/2}\right)$. ◀

Alternatively, there is a similar reduction to Bipartite Matching, which also solves the problem in time $\mathcal{O}\left(n^{5/2}\right)$ with the Hopcroft-Karp [13] algorithm. For further details on the well-known equivalence between Independent Set, Matching, and Vertex Cover in case of bipartite graphs, see standard textbooks like, e.g., [15].

3.2 Non-Uniform Cost

Here we treat the general Train Shunting problem with non-uniform additional cost w_j for inner shunting operations.

▶ **Corollary 8.** Generic *solves* Train Shunting *optimally and can be implemented to run in time* $\mathcal{O}(n^3)$.

Proof. Here, the reduction used in Corollary 7 is modified with respect to capacities: The capacity of any arc (s,t) for $s \in S$ and $t \in T$ is unbounded, any arc (u,s) for $s \in S$ has capacity w_j for the car j with $s_j = s$, and any arc (t,v) for $t \in T$ has capacity w_j for the car j with $t_j = t$. These are the weights defined in the constraint graph defined above. For this case, e.g., the algorithm Goldberg-Tarjan [9] solves the problem in time $\mathcal{O}(n^3)$. ◀

3.3 Symmetric Train Shunting

In the Symmetric Train Shunting problem, the low cost for an outer shunting apply at the *front or end* of the train. This variant reduces to the ordinary Train Shunting problem as follows:

▶ **Corollary 9.** *The* Generic *algorithm can be modified to solve* (Uniform) Symmetric Train Shunting *with the same respective asymptotic running time as in the ordinary variant.*

Proof. Let $G = (S \cup T, E)$ be the constraint graph for (Uniform) Train Shunting. Introduce a new graph G'' as follows. Let $G' = (S' \cup T', E')$ be a copy of G. Now define $G'' = ((S \cup T') \cup (T \cup S'), E \cup E' \cup F)$, with the following set $F = \{\{s, s'\} \mid s \in S\} \cup \{\{t, t'\} \mid t \in T\}$. That is, the left side of G'' consists of the vertices $S \cup T'$, while the right side consists of the vertices $T \cup S'$. Observe that the edges in F only connect vertices on different sides. Hence G'' is bipartite and we are interested in a maximum independent set therein.

The vertices of G' correspond to the events that occur at the front of the train, while the vertices of G, as usual, correspond to the events that occur at the end. The edges F ensure that no event occurs at the same time at the beginning and at the end of the train. Observe that the construction of deriving a solution of (Uniform) Train Shunting from an independent set in G'' transfers analogously to (Uniform) Symmetric Train Shunting.

As we have only at most tripled the set of vertices and edges, the asymptotic running times remain the same. ◀

4 Dynamic Programming

In this section, we give a dynamic programming approach to solve Train Shunting. For the sake of exposition, we only explain the dynamic program in detail for Uniform Train Shunting with $w_j = 1$, general Train Shunting and several extensions are then sketched in the contained subsections.

Recall the strict temporal ordering $<$ of the events from Lemma 5 which extends the ordering induced by the stations. Because ties are here broken in favor of generating independent car pairs, it follows for each overlapping pair of cars j, j' that even $i(s_j) < i(s_{j'}) < i(t_j) < i(t_{j'})$. We then say that j *left-overlaps* with j', and analogously, we say that j' *right-overlaps* with j. Moreover, the reduction of Uniform Train Shunting to finding a maximum independent set in Section 2 can be interpreted such that the goal is to *mark* as many events as possible subject to the constraint that, for any overlapping pair of cars j, j' with $i(s_j) < i(s_{j'}) < i(t_j) < i(t_{j'})$, at most one of the events $s_{j'}$ and t_j is marked. Thus, marking an event corresponds to adding this event to the independent set, and therefore, marking the source and target event of a car means that we use a cheap outer shunting

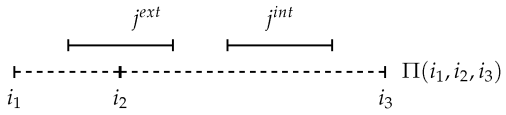

Figure 1 Subinstance.

operation to add and remove it to the train, respectively. Finally, note that we can also think of a car j as an interval $[s_j, t_j]$ with left endpoint s_j and right endpoint t_j. We will use this interpretation in the figures throughout this section.

We will now outline the dynamic program. To this end, consider a dynamic programming array Π with entries of the form $\Pi(i_1, i_2, i_3)$, where the $i_1 \leq i_2 \leq i_3$ are stations from the set I. Since we may assume that $I = \{1, 2, \ldots, 2n\}$, the array Π has thus polynomial size $\mathcal{O}(n^3)$. The goal is to fill Π such that each entry $\Pi(i_1, i_2, i_3)$ gives the maximum number of events which can be marked in the subinstance consisting of the cars j with

(1) $i_2 \leq i(s_j) < i(t_j) \leq i_3$, in which case we call j an *internal car*, or
(2) $i_1 \leq i(s_j) < i_2 \leq i(t_j) \leq i_3$, in which case we call j an *external car*,

where we are only allowed to mark target events of external cars. An example subinstance is schematically depicted in Figure 1, where j^{ext} is an external car and j^{int} is an internal car.

To see how to fill Π, consider a fixed entry $\Pi(i_1, i_2, i_3)$. Note that it always holds that the latest target event of some car is marked, since there is no other car which might right-overlap with this car. Therefore, since there is hence always some car whose target event is marked, we may distinguish the following cases:

Case 1. The only cars j whose target events t_j are marked are the ones with $i(t_j) = i_3$. In this case, $\Pi(i_1, i_2, i_3) = |A(i_1, i_2, i_3)| + |R(i_1, i_2, i_3)|$, where $A(i_1, i_2, i_3)$ denotes the set of internal cars and $R(i_1, i_2, i_3)$ denotes the set of internal or external cars j with $i(t_j) = i_3$. We add $A(i_1, i_2, i_3)$ since we may assume that all source events of internal cars are marked.

Case 2. There is an external car j whose target event t_j is marked and $i(t_j) < i_3$. In this case, pick the external car j with marked target event t_j and latest target station $i(t_j) < i_3$, where ties are broken arbitrarily. We collect several facts:

(1) There is no internal car j' whose source event is marked which right-overlaps with j.
(2) If there is an internal car j' which right-overlaps with j and whose target event t_j is not marked, then there is another internal car j'' with $i(s_{j''}) \geq i(t_j)$ which right-overlaps with j' whose source event is marked. First, there must be such an interval j'', since we would otherwise mark t_j. Moreover, if $i(s_{j''}) < i(t_j)$, and hence j'' right-overlaps with j, then it is not possible that t_j is marked.
(3) By the selection of j, there is no external car j' with marked target event $t_{j'}$ and $i(t_{j'}) > i(t_j)$.

This shows that we get the decomposition $\Pi(i_1, i_2, i_3) = \Pi(i_1, i_2, i(t_j)) + \Pi(i_2, i(t_j), i_3)$, as schematically depicted in Figure 2, where the dot on the target event of car j indicates that this event is marked.

Case 3. There is no external car whose target event is marked, but an internal car j with $i(t_j) < i_3$, see Figure 3. In this case, pick the internal car j with marked target event t_j, $i(t_j) < i_3$, and earliest source station $i(s_j)$, where ties are broken arbitrarily. Note that the facts (1) and (2) from the last case also hold in this case. We moreover obtain the following fact:

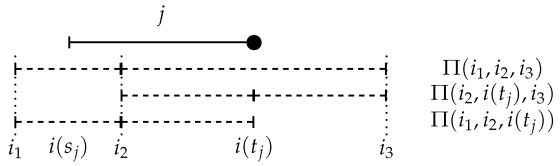

Figure 2 Marked external car.

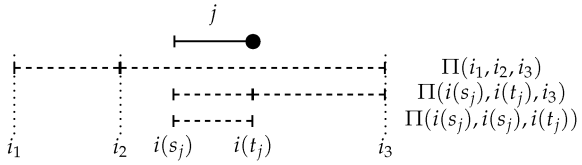

Figure 3 Marked internal car.

It holds for each internal car j' with $i(s_{j'}) < i(s_j)$ that its target event is not marked, but its source event is marked. The first fact follows from the selection of j. To see the second fact, assume that its source event is not marked. In this case, there must be a car which left-overlaps with j' whose target event is marked. However, this would be a better selection for j, leading to a contradiction as well.
This gives the decompositon $\Pi(i_1, i_2, i_3) = \Pi(i(s_j), i(s_j), i(t_j)) + \Pi(i(s_j), i(t_j), i_3) + |L(i_2, i(s_j), i_3)|$, where $L(i_2, i(s_j), i_3)$ denotes the set of internal cars j' with $i_2 \leq i(s_{j'}) < i(s_j)$.

Summarizing these cases gives the following recurrence relation:

$$\begin{aligned}\Pi(i_1, i_2, i_3) \quad = \quad & \max \Big\{ |A(i_1, i_2, i_3)| + |R(i_1, i_2, i_3)|, \\ & \max_{j \text{ external}} \{\Pi(i_1, i_2, i(t_j)) + \Pi(i_2, i(t_j), i_3)\}, \\ & \max_{j \text{ internal}} \{\Pi(i(s_j), i(s_j), i(t_j)) + \Pi(i(s_j), i(t_j), i_3) + |L(i_2, i(s_j), i_3)|\} \Big\}\end{aligned}$$

Hence, applying this recurrence relation takes $\mathcal{O}(n)$ time, which shows that Π can be filled in $\mathcal{O}(n^4)$ time if we compute all values $|A(i_1, i_2, i_3)|$, $|R(i_1, i_2, i_3)|$, and $|L(i_2, i(s_j), i_3)|$ beforehand. However, this can be done in time $\mathcal{O}(n^3)$. We conclude with the following theorem.

▶ **Theorem 10.** *The described dynamic programming scheme solves* Uniform Train Shunting *in* $\mathcal{O}(n^4)$ *time.*

4.1 Non-Uniform Cost

The dynamic programming scheme can be easily extended to general Train Shunting with arbitrary w_j. Specifically, we only need to replace the term $|A(i_1, i_2, i_3)|$ by $\sum_{j \in A(i_1, i_2, i_3)} w_j$, $|R(i_1, i_2, i_3)|$ by $\sum_{j \in R(i_1, i_2, i_3)} w_j$, and $|L(i_2, i(s_j), i_3)|$ by $\sum_{j \in L(i_2, i(s_j), i_3)} w_j$ in the recurrence relation. Note that the asymptotic running time of the dynamic program is not affected by this change.

4.2 Economies of Scale

It is natural to assume that there are economies of scale when performing shunting operations. For instance, if we do an inner shunting operation for some car j at some station, then

doing an additional inner shunting operation for another car $j' \neq j$ at the same station should not matter much more. To this end, consider a monotone increasing concave function $x \mapsto W_i(x)$ which indicates the cost savings when doing x outer shunting operations at station i. Using this, we need to replace the term $|A(i_1, i_2, i_3)|$ by $\sum_{i=i_2}^{i_3} W_i(|A_i(i_1, i_2, i_3)|)$ where $A_i(i_1, i_2, i_3) := \{j \in A(i_1, i_2, i_3) | i(s_j) = i\}$, the term $|R(i_1, i_2, i_3)|$ by $W_{i_3}(|R(i_1, i_2, i_3)|)$, and the term $|L(i_2, i(s_j), i_3)|$ by $\sum_{i=i_2}^{i(s_j)} W_i(L_i(i_2, i(s_j), i_3))$ where $L_i(i_2, i(s_j), i_3) := |\{j' \in L(i_2, i(s_j), i_3) | i(s_{j'}) = i\}|$. This allows us to treat economies of scale in the recurrence relation. Also this extension does not increase the asymptotic running time of the dynamic program.

4.3 Station-Dependencies

There might be constraints regarding the sequence of shunting operations. For instance, in order to evenly distribute delays, it might be convenient to avoid expensive inner shunting operations at two consecutive stations. To incorporate this into the dynamic programming array, for each entry $\Pi(i_1, i_2, i_3)$, we need to distinguish the cases that there is a shunting operation at station $i_2 - 1$ or not. We can do this by extending such an entry as $\Pi(i_1, i_2, i_3, x)$, where x is a boolean variable that indicates whether there is a shunting operation at station $i_2 - 1$, and if x is true, then $\Pi(i_1, i_2, i_3, x)$ gives the maximal number of events we can mark without marking any events with associated station i_2, and if x is false, then there is no such constraint. This case distinction can be easily incorporated into the recurrence relation. Since this only doubles the size of Π, the running time remains asymptotically the same. However, more complicated constraints regarding the sequence of shunting operations can be added in a similar way.

4.4 Prize-Collection

Consider the scenario that each delivered car j gives us some integral profit p_j. In this case, we need to find a trade-off in between profit and shunting cost. For the case that the profits p_j are encoded as unaries, and hence $p_j = \mathcal{O}(n)$, this price-collection scenario can be incorporated into the dynamic programming array by adding one additional price bound. That is, the entries then have the form $\Pi(i_1, i_2, i_3, p)$ for some integer p, and this entry indicates the minimum shunting cost for the case that the sum of the collected profits is at least p. If the profits are bounded by $\mathcal{O}(n)$, then this increases the size of the array to $\mathcal{O}(n^5)$. Moreover, since each recurrence relation can then be implemented in $\mathcal{O}(n^2)$ time, the total running time is $\mathcal{O}(n^7)$.

References

1 Christoph Ambühl. Offline list update is np-hard. In Paterson [20], pages 42–51.

2 Katharina Beygang, Florian Dahms, and Sven O. Krumke. Train marshalling problem: Algorithms and bounds. *Technical report*, pages 1 – 25, 2010.

3 Alberto Ceselli, Michael Gatto, Marco E. Lübbecke, Marc Nunkesser, and Heiko Schilling. Optimizing the cargo express service of swiss federal railways. *Transportation Science*, 42(4):450–465, November 2008.

4 S. Cicerone, G. D'Angelo, G. Di Stefano, D. Frigioni, and A. Navarra. Recoverable robustness for train shunting problems. *Algorithmic Operations Research*, 4(2):102–116, 2009.

5 Gabriele Di Stefano and Magnus Love Koci. A graph theoretical approach to the shunting problem. *Electr. Notes Theor. Comput. Sci.*, 92:16–33, 2004.

6 S. Even and R. E. Tarjan. Network flow and testing graph connectivity. *SIAM Journal on Computing*, (4):507–518, 1975.

7 M. R. Garey and D. S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA, 1979.

8 Michael Gatto, Jens Maue, Matús Mihalák, and Peter Widmayer. Shunting for dummies: An introductory algorithmic survey. In Ravindra K. Ahuja, Rolf H. Möhring, and Christos D. Zaroliagis, editors, *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 310–337. Springer, 2009.

9 Andrew V. Goldberg and Robert E. Tarjan. A new approach to the maximum flow problem. *Proceedings of the 18th Annual ACM Symposium on Theory of Computing (STOC '86)*, pages 136–146, 1986.

10 Alexander Golynski and Alejandro López-Ortiz. Optimal strategies for the list update problem under the mrm alternative cost model. *Inf. Process. Lett.*, 112(6):218–222, 2012.

11 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer Verlag, 1988.

12 R.S. Hansmann. *Optimal Sorting of Rolling Stock*. Cuvillier, 2010.

13 John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225 – 231, 1973.

14 R. Jacob, P. Marton, J. Maue, and M. Nunkesser. Multistage methods for freight train classification. *Networks*, 57(1):87–105, 2011.

15 Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer Verlag, 2000.

16 Leo Kroon, Dennis Huisman, Erwin Abbink, Pieter-Jan Fioole, Matteo Fischetti, Gábor Maróti, Alexander Schrijver, Adri Steenbeek, and Roelof Ybema. The new dutch timetable: The or revolution. *Interfaces*, 39(1):6 – 17, 2009.

17 Conrado Martínez and Salvador Roura. On the competitiveness of the move-to-front rule. *Theor. Comput. Sci.*, 242(1-2):313–325, 2000.

18 Marc Nunkesser Michael Gatto, Riko Jacob. Optimization of a railway hub-and-spoke system: Routing and shunting. In *WEA*, 2005.

19 J. Ian Munro. On the competitiveness of linear search. In Paterson [20], pages 338–345.

20 Mike Paterson, editor. *Algorithms - ESA 2000, 8th Annual European Symposium, Saarbrücken, Germany, September 5-8, 2000, Proceedings*, volume 1879 of *Lecture Notes in Computer Science*. Springer, 2000.

21 Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202 – 208, 1985.