Algorithms for Combinatorial Problems Related to Train Marshalling

Elias Dahlhaus
Dept. of Computer Computer Graphics,
Algorithms and Data Structures Group
Vienna University of Technology
Austria
e-mail: dahlhaus@cs.uni-bonn.de and dahlhaus@math.tu-berlin.de
phone: +49-30-8033917,

Fredrik Manne
Department of Informatics
University of Bergen
Norway,

Mirka Miller
Department of Computer Science and Software Engineering
University of Newcastle
Australia

Joe Ryan
School of Management
University of Newcastle
Australia

Abstract

We discuss a train marshalling principle on a hump yard based on radix sort. Initially we show that the number of sorting steps is dependent on the number of "chains" in the permutation π that maps the final position of each car to its initial position. A chain is here a maximal interval $I = [i, j]$, such that π is monotonically increasing. This adaptive radix sorting scheme requires that the numbers of the items form an initial segment $\{1, \ldots, n\}$ of the natural numbers. We also discuss the problem how to behave if the final position is not fixed but only has to satisfy certain requirements, e.g. cars of the same train have to appear consecutively. In general, we specify an ordering requirement by a P-Q-tree where the

leaves are the cars and inner nodes correspond to "blocks" (a train forms a block, a final destination in a train forms a block). In some blocks, the subblocks may be permuted in any order (corresponding to P-nodes), whereas in other blocks, the sequence of immediate subblocks is fixed (Q-nodes). The problem is to minimize the number of chains, given an ordering requirement and the initial positions of the cars. In general, the problem is NP-complete [4]. We discuss efficient special cases and approximative solutions.

1 Introduction

The general problem of train marshalling is to create new trains from a set of arriving trains. In our model, the "classification yard" consists of a hump and a set of parallel "classification tracks" that are joined by a common track with the hump. Any car can roll down from the hump to a classification track without a push of the engine.

There are certain criteria of optimality. Examples are the number of switches, the number of couplings and decouplings, and the number of back and forth movements of the engine. Exact solutions of certain optimization problems are expected to be NP-complete. But there are certain schemes that are discussed in the literature [3, 8]. The problems with these schemes are that they either require additional tracks (sorting by trains, sorting by block numbers) or that the tracks are occupied quite unsymetrically (triangular sorting, geometric sorting).

It is quite natural to take over some ideas from sorting algorithms to find a new scheme that needs only logarithmically many back and forth movements of the engine and that works even if there are no additional tracks. The most appropriate approach is radix sort (see for example [2]) . It comes out that the number of back and forth movements of the engine through the hump is logarithmically bounded by the number of cars.

We are also interested in an appropriate parameter from which the number of back and forth movements can be determined as this can be significantly smaller than the logarithm of the number of cars. In case of merge sort, the number of merge steps is the logarithm of the number of colors of the associated permutation graph [5]. In case of radix sort, an appropriate parameter is the number of "chains", i.e. the number of tracks that are needed if we would like to get all cars in the right position in one back and forth movement step. Contrary to merge sort, we have to assume that the numbers of items (final position numbers of cars) are $\{1, \ldots, n\}$ and *not any* set of numbers. One can call it *consecutive integers adaptive radix sort*. This kind of adaptive sorting procedure is not an interesting approach for sorting unknown items but an interesting one to put known items onto the right place.

Next we would like to mention that it is not necessary that all cars are in a fixed position. It might be sufficient that cars of a certain destination appear consecutively. We say that cars of the same destination or cars that should belong to the same train form a "block". Two blocks are either disjoint or one block is a subset of the other block. It might be possible that cars or subblocks

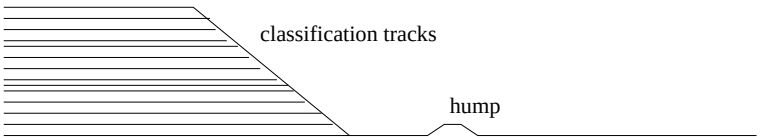

Figure 1: The Structure of a Hump Yard

of a certain block have to appear in a certain sequence (due to timetable restrictions) or not (the subblocks only have to appear consecutively). The partition of the car set into blocks and subblocks together with the requirements whether the subblocks have to appear in a certain sequence or not is called the *blocking scheme*. We are interested, given for each car its initial position and a blocking scheme, in finding an ordering of the cars that satisfies the blocking scheme, such that the number of back and forth movements, i.e. the number of chains, is minimized.

The general problem and even a certain special case is NP-complete [4]. We will discuss two cases that can be solved efficiently. One case is a generalization of the approach of [9]. A second efficient case can be reduced to a generalization of interval graph coloring.

2 The Yard Model and one Humping Step

The type of yard we would like to consider consists of, say k, parallel *classification tracks* that are joined by one long *humping track* on which there is a hump.

In general a *humping step* is defined as follows. Let S_i be the sequence of cars on track i. We select a final segment S_i' of S_i that has to be reordered, i.e. $S_i = S_i'' S_i'$. We concatenate the sequences S_i' to a sequence $T' = S_1', \ldots, S_k'$ (and draw the cars of T' behind the hump). For each car t of T', we select a track i_t, on which it has to be pushed, i.e. T' is split into k subsequences T_i' consisting of each car t with $i_t = i$. The new configuration consists therefore of the sequences $T_i = S_i'' T_i'$.

In our paper, we consider only humping steps where each S_i'' is empty. That means, in one humping step, we transform S into a sequence T as follows. We split S into subsequences $S_1', \ldots, S_k'$ and T is the concatenation of $S_1' \ldots S_k'$.

The goal is to transform an initial sequence $S_{in} = S_1 \ldots S_k$ into a final sequence $S_{fin} = T_1 \ldots T_k$. The *final position* of a car c is i if c is the ith element of S_{fin}, and the *initial* position of c is the j, such that c is the jth element of S_{in}. The final position of c is denoted by i_c and the initial position of c is denoted by j_c.

Throughout the whole paper, we call the ith car in the final position simply the ith car or car i

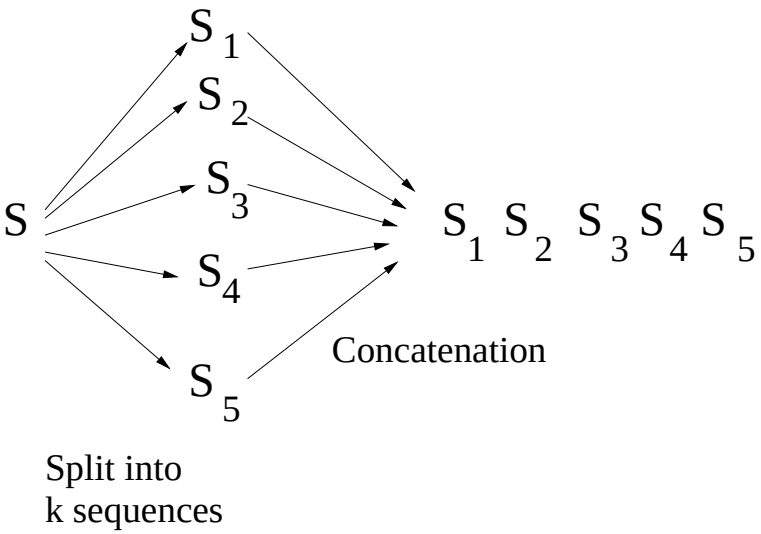

Figure 2: One Sorting Step

3 Logarithmic Sorting Scheme

3.1 Basic Idea

The basic idea is *radix sort* (see for example [2]). Although the scheme is quite well known, we will give a description to explain the idea of the adaptive scheme. We assume that the storage yard has k classification tracks. We also assume that the length of the tracks is not bounded.

We denote by p_i^ν the position of the ith car in step ν. We take care that after the νth step, the sequences $P_\mu^\nu = (p_i^\nu : i = k^\nu(\mu-1)+1, \ldots, k^\nu\mu)$ are increasing (and of length k^ν), i.e. the current positions of cars $k^\nu(\mu-1)+1, \ldots, k^\nu\mu)$ form an increasing sequence. Note that after $\log_k n$ many steps, there is only one sequence P_μ^ν that is increasing and all cars are therefore in final position.

One step of transformation can be done as follows. We assume that all $P_\mu^{\nu-1} = (p_i | i = k^{\nu-1}(\mu-1)+1, \ldots, k^{\nu-1}\mu)$ are increasing sequences (of length $k^{\nu-1}$), i.e. the actual positions of cars $k^{\nu-1}(\mu-1)+1, \ldots, k^{\nu-1}\mu)$ are increasing. If $\mu = (\mu'-1)k + l$, $l = 1, \ldots, k$ (i.e. $l = \mu \, mod \, k$), then we put cars of the sequence $P_\mu^{\nu-1}$ into track l and therefore, with $l_1 < k$, $\mu_1 = (\mu'-1)k + l_1$, and $\mu_2 = (\mu'-1)k + l_2$, the cars of $P_{\mu_1}^{\nu-1}$ have smaller positions than the cars of $P_{\mu_2}^{\nu-1}$. The sequences $P_{\mu'}^\nu$, i.e. the positions of the cars of final position $k^\nu(\mu'-1)+1, \ldots, k^\nu\mu')$ become increasing.

3.2 An Adaptive Radix Sorting Scheme

We again consider the sequence $P = (p_i | i = 1, \ldots n)$ of the positions of the cars (p_i is the actual position of car i). We divide the sequence P into maximal

increasing subsequences $(p_i, \ldots, p_j)$, we also call *chains*. We assume that P is the concatenation of the chains $P_1, \ldots, P_N$. If $\mu = \mu' k + l$ with $l = 1, \ldots, k$, we put cars belonging to chain P_μ into track l. The concatenation of the chains $P_{\mu'+l}$, $l = 1, \ldots, k$, becomes an increasing sequence, i.e. it becomes a chain. Therefore the number of chains is reduced by a factor of k in one step.

Theorem 1 *Let N be the number of chains of P. Then it is possible to transform the cars of P into their final positions in $\log_k N$ steps.*

This means the number of chains N is an important parameter of "presortedness".

It is worth mentioning that our adaptive radix sorting scheme is significantly different from the adaptive scheme in [6]. In our scheme it is essential that the numbers of the items form an initial segment of the natural numbers. The problem thus becomes to put the items into the right place in as few steps as possible and not to find a sorting.

We did not yet discuss how to behave if the lengths of tracks are bounded. In the final version of this paper we will demonstrate how to proceed in case of bounded track lengths and show that we still get a logarithmic bound on the number of chains.

4 Requirement Specification by Block Schemes (P-Q Trees)

Here we discuss the question "How should the order requirement be specified?". It is not necessary that the cars are rearranged into a fixed order. The ordering has only to satisfy that cars of the same outbound train have to appear consecutively or that cars of the same destination have to appear consecutively. We now discuss in more detail how to specify order requirements.

4.1 Order Requirement Specification by Trees

In general, the set of cars is divided into *blocks*, and each block might be recursively divided into subblocks. (Blocks subblocks correspond to outbound trains or or outbound trains in the next classification yard or destinations). The immediate subblocks of a block have to appear in a fixed sequence (e.g. due to time table restrictions) or they can be permuted. Essentially cars of the same block have to appear consecutively.

That means that we can specify an ordering requirement by a tree structure T, also called block tree.

1. The leaves are the cars.

2. The inner nodes correspond to the blocks, i.e. the block b_t corresponding to the node t is the the set of cars that are descendents of t in T.

3. A node t is marked as a P-node if the children of t (immediate subblocks of t) may appear in any order.

4. A node t is marked as a Q-node if the children of t have to appear in a fixed sequence.

In general, one can show the following.

Theorem 2 *Minimizing the number of chains is NP-hard.*

[4] have shown that minimizing the number of chains is NP-complete if the given P-Q-tree consists of a P-node as root that has only P-nodes as children and their children are leaves.

5 Efficient Special Cases for Chain Minimization

5.1 The Case of a Q-node with P-Children and Generalizations

We are given a Q-node q with P-node children $p_1, \ldots, p_k$. For each P-node p_i, let $w_{i,j}$ be the j^{th} child of p_i where the children of p_i appear in the same order as in the initial ordering of the cars. Let n_i be the number of children of p_i. An algorithm that determines a final ordering with a minimum number of chains works as follows.

Througout the algorithm r is the initial position of the last car that has been considered, c is the number of known chains, and f is the last final position that has been considered. We identify the car $w_{i,j}$ with its initial position.

1. Initially, $r = 0$, $f = 0$, and $c = 1$.

2. For $i = 1, \ldots, k$, we proceed as follows.

   - (Determine the first car that is a child of p_i) We determine the smallest j, such that the initial $w_{i,j} > r$. If such a $w_{i,j}$ does not exist, we increase c by one and we select $w_{i,j}$ with $j = 1$ start with the first car).
   - (Determine the final positions of the children of p_i) Let j be defined as in previous step. The final position $f_{i,\nu}$ of car $w_{i,\nu}$ is $f + \nu - j + 1$, for $\nu = j, \ldots, n_i$ and $f + n_i - j + 1 + \nu$, for $\nu = 1, \ldots, j - 1$.
   - (Update c, r, and f) If $j = 1$ then $r := w_{i,n_i}$ and c does not change. If $j \neq 1$ then $r := w_{i,j-1}$ and c is increased by one.
     f is updated by $f + n_i$.

Theorem 3 *The number of chains c computed by this algorithm is minimum. The number r is the minimum possible initial position of the last car of the last chain of an ordering compatible with the given P-Q-tree T with a Q-node root and P-node children.*

5.1.1 Relaxed Q-Nodes

Consider an enumeration $q_1, \ldots, q_k$ of the children of q. The enumeration $q_1, \ldots, q_k$ satisfies the *l-relaxed* Q-node condition if with $q_i = p_j$, $|j - i| \leq l$. An ordering of the cars (leaves) of the P-Q-tree T with a Q-node q with P-node children $p_1, \ldots, p_k$ satisfies the l-relaxed Q-node condition with respect to T if

1. the children of each p_i appear consecutively (that means the ordering of the cars induces an ordering of the children of q in a natural way).
2. the ordering of the children of q induced by the ordering of the cars satisfies the l-relaxed Q-node condition.

One can interpret this model in the sense that there is some tolerance as to when each train can depart. For example let $l = 1$. Train i can depart before train $i + 1$, but not before train $i + 2$. Zhu and Zhu [9] have shown how to minimize the number of chains for final orderings that satisfy the 1-relaxed Q-node condition, provided a Q-node q with P-node children is given and the P-nodes have only leaves as children. They developed an algorithm with a time bound of n^2 where n is the number of cars.

We get an extended result.

Theorem 4 *Let l be fixed. Given a P-Q-tree T with a Q-node q as root, such that all children of q are P-nodes and all grandchildren of q are leaves, and given any initial ordering $v_1, \ldots, v_n$ of the cars (leaves) of T, an ordering $w_1, \ldots, w_n$ that satisfies the l-relaxed Q-node condition with respect to T and that has a minimum number of chains can be determined in linear time.*

Sketch of Proof: Consider any enumeration $q_1, \ldots, q_k$ of the children of q that satisfies the l-relaxed Q-node condition. Then $\{q_1, \ldots, q_i\} = \{q_1, \ldots, q_{i-l}\} \cup S_i$ where

1. S_i is an l size subset of $\{q_{i-l+1}, \ldots, q_{i+1}\}$ if $i = l, \ldots, k - l$
2. S_i is an i size subset of $\{q_1, \ldots, q_{i+l}\}$ if $i < l$ and
3. S_i is a $k - i + 1$ size subset of $\{q_{i-l+1}, \ldots, q_k\}$ if $i > k - l$.

The sets S_i are called i-states. We consider the *state graph* defined as follows.

1. the nodes are the i-states, $i = 1, \ldots, k$,
2. there is a directed edge from an i-state S_i to an $(i + 1)$-state S_{i+1} if and only if $S_{i+1} = S_i \cup \{q_j\} \setminus \{q_{i-l+1}\}$.

Lemma 1 *The directed paths of the state graph from any 1-state S_1 to the k-state S_k and the sequences $q_1, \ldots, q_k$ meeting the l-relaxed Q-node condition correspond each other.*

An optimal solution can be obtained in a similar way as single source shortest path.

End of sketch of proof (Theorem)

5.2 The Case of a P-node with Q-Children

Recall that given the initial and the final positions of the cars of a set C, the permutation π maps the final position i_c of each car c into its initial position $j_c = \pi(i_c)$. Apart from minimizing the number of chains, we also can minimize the number of *jumps*, i.e. consecutive places $(i, i+1)$, such that $\pi(i) > \pi(i+1)$.

Now we are given a P-node p with Q-node children $q_1, \ldots, q_k$. The children of each q_i are leaves (or single cars). The children of each q_i are some $r_{i,1}, \ldots, r_{i,n_i}$. Note that the final position $i_{r_{i,j+1}} = i_{r_{i,j}} + 1$. Therefore the number of *jumps* in q_i, i.e. the number of jumps $(i_{r_{i,j}}, i_{r_{i,j+1}})$ is independent of the sequence $q_1, \ldots, q_k$ of the children of q that we choose. Only the number of jumps $(i_{r_{i,n_i}}, i_{r_{i+1,1}})$ might be dependent on the particular sequence $q_1, \ldots, q_k$. We call these jumps *external jumps*. They take only the initial position of the first car of a Q-node and the initial position of the last car of another Q-node into account.

The minimization of the number of external jumps is therefore equivalent to the following problem.

Sequencing of Ordered Pairs

Input: A set P of ordered pairs of natural numbers (in the special case, for each q_i, the initial position of the first and of the last car that is a child of q_i)

Output: A partition of P into sequences $P_1, \ldots, P_k$, such that

1. each P_i is *locally increasing*, i.e. if (x, y) and (z, w) are consecutive in P_i then $y < z$,
2. each (x, y) appears in exactly one sequence P_i, and
3. the number k of sequences P_i is minimum.

Note that if for all $(x, y) \in P$, $x < y$ then this problem is equivalent to interval graph coloring.

Theorem 5 *Sequencing of Ordered Pairs can be solved in linear time.*

Sketch of Proof: First we discuss a lower bound for the number of sequences $P_1, \ldots, P_k$, we also call *color sequences*. We proceed in a similar way as in interval graphs [5].

We call a pair (x, y) *positive* if $x < y$ and *negative* if $x > y$.

For any real number r, Pos_r is the number of positive $(x, y) \in P$ with $x \leq r \leq y$ and Neg_r is the number of negative $(x, y) \in P$ with $y < r < x$. The *clique size* of r in P is $C_r^P := Pos_r - Neg_r$.

Lemma 2 *Let $P_1, \ldots, P_k$ be a partition of P into color sequences. Then $k \geq C_r^P$, for any real number r.*

The *clique number* $\omega(P)$ of P is the maximum clique size of a real number r in P.

Lemma 3 *There is an efficient algorithm that computes a partition of P into color sequences $P_1, \ldots, P_k$ with $k \leq \omega(P) + 1$.*

Proof: We proceed in a similar fashion to the interval graph coloring. We sort the numbers x and y appearing in some $(x, y) \in P$ in increasing order.

Now we may assume that the numbers appearing in P are $1, \ldots, n$.

For each color sequence $cs = d_1, \ldots, d_q$, the *first element* is the first component of d_1 and the *last element* is the second component of d_q.

Initially the set of color sequences consists of the one element sequences of the elements of P.

For each $i = 1, \ldots, n$, we concatenate color sequences with last element $< i$ with color sequences with first element $= i$ as much as possible. As much as possible, we avoid concatenating a color sequence with itself. If a color sequence concatenates with itself, we declare it as "cyclic" (meaning that any cyclic permutation of the sequence is a color sequence).

We finally have $\omega(P)$ non cyclic color sequences and some cyclic color sequences. It is possible to concatenate the cyclic color sequences to one sequence.

Q.E.D.

The number of color sequences we just computed is at most one more than the minimum number of color sequences. To get the minimum, we still keep track of the remaining cyclic color sequences and integrate them into the non cyclic color sequences as long as possible, i.e. we have a cyclic color sequence ccs and a noncyclic color sequence cs' and create a color sequence $cs'_1 ccs' cs'_2$ where cs'_1 is an initial segment of cs', ccs' is a cyclic permutation of ccs, and cs'_2 consists of the remaining elements of cs'.

We will show in the full paper, how to determine the minimum number of color sequences in linear time, provided we sorted the numbers appearing in P in advance, i.e. the appearing numbers are $1, \ldots, n$.

End of sketch of proof (Theorem)

6 Conclusions

First it should be mentioned that for the case of a P-node root with P-node children and only leaves as grandchildren, one can get a 2-approximate solution applying interval graph coloring. In general, one gets a more complicated algorithm that approximates the number of jumps up to a constant factor. This will be presented in the final version of the paper. Here we have minimized the number of essential sorting steps that can be considered as a parameter to minimize the energy of the engine (the number of passing of the hump times two). Another problem is to try to minimize the number of couplings and decouplings of cars. In the European network, coupling and decoupling of cars is very time consuming, because the old fashioned couplers of last century are still in use. Automatic coupling has not been introduced yet.

References

[1] K. Booth, G. Lueker, *Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using PQ-Tree Algorithms*, Journal of Computer and Systems Sciences 13(1976), S. 335-379.

[2] T. Cormen, C. leiserson, R. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge MA, McGraw Hill Book Company, New York.

[3] C. Daganzo, *Static Blocking at Railyards: Sorting Implications and Track Requirements*, Transportation Science 20 (1986), pp. 189-199.

[4] E. Dahlhaus, P.Horak, M. Miller, J. Ryan, *The Train Marshalling problem*, accepted for Discrete Applied Mathematics.

[5] M. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.

[6] S. Nilsson, *Radix Sorting and Searching*, Ph.D. Thesis, Department of Computer Science, Lund University.

[7] E. Petersen, *Railyard Modeling: Part II, The Effect of Yard Facilities on Congestion*, Transportation Science 11 (1977), pp. 51-59.

[8] M. Sidiqee, *Investigation of Sorting and Train Formation Schemes for a Railroad Hump Yard*, in *Traffic Flow and Transportation*, G. Newell ed. (1972), pp. 377-388.

[9] Zhu, Y. and Zhu, R, Sequence reconstruction under some order-type constraints, Scientia Sinica, Series A, Vol. 26, No.7, pp.702-713, 1983.