Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

ARMIN FÜGENSCHUH[1]
KONSTANTY JUNOSZA-SZANIAWSKI[2]
TORSTEN KLUG[3]
SŁAWOMIR KWASIBORSKI[2]
THOMAS SCHLECHTE[3]

Fastest, average and quantile schedule

[1] Helmut Schmidt University / University of the Federal Armed Forces Hamburg, Holstenhofweg 85, 22043 Hamburg, Germany fuegenschuh@hsu-hh.de
[2] Politechnika Warszawska, Matematyki ul. Koszykowa 75, 00-662 Warszawa, Poland {k.szaniawski,kwasiborskis}@mini.pw.edu.pl
[3] Zuse Institute Berlin, Takustraße 7, 14195 Berlin, Germany {klug,schlechte}@zib.de

ZIB Report 14-49 (Oktober 2014)

Herausgegeben vom
Konrad-Zuse-Zentrum für Informationstechnik Berlin
Takustraße 7
D-14195 Berlin-Dahlem

Telefon: 030-84185-0
Telefax: 030-84185-125

e-mail: bibliothek@zib.de
URL: http://www.zib.de

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

Fastest, average and quantile schedule

Armin Fügenschuh[1], Konstanty Junosza-Szaniawski[2], Torsten Klug[3], Sławomir Kwasiborski[2], and Thomas Schlechte[3]

[1] Helmut Schmidt University / University of the Federal Armed Forces Hamburg, Holstenhofweg 85, 22043 Hamburg, Germany, http://am.hsu-hh.de

[2] Politechnika Warszawska, Matematyki ul. Koszykowa 75, 00-662 Warszawa, http://mini.pw.edu.pl

[3] Department of Optimization, Zuse Institute Berlin, Takustraße 7, 14195 Berlin, Germany, http://www.zib.de/

Abstract. We consider problems concerning the scheduling of a set of trains on a single track. For every pair of trains there is a minimum headway, which every train must wait before it enters the track after another train. The speed of each train is also given. Hence for every schedule - a sequence of trains - we may compute the time that is at least needed for all trains to travel along the track in the given order. We give the solution to three problems: the fastest schedule, the average schedule, and the problem of quantile schedules. The last problem is a question about the smallest upper bound on the time of a given fraction of all possible schedules. We show how these problems are related to the travelling salesman problem. We prove NP-completeness of the fastest schedule problem, NP-hardness of quantile of schedules problem, and polynomiality of the average schedule problem. We also describe some algorithms for all three problems. In the solution of the quantile problem we give an algorithm, based on a reverse search method, generating with polynomial delay all Eulerian multigraphs with the given degree sequence and a bound on the number of such multigraphs. A better bound is left as an open question.
Keywords: Schedule, generating permutations with repetitions, Eulerian multigraphs.

1 Introduction

In the theory of combinatorial algorithms typically the following problems are considered: find any feasible solution, find one feasible solution, find an optimal solution, enumerate all solutions (with minimum weight), count all solutions (with a given weight). We ask a natural follow-up question: what is a quantile of the given fraction of feasible solutions, i.e., what is the minimum number a such that the weights of all feasible solutions of a given fraction are not exceeding a. For example for the travelling salesman problem the question about the 0.8-quantile is: what is the smallest number a such that 80% of all travelling salesman tours for a set of given cities have a weight of at most a.

Problems considered in this paper originate from railway track allocation in a real world application [6].

Let us consider the strategic routing of freight trains in a highly utilized network. Given a railway network that is utilized by a set of passenger trains and a model day that is partitioned into a few time slices. Each time slice represents a special traffic situation of the day and comprised several hours, for instances the morning and afternoon peak of a working day or the night with a rather small amount of passenger traffic. We classify the trains into different train

types, which describe the characteristic properties of the trains,e.g., running times, headway times or special requirements for the track. The preset passenger traffic for each slice is simply described by the number of trains of a specific train type. In particular there is no information of the actual schedule of the trains. We only know that this trains used the track within the time slice. On the demand side the freight trains are defined by an origin destination pair, the departure time and its train type. The task is to find a route for each freight train that does not exceed a given distance and running time limit and minimize the expected delays. Since we are only interested in a strategic routing with a rough approximated timing, the minimal expected delays should ensure the existence of a feasible timetable or at least increase the possibility for one.

At this point the topic of the paper pops up. We need for each track an estimation of the expected delays. This could be done by estimating the possible schedules for each slice, track and train set. In our case we have a fixed set of passenger trains and a huge amount of possible freight train sets. Let us denote by a configuration a vector that contains a number of trains for each freight train type. Now we are looking for an assignment of configurations to tracks and slices such that a feasible routing for the requested demand exists. Since we can assume that the cost of a schedule increases with the number of trains, it is possible to assign a configuration with much more trains in it than in the end are routed over the specific track. We could interpret an assigned configuration as capacity. Therefore a solving procedure can start with a subset of the possible configurations and can generate on demand new smaller configurations with better cost and tighter capacity to improve the overall solution. In particular, it is usefull within a column generation approach to solve a mixed integer program formulation of the problem. Finally we come to the following problem that had to be solved as sub-problem several times.

We consider a single railway track (from station A to B) and a set of trains, with their speeds and minimal headway between every pair of trains. For any given sequence of trains we can compute the least time that is needed for all trains of the sequence to arrive at B.

Example 1 *Let us consider the following trivial example. We are given two train types with running time 3 and 5 and the headways are given in the matrix* $\begin{pmatrix} 1 & 1 \\ 3 & 1 \end{pmatrix}$. *Figure 1 shows three potential orderings of 4 trains (2 of each type).*

A natural question is which sequence gives the minimal time. Under some natural conditions with respect to the speeds of the trains and the minimal headways we prove that a sequence of trains ordered with non-increasing speed is fastest. However in the general case the problem of the fastest schedule is NP-complete. For the general case we give an algorithm for finding the fastest schedule, based on dynamic programming. Moreover we give an explicit formula for the average time taken over all possible schedules. This problem is equivalent to the problem of determining the average weight of all Hamilton cycles.

The last question and most interesting from both a practical and theoretical point of view is the question about the quantile of the schedule, i.e., what is the minimum number a such that the weights of all schedules of a given fraction are not exceeding a. For example, if a 0.8-quantile is equal to a, then 80% of all schedules can be realized in time not exceeding a. To solve the problem we take advantage of the fact that the speeds of trains and the minimal headways

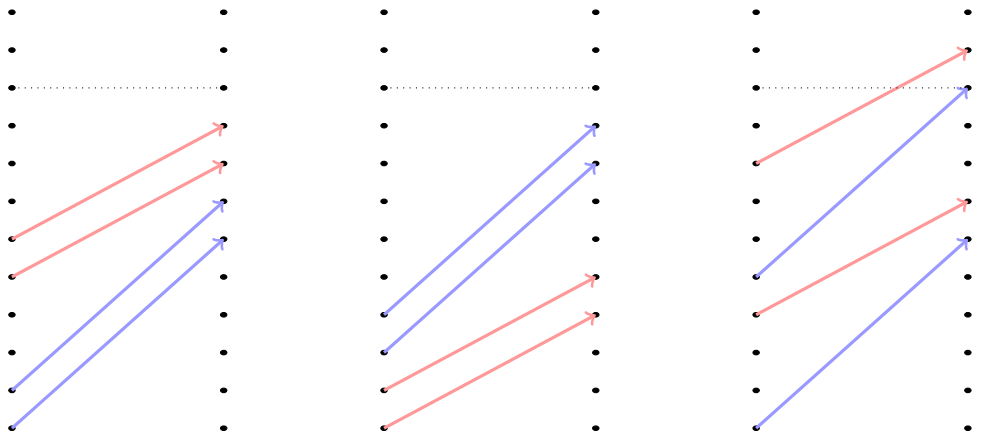

Fig. 1: Example for schedules of two slow and two fast trains and their times.

depend only on the type of trains. In addition, there is only a small number of types compared to the number of trains. The simplest way is to generate all sequences of trains or sequences of types of trains, and to compute the quantile directly. We solve the problem in a more sophisticated way. First we define an equivalence relation on the set of schedules such that any two schedules in the relation have the same time for finishing. Then we generate all equivalence classes and compute the time for finishing for every class. The equivalence classes of our relation directly correspond to Eulerian multigraphs with a given degree sequence. To generate the multigraphs we use the reverse search method introduced by Avis and Fukuda [2]. We prove that there are $O(n^{k^2-1})$ Eulerian multigraphs on k vertices with n edges. This bound is not tight. Any better bound on this number would give a better complexity bound of our algorithm since we generate Eulerian multigraphs with polynomial delay.

2 Preliminaries and Problem Formulation

For $n \in \mathbb{N}$ we denote $\{1 \dots n\}$ by $[n]$ and by $[n]^*$ the set of all finite sequences of elements of the set $[n]$. Let $X = \{x_1, x_2, \dots, x_n\}$ be the set of trains. Let the function $t\colon X \to [k]$ assign every train its type. We assume that the set X and the function t are fixed in the following. Let $l\colon [k] \to \mathbb{N}$ be a function defined by $l(i) = |t^{-1}(i)|$. Thus $l_i = l(i)$ denotes the number of trains of the i-th type. Let $r\colon [k] \to \mathbb{R}_+$ assign the running time to every train type and let $m\colon [k] \times [k] \to \mathbb{R}_+$ be the function determining the minimal headway between the trains of certain types.
Let $S(X)$ denote a set of all permutations of the set X. We will call the permutations of X schedules. The minimal running time of the schedule $y \in S(X)$ is computed as

$$RT(y) = \sum_{i=1}^{n-1} m(t(y_i), t(y_{i+1})) + r(t(y_n)).$$

We define the following three problems:

Problem 1 *Fastest-schedule*
Input: (X, t, r, m, k).
Output: YES if and only if there exists a schedule $s \in S(X)$ such that $RT(s) \leq k$.

Problem 2 *Average-schedule*
Input: (X, t, r, m).
Output: $\bar{\tau} \in \mathbb{R}$ - average running time of a schedule, i.e.,

$$\bar{\tau} = \frac{\sum_{y \in S(X)} RT(y)}{|S(X)|}.$$

Problem 3 α*-quantile schedule*
Input: (X, t, r, m, α^1).
Output: τ - time needed to realize $\alpha|S(X)|$ schedules, i.e.,

$$\tau = \min\{rt : \frac{|\{s \in S(X) : RT(s) \leq rt\}|}{|S(X)|} \geq \alpha\}.$$

3 Fastest schedule problem

Theorem 2 *Fastest-schedule is NP-complete regarding to number of train types.*

We can solve the fastest schedule problem by interpreting it as a slight modification of the TSP. The modification will state that a city from the TSP setting must be visited a given number of times. It is allowed to visit the same city multiple times in a row but there is a non zero "distance" assigned to such operation. As cities we will denote members of set $[k] \cup 0$. Distances are given by following function:

$$d(x, y) = \begin{cases} m(x, y), & x \neq 0 \wedge y \neq 0, \\ r(x), & x \neq 0 \wedge y = 0, \\ 0, & x = 0 \wedge y \neq 0. \end{cases}$$

The number of times that the city denoted by $x \in [k]$ must be visited is given by the function $l(x)$ as defined in the preliminaries. The city with number 0 must be visited exactly once. Notice that every tour in this graph corresponds to a number of schedules (the tour determines the order of train types so actual trains of the same type can be permuted). The tour length is equal to the running time of the corresponding schedules.
Such modification of the TSP can be solved using a modification of the classic dynamic programming algorithm given by Bellman [3]. Let $\mu \colon \mathbb{N} \times [k] \to \mathbb{N}$ be a function. By $\mu(t, (x_1 \dots x_k))$ we will denote the minimal length of tour from 0 to t passing through each city $i \in [k]$ exactly x_i times (starting and ending visits are not counted). We can define μ recursively as:

$$\mu(t, (x_1, \dots, x_k)) = \min_{i \in [k] \wedge x_i \neq 0} \{\mu(i, (x_1, \dots x_i - 1, \dots, x_k)) + w(i, t)\}.$$

[1] with α - a given fraction of the schedules that have to be realizable (1 means that all schedules must be realizable, 0.5 means that half of the potential schedules must be realizable)

The iterative procedure can be initiated for all i by:

$$\mu(i,(0,\ldots,0)) = w(0,i),$$

from which we can obtain the respective next values using the recursive formula. Value of $\mu(0,(l(1),\ldots,l(k)))$ gives a solution to the Fastest-schedule problem. The computational complexity of this algorithm can be calculated by estimating the number of different parameter sets of function μ that must be calculated and the time for computing a single value. The first parameter can be picked in k ways and the second parameter is the number of solutions of the inequality: $x_1 + \ldots + x_k \leq n$. The number of solutions of this inequality can be estimated by n^k. Each value of the function μ can be computed in linear time. From above we conclude that the algorithm runs in time $O(n^{k+1})$.
The most natural candidate for an optimal solution is a schedule with trains that are ordered non-decreasing by their running time. This simple solution seems to work in real world scenario, but it can be shown that it is not correct in the general case. A question rises what conditions have to be fulfilled for this simple solution to be correct.

Theorem 3 *Let (X,t,r,m) be an instance of the fastest-schedule problem. If we assume that:*

$$\begin{aligned}\forall_{x_1,x_2,x_3,x_4\in X}\ & r(t(x_1)) \leq r(t(x_2)) \wedge r(t(x_1)) \leq r(t(x_3)) \wedge r(t(x_1)) \leq r(t(x_4)) \\ & \Rightarrow m(t(x_1),t(x_2)) + m(t(x_3),t(x_4)) \leq m(t(x_3),t(x_1)) + m(t(x_1),t(x_4)) \qquad (1)\end{aligned}$$

and

$$\begin{aligned}\forall_{x_1,x_2,x_3,x_4,x_5\in X}\ & r(t(x_1)) \leq r(t(x_2)) \leq r(t(x_3)) \wedge r(t(x_2)) \leq r(t(x_4)) \wedge r(t(x_2)) \leq r(t(x_5)) \\ & \Rightarrow m(t(x_1),t(x_2)) + m(t(x_2),t(x_3)) + m(t(x_4),t(x_5)) \\ & \leq m(t(x_1),t(x_3)) + m(t(x_4),t(x_2)) + m(t(x_2),t(x_5)) \qquad (2)\end{aligned}$$

then the schedule consisting of trains ordered not-decreasing by running time, is the solution to fastest schedule problem.

4 Average schedule problem

Besides the question for an optimal solution, let it be minimum or maximum, finding the running time of an *average* schedule could be of interest. This problem can be solved in polynomial time. First we reduce the problem of the average schedule to the problem of the average Hamilton cycle length in a complete graph. Let $X = \{1,\ldots,n\}$ be a set of trains. For each pair of trains $i,j \in V$ with $i \neq j, m(t(i),t(j))$ determines minimal headway between i and j. Let $V = X \cup \{0\}$ be the vertex set of the graph, $A = [n] \times [n] \setminus \{(i,i) : i \in [n]\}$ be the arc set and let weight function be given by:

$$w(i,j) = \begin{cases} m(t(i),t(j)), & i \neq 0 \wedge j \neq 0 \\ r(t(i)), & i \neq 0 \wedge j = 0 \\ 0, & i = 0 \wedge j \neq 0. \end{cases}$$

Observe that the tour in this graph corresponds to exactly one schedule and that the tour length is equal to this schedule running time. Hence the average time of all schedules is equal to the average Hamilton cycle length.

Theorem 4 *Let $G = (V, E, w)$ be a weighted undirected complete graph. Then the average tour length for a Hamiltonian cycle in G is $\frac{2}{n-1} \sum_{e \in E} w_e$.*

Proof. Since G is a complete graph, each edge is contained in exactly $(n-2)!$ Hamiltonian cycles. There are in total $\frac{n!}{2n}$ Hamiltonian cycles in a complete graph. Therefore the average weight of all Hamiltonian cycles is

$$\frac{2n}{n!} \cdot \sum_{e \in E} (n-2)! \cdot w_e = \frac{2}{n-1} \sum_{e \in E} w_e. \tag{3}$$

□

Corollary 5 *Let $D = (V, A, w)$ be a weighted directed complete graph. Then the average tour length for a Hamiltonian cycle in D is $\frac{1}{n-1} \sum_{a \in A} w_a$.*

Proof. There are in total $\frac{n!}{n}$ Hamiltonian cycles in a complete directed graph. □

Corollary 6 *Let $G = (V, E, w)$ be a weighted undirected complete graph. Then the average length of a Hamiltonian path in G is $\frac{2}{n} \sum_{e \in E} w_e$.*

Corollary 7 *Let $D = (V, A, w)$ be a weighted directed complete graph. Then the average length of a Hamiltonian path in D is $\frac{1}{n} \sum_{(i,j) \in A} w(i, j)$.*

Proof. Let $V' := V \cup \{0\}$ as above and $A' := A \cup \{(0, i) : i \in V\} \cup \{(i, 0) : i \in V\}$. Let $w'_a := w_a$ for $(i, j) \in A$, $w'_a := 0$ for $a \in A' \backslash A$. Again, there is a bijection between Hamiltonian cycles in $D' := (V', A', w')$ and Hamiltonian paths in D, from which the formula follows. □

We can conclude that average running time of a schedule for given (X, t, r, m) equals to $\frac{1}{n} \sum_{(i,j) \in A} w(i, j)$, where A and $w(i, j)$ are defined as above.

5 Schedules Quantiles

The problem "quantile schedule" is at least as hard as the fastest schedule. For $\alpha = \frac{1}{n!}$, where n is the number of trains, a solution of α-quantile is also a solution of the problem Fastest-schedule.

We need some more notations. For $b, k \in \mathbb{N}$, $s = (s_1, \ldots, s_b) \in [k]^*, p, q \in [k]$, by $\Delta(s, p, q)$ we will denote $\{i \in [b-1] : p = t(s_i), q = t(s_{i+1})\}$ which is a set of all the indices on which a train type changes from p to q in the sequence s. By $\delta(s, p, q)$ we will denote $|\Delta(s, p, q)|$.

Let $\sim \subset S(X) \times S(X)$ be a relation defined on permutations of the set of the trains.

We say that:

$$y \sim z :\Leftrightarrow \forall_{p,q \in [k]}\ \delta(y, p, q) = \delta(z, p, q).$$

Lemma 8 *For any $y, z \in S(X)$ if $y \sim z$ then $t(y_n) = t(z_n)$ and $t(y_1) = t(z_1)$.*

Proof. Notice that the sum $\sum_{q \in [k]} \delta(y, t(y_n), q)$ is equal to the number of occurrences of a train of the type $t(y_n)$ in the schedule y on positions from 1 to $n-1$. Hence

$$\sum_{q \in [k]} \delta(y, t(y_n), q) = l_{t(y_n)} - 1.$$

By the definition of the relation $\sim$ we get:

$$\sum_{q\in[k]} \delta(z, t(y_n), q) = \sum_{q\in[k]} \delta(y, t(y_n), q) = l_{t(y_n)} - 1.$$

So the trains of the type $t(y_n)$ occur $l_{t(y_n)} - 1$ times on positions from 1 to $n-1$ in the sequence z, but trains of the type $t(y_n)$ occur $l_{t(y_n)}$ times (on positions from 1 to n) in the sequence z, hence $t(z_n) = t(y_n)$. The proof of $t(y_1) = t(z_1)$ is analogue. □

Theorem 9 *For any $y, z \in S(X)$ if $y \sim z$, then $RT(y) = RT(z)$.*

Proof. We can observe that

$$RT(y) = \left(\sum_{i=1}^{n-1} m(t(y_i), t(y_{i+1}))\right) + r(t(y_n)) \qquad = \left(\sum_{p,q\in[k]} m(p,q)\delta(y,p,q)\right) + r(t(y_n))$$

From Lemma 8 and $y \sim z$, $t(y_n) = t(z_n)$, we obtain $r(t(y_n)) = r(t(z_n))$. From the definition of $\sim$ we have that

$$\forall_{p,q\in[k]}\ \delta(y,p,q) = \delta(z,p,q)$$

from above:

$$\begin{aligned} RT(y) &= \left(\sum_{p,q\in[k]} m(p,q)\delta(y,p,q)\right) + r(t(y_n)) \\ &= \left(\sum_{p,q\in[k]} m(p,q)\delta(z,p,q)\right) + r(t(z_n)) = RT(z). \end{aligned}$$

□

Theorem 10 *There exist functions m, r such that if $y, z \in S(X)$ and $y \not\sim z$, then $RT(y) \neq RT(z)$.*

Let us denote the equivalence classes of the relation $\sim$ by $[s]_\sim$. By $\delta([s]_\sim, p, q)$ we will denote the value of $\delta(y, p, q)$ for any $y \in [s]_\sim$. This notation is well-defined since from the definition of relation $\sim$ for any $y \in [s]_\sim$ it holds that $\forall p, q \in [k]\ \delta(s, p, q) = \delta(y, p, q)$.

By a block of the trains of the type i we denote a sequence of consecutive trains of type i such that a train directly before and after the block are of any type not equal to i. Given $s \in S(X)$, by $b_s(i)$ we denote number of blocks of the trains of the type i.
We define a function $R\colon S(X) \to [k]^*$ as follows: for $s \in S(X)$, $R(s)$ is a sequence obtained from s by replacing every block of trains of type i by single appearance of i. Notice that $R(s)$ is a sequence of length $\sum_{i\in[k]} b_s(i)$. Moreover notice that:

$$\delta(R(s), p, q) = \begin{cases} \delta(s,p,q), & p \neq q, \\ 0, & p = q. \end{cases}$$

It is easy to observe that if $R(y) = R(z)$ then $y \sim z$. Let $R([s]_\sim) = \{R(y) : y \in [s]_\sim\}$ and $R^{-1}(R(s)) = \{y \in S(X)\colon R(y) = R(s)\}$.

Lemma 11 *For any $s \in S(X)$*

$$|R^{-1}(R(s))| = \prod_{i=1}^{k} l(i)! \binom{l(i)-1}{b_s(i)-1}.$$

From above lemma directly follows:

Corollary 12 *For any $s \in S(X)$*

$$|[s]_\sim| = |R([s]_\sim)| \cdot \prod_{i=1}^{k} l(i)! \binom{l(i)-1}{b_s(i)-1}.$$

Hence to count the number of schedules in $[s]_\sim$, it is enough to count the number of sequences in $R([s]_\sim)$.
Let $G_{[s]_\sim} = (V_{[s]_\sim}, \mu_{[s]_\sim})$ be a directed multigraph where $\mu_{[s]_\sim} : V^2_{[s]_\sim} \to \mathbb{N}$ is function assigning to vertices p, q the number of arcs from p to q. The multigraph is constructed as follows: $V_{[s]_\sim} = [k] \cup \{0\}$, for all $p, q \in [k]$, $\mu_{[s]_\sim}(p, q) = \delta([s]_\sim, p, q)$, moreover $\mu_{[s]_\sim}(t(s_n), 0) = 1$ and $\mu_{[s]_\sim}(0, t(s_1)) = 1$. By Eulerian cycle in $G_{[s]_\sim}$ we mean a vertex sequence in $G_{[s]_\sim}$ containing every pair $(p, q) \in V^2_{[s]_\sim}$ as consecutive pair pq exactly $\mu_{[s]_\sim}(p, q)$ times. By $\widehat{G}_{[s]_\sim} = (V_{[s]_\sim}, \widehat{\mu}_{[s]_\sim})$ we denote the multigraph obtained from $G_{[s]_\sim}$ by deleting all loops (for each vertex $v \in V_{[s]_\sim}$ $\widehat{\mu}_{[s]_\sim}(v, v) = 0$).
Let $G = (V, \mu)$ be a multigraph, by $deg^-_G(i) = \sum_{v \in V} \mu(v, i)$ we denote the indegree of vertex i in graph G, and by $deg^+_G(i) = \sum_{v \in V} \mu(i, v)$ we denote out degree of vertex i in graph G. It can be noted that for all $i \in [k]$ it holds that $deg^-_G(i) = deg^+_G(i)$. Moreover it can by shown that $deg^-_G(i) = b_i$ for $i \in [k]$.
The following observation is the key to our algorithm.

Remark 13 *Every sequence $r \in R([s]_\sim)$ corresponds to one Euler vertex sequence in $\widehat{G}_{[s]_\sim}$.*

Remark 14 *Graph $\widehat{G}_{[s]_\sim}$ is connected for any $s \in S(X)$.*

For a multigraph $G = (V, \mu)$ we define the Kirchhoff matrix $K(G)$ as follows :

$$K(G)_{ij} = \begin{cases} deg^-_G(i), & \text{if} \quad i = j, \\ -\mu_G(i, j), & \text{if} \quad i \neq j. \end{cases}$$

For $i \in [n]$ we denote by K_i the matrix obtained from K by deleting the i-th row and the i-th column. By $\det(K)$ we denote determinant of matrix K. By $ec(G)$ we denote number of Euler cycles in G.

Theorem 15 (de Bruijn, van Aardenne-Ehrenfest, Smith, Tutte [1]) *Given a multigraph $G = (V, \mu)$ then the number of Eulerian cycles $ec(G)$ is given by:*

$$ec(G) = \frac{t_1(G) \prod_{v \in V} \left(deg^-_G(v) - 1\right)!}{\prod_{i,j \in [k], i \neq j} (\mu(i, j))!},$$

where $t_v(G)$ denotes number of trees rooted at vertex v.

Theorem 16 (Tutte Matrix Tree Theorem [5]) *Given a multigraph $G = (V, \mu)$ with Kirchhoff matrix $K(G)$, then the number of trees rooted at vertex v is equal to* $\det(K_v(G))$.

Theorem 17 *For any $s \in S(X)$ it holds that*

$$|[s]_\sim| = \frac{\det\left(K_1(\widehat{G}_{[s]_\sim})\right)\prod_{i=1}^{k}\left(deg^-_{\widehat{G}_{[s]_\sim}}(i)-1\right)!}{\prod_{i,j\in[k],i\neq j}(\delta(s,i,j))!}\cdot\prod_{i=1}^{k} l(i)!\binom{l(i)-1}{deg^-_{\widehat{G}_{[s]_\sim}}(i)-1}.$$

Proof. Follows directly from the Lemma 11 and the Theorems 15 and 16. □

6 Algorithm

Instead of enumerating all equivalence classes of relation $\sim$, we can enumerate all connected Eulerian multigraphs with given vertex degree sequence. To generate only connected graphs with desired properties, we use the reverse search method described in the next part of the paper. Every graph identifies one $\sim$ equivalence class therefore corresponding schedules have equal running times.
The algorithm generates all graphs. Then sorts them by running time of corresponding schedules. Then it finds a first equivalence class such that number of schedules in this class and in proceeding classes is at least α fraction of all schedules and returns its running time. The following algorithm solves the running time problem:

Algorithm 1 RunningTime(X, t, r, m, α)

```
Generate all graphs for X into S using reverse search.
Order the schedules in S by running time in ascending order
allSchedulesNumber = Σ_{s∈S} |[s]_∼|
τ = 0
currentNumberOfSchedules = 0
while currentNumberOfSchedules < α · allSchedulesNumber do
    s = S.Pop
    currentNumberOfSchedules+ = |[s]_∼|
    τ = RT(s)
end while
return τ
```

Functions RT and l are defined in terms of t, r, m as in former part of the paper.

Theorem 18 *The RunningTime algorithm returns a valid result.*

Proof. The validity of result follows from Theorem 17. □

Theorem 19 *There are at most $O(n^{k^2-1})$ connected multigraphs with given vertex degree sequence.*

From Theorem 19 we know that there are at most $O(n^{k^2-1})$ connected multigraphs with given vertex degree sequence. All operations conducted on single graphs take polynomial time Therefore the complexity of the whole algorithm is at most $O(n^{k^2-1})$.

7 Algorithm for generating connected graphs

To generate connected graphs efficiently, we can use a method of Avis and Fukuda called *Reverse Search* [2]. The main idea of this technique is to define the graph on the set of objects to generate and perform a search (e.g. breadth-first-search) on its spanning tree generating one object by visiting each vertex. To be precise: a triple $(\Gamma, \widehat{S}, f)$, where $\Gamma = (\mathcal{V}, \mathcal{E})$, $\widehat{S} \in \mathcal{V}$, f is a mapping $\mathcal{V} \setminus \{\widehat{S}\} \to \mathcal{V}$, is called *local search* if
(L1) $\{v, f(v)\} \in \mathcal{E}$ for each $v \in \mathcal{V} \setminus \{\widehat{S}\}$.
Local search $(\Gamma, \widehat{S}, f)$ is called *finite local search* if
(L2) for each $v \in \mathcal{V} \setminus \{\widehat{S}\}$ there exists a positive integer i such that $f^i(v) = \widehat{S}$.
The *trace* of local search $(\Gamma, \widehat{S}, f)$ is a directed sub-graph $T = (\mathcal{V}, \mathcal{E}(f))$ where $\mathcal{E}(f) = \{(v, f(v)) : v \in \mathcal{V} \setminus \{\widehat{S}\}\}$. T is simply the directed spanning tree of Γ, rooted in $\widehat{S}$, defined by f.

Let $(\Gamma, \widehat{S}, f)$ be a finite local search with trace T. As "abstract reverse search" we call a routine of traversing T and outputting all its vertices. The traversal can be implemented in any way. In this paper we will conduct the traversal by breadth first search starting from the sink and traversing all edges in a way opposite to their direction.

By $N_\mu(v) = \{u \in V : \mu(v, u) > 0\}$ we denote the neighbourhood of vertex v. By $C(V, \mu)$ we denote the number of connected components of graph $G = (V, \mu)$. For $\mu : V \times V \to \mathbb{N}$ such that $\mu(u, v) > 0$ we define $\mu - (u, v) = \mu'$ by

$$\mu'(p, q) = \begin{cases} \mu(u, v) - 1, & \text{for } (p, q) = (u, v), \\ \mu(p, q), & \text{for } (p, q) \neq (u, v). \end{cases}$$

For a graph $G = (V, \mu)$ a traversal from u to v we call a "bridge traversal" if and only if $C(V, \mu) < C(V, \mu - (u, v))$. By non-bridge neighbours of v we denote the set $NN_\mu(v) = \{u \in N_\mu(v) : (v, u)$ *is not a bridge traversal*$\}$.

For a $G = (V, \mu) \in \mathcal{G}_l$ by $\overrightarrow{G}$ we will denote a minimal Euler cycle for graph G - a cycle generated by the following algorithm:

Algorithm 2 MinimalEulerCycle(V, μ)

```
1: μ_F = μ, v = 0, u = 0
2: mec = "empty sequence" {minimal Euler cycle}
3: repeat
4:     mec += u {append to the end of the sequence}
5:     if NN_{μ_F}(v) ≠ ∅ then u = min(NN_{μ_F}(v))
6:     else u = min(N_{μ_F}(v)) end if
7:     μ_F = μ_F − (v, u)
8:     v = u,
9: until v = 0
10: return mec
```

Assuming that we use Tarjan's [7] algorithm for finding bridges then the time complexity of above algorithm is $O(|E|^2)$ where $|E| = \sum_{v \in V} deg^+_G(v)$ is the number of edges in the graph G.

The algorithm is a realization of Fleury's algorithm [4] for finding Euler cycle determining the order in which non-bridge and then bridge edges are traversed. The correctness of the algorithm follows directly from the correctness of Fleury's algorithm.

Let $w, x, y, z \in V$ such that $\mu(w,x), \mu(x,y), \mu(y,z) > 0$. By $t(G,(w,x,y,z)) = (V, \mu_t)$ we will denote a multigraph obtained from G by following modification of μ:

$$\mu_t(p,q) = \begin{cases} \mu(p,q) - 1, & \text{for } (p,q) \in \{(w,x),(x,y),(y,z)\}, \\ \mu(p,q) + 1, & \text{for } (p,q) \in \{(w,y),(y,x),(y,z)\}, \\ \mu(p,q), & \text{otherwise.} \end{cases}$$

It can be noted that the transformation t preserves the vertex degree sequence and the connectivity of the graph. It should be noted that we did not assume that vertexes w, x, y, z are not equal, so it is possible that two, three, or all are equal. It can also be noted that for every $G \in \mathcal{G}_l$ and w, x, y, z there exist w', x', y', z', such that if $G' = t(G,(w,x-,y,z))$, then $t(G',(w',x',y',z')) = G$.

Let $\widehat{s} \in \{0,\ldots,k\}^n$ and $\widehat{s} = (s_0,\ldots,s_n)$ be a sequence where $s_0 \leq \ldots \leq s_n$. By $(s)_i$ we denote the i-th element of s, by $(s)_{\leq i} = (s_1,\ldots,s_i)$ we denote the sequence containing the first i elements of s. By $(s)_{\geq i} = (s_i,\ldots,s_n)$ we denote the sequence containing elements of s starting from the i-th element.
Let $P(G) = \max\{i \in \{0,\ldots,n\} : (\overrightarrow{G})_{\leq i} = (\widehat{s})_{\leq i}\}$.
Let $Pv(G) = \min\{(\overrightarrow{G})_{P(G)+1},\ldots,(\overrightarrow{G})_n\}$.
Let $PP(G) = \min\{i > P(G) : (\overrightarrow{G})_i = Pv(G)\}$.

Lemma 20 *Let $G = (V,\mu) \in \mathcal{G}_l$. It holds that: $PP(G) > P(G) + 1$.*

Corollary 21 *Let $G = (V,\mu) \in \mathcal{G}_l$. It holds that: $PP(G) \geq 2$.*

Let $f : \mathcal{G}_l \setminus \{G_{[\widehat{s}]_\sim}\} \to \mathcal{G}_l$ be a function declared as follows:
let $h = ((\overrightarrow{G})_{PP(G)-2}, (\overrightarrow{G})_{PP(G)-1}, (\overrightarrow{G})_{PP(G)}, (\overrightarrow{G})_{PP(G)+1})$ then $f(G)$ equals to $t(G,h)$. Notice that the function f is well defined because of Corollary 21.

Remark 22 *It can be noted that function f preserves the first $PP(G)-2$ elements of $\overrightarrow{G}$, i.e., $\overrightarrow{G}_{\leq PP(G)-2} = \overrightarrow{f(G)}_{\leq PP(G)-2}$.*

By $f^{-1}(G) = \{H \in \mathcal{G}_l : f(H) = G\}$ we will denote the inverse function of f.

Lemma 23 *Let $G \in \mathcal{G}_l$ then $P(G) \leq P(f(G))$.*

Lemma 24 *Let $G = (V,\mu) \in \mathcal{G}_l$. If $P(G) = P(f(G))$ then $PP(f(G)) < PP(G)$.*

Theorem 25 *For each $G \in \mathcal{G}_l$ there exists $i \in \mathbb{N}$ such that $f^i(G) = G_{[\widehat{s}]_\sim}$.*

Let $\Gamma = (\mathcal{G}_l, \mathcal{E})$ be a graph. $\{G_1, G_2\} \in \mathcal{E}$ if and only if G_1 can be obtained from G_2 by applying the transformation t to G_1 or vice-versa. Let $\widehat{S} = G_{[\widehat{s}]_\sim}$, it is clear that for every $G \in \mathcal{G}_l \setminus \{\widehat{S}\}$ it holds that $\{G, f(G)\} \in \mathcal{E}$.
From above and from Theorem 25 it follows that $(\Gamma, \widehat{S}, f)$ is a finite local search and a reverse search method can be applied to generate all graphs in $\mathcal{G}_l$.

Lemma 26 *The time complexity of $f(G)$ is $O(n^2)$.*

Lemma 27 *The time complexity of $f^{-1}(G)$ is $O(n^6)$.*

Theorem 28 *Traversing Γ by the reverse search method outputs elements with maximal headway of $O(n^6)$.*

Theorem 29 *The computation complexity of the RunningTime algorithm is at most $O(n^{k^2-1})$.*

References

1. Aardenne-Ehrenfest, van T., and de NG Bruijn. "Circuits and trees in oriented linear graphs." Simon Stevin: Wis-en Natuurkundig Tijdschrift 28 (1951): 203.
2. Avis, David, and Komei Fukuda. "Reverse search for enumeration." Discrete Applied Mathematics 65, no. 1 (1996): 21-46.
3. Bellman, Richard. "Dynamic programming treatment of the travelling salesman problem." Journal of the ACM (JACM) 9, no. 1 (1962): 61-63.
4. Lucas, Édouard. Récréations mathématiques. Vol. 1. Gauthier-Villars, 1882.
5. Kirchhoff, Gustav Robert "Über die Auflösung der Gleichungen, auf welche man bei der Untersuchung der linearen Verteilung galvanischer Ströme geführt wird" Ann. Phys. Chem, 72 (1847): 497–508
6. Schlechte, Thomas, Ralf Borndörfer, Berkan Erol, Thomas Graffagnino, and Elmar Swarat. "Micro–macro transformation of railway networks." Journal of Rail Transport Planning & Management 1, no. 1 (2011): 38-48.
7. Tarjan, R. Endre. "A note on finding the bridges of a graph." Information Processing Letters 2, no. 6 (1974): 160-161.

8 Appendix

Proof (Theorem 2). First we will show that Fastest-schedule is NP-hard. Let (C, d, B) be a travelling salesman problem (TSP) instance where C is the set of cities, $d : C^2 \to \mathbb{N}$ is the distance function and B is the maximal searched tour length. For $c \in C$ by $i(c)$ we will denote index of c in an arbitrarily chosen ordering of C, and by c_i we will denote i-th city in the ordering. We define Fastest-schedule instance as follows: $X = C$ for all $c \in X, t(c) = i(c)$, $m(i, j) = d(c_i, c_j), r = 0, k = B$. Every schedule of a given instance contains every train from X exactly once. Therefore it contains each train of every type exactly once. From above every schedule corresponds to a travelling salesman tour (a sequence of types of trains induce travelling salesman tour (TS-tour)). The running time of a schedule of this instance is equal to the length of the corresponding travelling salesman tour. Therefore the fastest schedule corresponds to the shortest TS-tour, so the answer to a given TSP problem is YES, if and only if the answer to the constructed Fastest-schedule problem is YES.
Moreover we will show that the Fastest-schedule problem is in NP. Given an instance of the Fastest-schedule problem (X, t, r, m, k) and a certificate C which is a sequence of trains, we can determine the running time of C by applying the running time formula in polynomial time. The answer to a given problem is YES, if and only if the running time of given certificate is less or equal to k. From the above it follows that the Fastest-schedule problem is NP-complete. □

Proof (Theorem 3). Without loss of generality we can assume that $t(i) \leq t(j) \Leftrightarrow r(t(i)) \leq r(t(j))$ - we relabel train types so trains of types with lower indices are faster. Let us assume that s is the solution to the Fastest-schedule problem and that s does not consist of trains ordered by train type. By s_i we will denote the i-th element of s. By s' we will denote the schedule consisting of trains ordered by train types thus ordered not-decreasing by running time. Let i be the fastest train such that $t(s_i) \neq t(s'_i)$. Let j be the index of the last appearance of a train of type $t(s_i) - 1$. If such train does not exist, i.e., s_i is the fastest train, let $j = 0$. Let us move train s_i to position $j + 1$ in s. The conditions from the theorem guarantee that such operation will not increase running time of the schedule - the condition (1) guarantees that moving the fastest train to the beginning will not increase the time, and the condition (2) guarantees that moving trains to positions grater than 1 will not increase running time. After applying this operation repeatedly, we will obtain s', with a running time that is not larger than the running time of s. □

Proof (Theorem 6). Let 0 be a vertex not in V. Let $V' := V \cup \{0\}, E' := E \cup \{\{0, i\} : i \in V\}$, and $w'_e := w_e$ for $\{i, j\} \in E$ and $w'_e := 0$ for $e \in E' \backslash E$. Then each Hamiltonian cycle in $G' := (V', E', w')$ corresponds to exactly one Hamiltonian path in G and vice versa. Therefore the average weight of all Hamiltonian paths in G equals the average weight of all Hamiltonian cycles in G', which is

$$\frac{2}{(n+1)-1} \sum_{e \in E'} w_e = \frac{2}{n} \sum_{e \in E} w_e. \tag{4}$$

□

Proof (Theorem 10). Let us define r such that $\forall_{i \in S(X)}\ r(i) = 0$. Let H be the sequence of all pairs of train types. Let $m(H_i)$ denote the value of m for the i-th pair from the sequence H. Let

$H_i = (a, b)$. By $l(H_i)$ we will denote $\max\{l(t(a)), l(t(b))\}$. We define m as follows: $m(H_1) = 1$ and $m(H_i) = 2^{\lceil log_2 \sum_{j=1}^{i-1} l(H_j)\rceil+1}$. The function m is constructed in such way that when we analyse RT expressed in a binary number system, we can observe that particular places correspond to a specific pair from H. For a given i values of RT at indices $2^{\lceil log_2 \sum_{j=1}^{i-1} l(H_i)\rceil+1}$ - $2^{\lceil log_2 \sum_{j=1}^{i} l(H_j)\rceil+1}$ depend only on the value of pair H_i. So if $y, z \in S(X)$ and $y \not\sim z$, there exist $p, q \in [k]$ such that $\delta(y, p, q) \neq \delta(z, p, q)$. Let us assume that pair (p, q) has index i in H. Binary notations of $RT(y)$ and $RT(z)$ differ on one of the following indices: $2^{\lceil log_2 \sum_{j=1}^{i-1} l(H_i)\rceil+1}$ - $2^{\lceil log_2 \sum_{j=1}^{i} l(H_j)\rceil+1}$. □

Proof (Lemma 11). In order to calculate the total number of schedules in $R^{-1}(R(s))$ we have to arrange the trains of every type into certain number of blocks. Type i has to be arranged into $b_s(i)$ blocks. This can be done in $l(i)!\binom{l(i)-1}{b_s(i)-1}$ ways - first we choose one of $l(i)!$ permutations and then divide it into the desired number of blocks. To obtain the final number of the schedules we have to take the product over all train types. □

Proof (Theorem 19). Given a number of trains n, a number of types k and a vertex degree sequence l, then every multigraph G with vertex sequence l can be represented as Kirchhoff matrix $K(G)$. All values in matrix $K(G)$ sum up to n, i.e., $\sum_{i,j\in[k]} K(G)_{i,j} = n$. From the former we know that the number of unique Kirchhoff matrices is at most the number of solutions of the following equation: $x_1 + \ldots + x_{k^2} = n$ (every variable in the equation corresponds to one value in $K(G)$). This equation has at most $\binom{n+k^2-1}{k^2-1} = O(n^{k^2-1})$ solutions. □

Proof (Lemma 20). From the definition of $PP(G)$ we know that $PP(G) > P(G)$, so let us suppose on the contrary that $PP(G) = P(G) + 1$. This means that $(\overrightarrow{G})_{P(G)+1} = \min\{(\overrightarrow{G})_{P(G)+1}, \ldots, (\overrightarrow{G})_n\} = \widehat{s}_{P(G)+1}$, which contradicts the definition of $P(G)$. □

Proof (Lemma 23). Since from Lemma 20 we know that $PP(G) \geq P(G) + 2$, it follows from Remark 22 that $\overrightarrow{G}_{\leq P(G)} = \overrightarrow{f(G)}_{\leq P(G)}$ and thus $P(G) \leq P(f(G))$. □

Proof (Lemma 24). From Remark 22 we know that the first $PP(G) - 2$ steps of the *MinimalEulerCycle* algorithm will be in $f(G)$ the same as in G. Let $h = (w, x, y, z)$ be the sequence selected in the definition of f. After $PP(G) - 2$ steps of the *MinimalEulerCycle* algorithm in $f(G)$ we are visiting vertex w. By μ_F^G we will denote function μ_F maintained by algorithm applied to G while visiting w and by $\mu_F^{f(G)}$ when applied to $f(G)$. There are two cases:
(1) the traversal from w to x is not a bridge traversal in (V, μ_F^G), then the traversal from w to y is not a bridge in $(V, \mu_F^{f(G)})$.
(2) the traversal from w to x is a bridge traversal in (V, μ_F^G), which means that $NN_{\mu_F^G}(w) = \emptyset$. From above it can be shown that $NN_{\mu_F^{f(G)}}(w) = \emptyset$.
By definition of f, y is vertex with smallest value in $(\overrightarrow{G})_{\geq P(G)+1}$. From the definition of Fleury's algorithm we also know that $NN_{\mu_F^G}(w)$ is a subset of the set of vertices occurring in $(\overrightarrow{G})_{\geq P(G)+1}$. From above and the case analysis in the former part of the proof we know that the algorithm will traverse from w to y. From the definition of f we know that $(\overrightarrow{G})_{PP(G)} = y$. From the fact that algorithm traverses from w to y instead of x we know that $(\overrightarrow{f(G)})_{PP(G)-1} = y$. From

the definition of f, it follows that $Pv(G) = y$. Because $P(G) = P(f(G))$, we know that also $Pv(G) = Pv(f(G))$, so $Pv(f(G)) = y$. From above it imminently follows that $PP(f(G)) = PP(G) - 1 < PP(G)$. □

Proof (Thorem 25). From Lemma 23 we know that $P(G) \leq P(f(G))$. Let us assume that $P(G) = P(f(G))$, then from Lemma 24 we know that $PP(f(G)) < PP(G)$. Because $PP(G) > P(G)$, there exists a finite j such that $P(G) < P(f^j(G))$. From the former observation it is clear that there exists a finite i such that $P(f^i(G)) = n$. If $P(f^i(G)) = n$, then $f^i(G) = G_{[\widehat{s}]_\sim}$. □

Proof (Lemma 26). The evaluation of function $f(G)$ requires to compute $\overrightarrow{G}$ and functions P, Pv, PP. Former can be done in $O(n^2)$ and latter in $O(n)$, which gives final complexity of $O(n^2)$. □

Proof (Lemma 27). To compute $f^{-1}(G)$, we can enumerate the entire graph H such that there exist $w, x, y, z \in V$ such that $t(H, (w, x, y, z)) = G$. There are $O(n^4)$ sequences $w, x, y, z \in V$ thus we have to enumerate at most $O(n^4)$. For each graph H we have to check if $f(H) = G$ which can be done in a time proportional to $O(n^2)$. From the above we get that the time complexity of f^{-1} is $O(n^6)$. □

Proof (Theorem 28). When performing a traversal of *trace* of Γ between outputting consecutive elements, we have to calculate f^{-1}. Aside from computing f^{-1}, the reverse search routine has to push computed vertices to a queue which can be done in $O(n)$. From the above it follows that dominating operation during reverse search is computing f^{-1} which from Lemma 27 can be done in $O(n^6)$. □