ISSN 0005-1179, Automation and Remote Control, 2014, Vol. 75, No. 7, pp. 1257–1266. © Pleiades Publishing, Ltd., 2014.
Original Russian Text © D.I. Kogan, A.S. Kuimova, Yu.S. Fedosenko, 2014, published in Avtomatika i Telemekhanika, 2014, No. 7, pp. 122–135.

TOPICAL ISSUE

Servicing Problems for a Binary Stream of Objects in a System with a Storage Component

D. I. Kogan*, A. S. Kuimova, and Yu. S. Fedosenko****

*Moscow State University of Instrument Engineering and Informatics, Moscow, Russia
**Volga State Academy of River Transport, Nizhni Novgorod, Russia
kdi_41@mail.ru, anastasia.kuimova@gmail.com, fds@vgavt-nn.ru

Received November 16, 2013

Abstract—We consider a model for single-stage servicing of a finite deterministic stream of objects by a processor with a storage component, a reservoir of bounded capacity. The stream consists of a stream of objects that fill the reservoir and a stream of objects that are filled from the reservoir. With each object, we associate a linear individual penalty function for the time it spends in the servicing system. We study the problem of constructing a schedule that minimizes total penalty over all objects in the stream. Our algorithms are based on dynamic programming, branch and bound techniques, and their joint implementation. We show results of computational experiments.

DOI: 10.1134/S0005117914070078

1. INTRODUCTION

The model we consider in this work arose in the study of cargo processing processes for the tanker fleet in *Severny Zavoz* (delivery of goods to the Northern Territories) [1] via a river port of the city of Salekhard.

During the short-term navigation period, large-capacity tanker fleet delivers diesel fuel along the river Ob from oil refineries in Western Siberia to the port of Salekhard. Arriving ships are sent, in a certain order, to a specialized terminal whose equipment pumps oil products into a reservoir of bounded capacity. Numerous consumption points of diesel fuel, which is the main energy resource in the Polar Region, are mostly located along the banks of the Gulf of Ob and small rivers of the adjacent region on the Yamal peninsula. Fuel is delivered to these points by water, from the Salekhard port by small-capacity ice-class tankers that are loaded on the said specialized terminal. Due to technical restrictions, the terminal cannot service more than one tanker at once.

In this scheme of the *Severny Zavoz*, tankers with different technical and economic parameters are used. The dispatching problem is to find a control strategy for the sequence of their cargo processing, i.e., a servicing schedule, which in the operative planning horizon will reduce the total maintenance costs caused by nonproductive delays in the fleet.

The specifics of dispatching problems is that between the moment when initial data of a specific problem have been completely determined and the moment when one has to begin work according to the constructed schedule only a relatively small time interval passes. During this time interval, the problem must be solved. Therefore, in a mathematical study of each typical dispatching problem it is important to obtain an estimate of its computational complexity. Vyacheslav Sergeevich Tanaev was one of the first mathematicians who did active research into the problems of computational complexity of scheduling theory problems; in the works of Tanaev and his coauthors, including world-renowned books [2, 3], one of the key problems is to divide the problems into polynomially solvable and NP-hard ones [4].

This paper consists of six sections and an Appendix. Section 2 gives a formal description for the servicing model, shows the mathematical setting of the synthesis problem for an optimal servicing schedule, and formulates computational hardness results related to this problem. Section 3 is devoted to a description of the procedure for solving the posed problem based on dynamic programming recurrent relations; a method for solving this problem with a branch and bound scheme is shown in Section 4. Section 5 considers some ways to reduce computation in the synthesis of servicing schedules, including a technique related to combined application of dynamic programming and the branch and bound approach. The same section also introduces a specialization of the considered problem that presupposes that the number of types of objects under servicing is bounded; this special case turns out to be polynomially solvable. Proofs of hardness theorems shown in Section 2 are relegated to the Appendix.

2. MATHEMATICAL MODEL OF SERVICING, OPTIMIZATION PROBLEM SETTING, AND A STUDY OF COMPUTATIONAL COMPLEXITY

We study a model $\boldsymbol{M}$ where a finite stream of objects $O_n = \{o_1, o_2, \ldots, o_n\}$ is subject to single-phase servicing by a stationary processor with a storage component represented by a reservoir of capacity V^*. For each object o_i, $i = \overline{1,n}$, the following integer values are known: t_i is the moment when this object arrives to the servicing queue; τ_i is the normal servicing duration; a_i is the penalty per unit of time that the object spends in the servicing system; v_i is the volume characteristic (object capacity), $V^* \geqslant v_i$. The stream O_n consists of substreams O^+ and O^-. Objects of substream O^+ are intended to fill the reservoir; object of substream O^-, to be filled from the reservoir. An object o_i, $i = \overline{1,n}$, belongs to one of the substreams according to the value of parameter w_i ($w_i = +1$ if $o_i \in O^+$; $w_i = -1$ if $o_i \in O^-$). Objects are numbered in the order of their arrival: $0 = t_1 \leqslant t_2 \leqslant \ldots \leqslant t_n$.

We assume that the processor cannot service more than one object at a time; servicing each object is done without interruption. We denote the reservoir content at time moment t with a variable $V(t)$ with known initial value $V(0)$. Servicing an object o_i from substream O^+ can begin if the reservoir has sufficient free space; as a result of servicing of an object o_i, the reservoir contents are increased by the value v_i. Servicing an object o_i from substream O^- can begin if the reservoir has sufficient fuel in it; as a result of servicing this object, the reservoir contents decrease by the value v_i.

A servicing schedule for stream O_n is defined as a permutation $\rho = \{i(1), i(2), \ldots, i(n)\}$ of the collection of indices $N = \{1, 2, \ldots, n\}$; when it is implemented, an object with index $i(k)$ is serviced as the kth in queue ($k = \overline{1,n}$). A schedule ρ is called admissible if during its implementation the reservoir satisfies the volume constraints indicated above, i.e., $0 \leqslant V(0) + \sum_{p=1}^{q} v_{i(p)} w_{i(p)} \leqslant V^*$, $q = 1, 2, \ldots, n$. We denote the collection of servicing schedules admissible in model $\boldsymbol{M}$ by Ω.

Obviously, the reservoir content after servicing of all objects from stream O_n equals $V(0) + \sum_{i=1}^{n} v_i w_i$, and the double inequality

$$0 \leqslant V(0) + \sum_{i=1}^{n} v_i w_i \leqslant V^* \tag{1}$$

is a necessary but not sufficient condition that the set Ω is nonempty. Let us give an illustrative example: we let $V(0) = V^* = 5$; substream O^+ has a single object with volume characteristic equal to five, and substream O^- contains five objects with volume characteristic of each equal to two. Here condition (1) does hold, but there obviously does not exist a servicing schedule.

We distinguish and call model $\boldsymbol{M}_0$ a common special case of model $\boldsymbol{M}$ in which all objects subject to servicing are present in the system from the very beginning: $t_k = 0$, $k = 1, 2, \ldots, n$.

Theorem 1. *The problem of finding, by the initial data of model $\boldsymbol{M}_0$ (given that condition (1) holds), whether the set of admissible servicing schedules in this problem is nonempty, is NP-complete in the strong sense.*

Proof of Theorem 1 is given in the Appendix.

It is easy to see that when condition (1) holds, the following condition:

$$2\max_{i=\overline{1,n}} v_i \leqslant V^* \tag{2}$$

is sufficient to ensure that the collection Ω is nonempty.

In the considered class of water transport applications, inequalities (1) and (2) always hold, and in what follows we will assume that they do hold.

We also note that conditions

$$\left\{\sum_{i:o_i\in O^-} v_i \leqslant V(0)\right\} \;\&\; \left\{\sum_{i:o_i\in O^+} v_i \leqslant V^* - V(0)\right\}$$

are necessary and sufficient in order for any servicing schedule to be admissible.

We assume that the processor is ready to service objects starting from time moment $t = 0$. Each admissible schedule $\rho = \{i(1), i(2), \ldots, i(n)\}$ uniquely defines, for an arbitrary object $o_{i(k)}$, the time moments when its servicing begins and ends; in what follows we will denote these moments $t_{beg}(i(k), \rho)$ and $t^*(i(k), \rho)$ respectively. The said moments, sequentially in the order of increasing values of the parameter k are computed as

$$\begin{gathered} t_{beg}(i(1), \rho) = t_{i(1)}; \\ t_{beg}(i(k), \rho) = \max[t^*(i(k-1), \rho), t_{i(k)}], \quad k = 2, 3, \ldots, n; \\ t^*(i(k), \rho) = t_{beg}(i(k), \rho) + \tau_{i(k)}, \quad k = 1, 2, \ldots, n. \end{gathered}$$

When implementing a schedule ρ, the total penalty $K(\rho)$ over all objects from the stream O_n equals $\sum_{i=1}^n a_i[t^*(i, \rho) - t_i]$.

Problem 1. Find an admissible servicing schedule that minimized the total penalty value, i.e.,

$$\min_{\rho\in\Omega} K(\rho). \tag{3}$$

Problem 1_0 is Problem 1 formulated for the special case model $\boldsymbol{M}_0$, i.e., for the case when all objects subject to servicing are present in the system from the very beginning: $t_k = 0$, $k = 1, 2, \ldots, n$.

Theorem 2. *Problem 1_0 is NP-hard in the strong sense.*

Proof of Theorem 2 is given in the Appendix.

3. SOLVING PROBLEM 1 WITH DYNAMIC PROGRAMMING

We consider finding the necessary sequence of object servicing as the synthesis of an optimal trajectory in the correspondingly constructed discrete system with a finite number of states and a finite number of controls possible in each state; stepwise penalties depend on the controls being chosen. We are to find the sequence of controls minimizing the total penalty that transfers the system from the original state into one of its final states. This problem can be solved with dynamic programming [5].

We denote the optimal value of the criterion in Problem 1 by K_{opt}. Let $R(t)$ be the subset of indices defined by this problem's input data for the objects of stream O_n that enter the servicing system at discrete time moment t; here $R(0)$ is the subset of object indices awaiting servicing at time moment $t = 0$. We denote the collection of object indices arriving to the system over time interval $[t+1, t+\Delta]$, where $\Delta \geqslant 1$, by $D(t, \Delta)$; thus, $D(t, \Delta) = \bigcup_{i=1}^{\Delta} R(t+i)$.

System state at each moment t of making a decision to load the processor (i.e., determine which of the waiting objects is to be accepted for immediate servicing) is completely characterized by a pair (t, Q), where Q is the collection of object indices that have arrived but not yet been serviced by the time moment t; at time moment t the processor is idle. We note that the reservoir fillup $V(t, Q)$ is uniquely determined by the set Q and time t. We denote by $Z(t, Q)$ the special case obtained from the original problem under the assumption that by time moment t we have to finish servicing all arriving objects except those whose indices fall into the collection Q; the minimized criterion is the total penalty over objects whose servicing ends no later than the time moment t. We denote the optimal value of this criterion in problem $Z(t, Q)$ by $K(t, Q)$.

The set of unserviced objects Q will always additionally include a dummy (zero) object o_0 with characteristics $a_0 = 0$, $\tau_0 = 1$, $v_0 = 0$; to be definite, we let $w_0 = +1$. Servicing a dummy object means processor downtime during one processing cycle. Processor downtime occurs if: a) all previously arrived objects have already been serviced, and other objects from the stream have not yet arrived; b) none of the objects that have previously arrived and are awaiting servicing from time moment t can be accepted for servicing because the reservoir either does not have enough free space (for objects from substream O^+) or does not have enough product (for objects from substream O^-); c) it makes sense to let the processor go idle in order to ensure high priority servicing of some object with high penalty per unit of idle time that has not yet arrived. In what follows we will not specifically note the presence of the dummy object in the set Q.

At the current state (t, Q) an object o_i, $i \in Q$, is admitted for servicing only under one of the following conditions: $(w_i = +1) \,\&\, (V(t, Q) + v_i \leqslant V^*)$ or $(w_i = -1) \,\&\, (V(t, Q) \geqslant v_i)$. We denote the set of indices of objects admitted for servicing in state (t, Q) by $Q^*(t, Q)$. After performing the servicing of object o_i, $i \in Q^*(t, Q)$, reservoir fillup becomes equal to $V(t, Q) + w_i v_i$.

Note that it is always admissible to service the dummy object: $0 \in Q^*(t, Q)$.

We call a state (t', Q') immediately preceding state (t, Q) if the set $Q^*(t', Q')$ containts an object o_α such that $t = t' + \tau_\alpha$ and $Q = (Q' \setminus \{\alpha\}) \cup D(t, \tau_\alpha)$. If in state (t', Q') the next serviced object is the said object o_α, then (t, Q) is the next system state, and the next decision regarding processor will be made there. We denote by $\alpha(t', Q', t, Q)$ the index of an object whose servicing leads to a transition of the system from state (t', Q') into state (t, Q). We denote the set of all states immediately preceding (t, Q) by $N(t, Q)$.

The initial system state is the pair $(0, R(0))$. We call states of the form $(t_n + \theta, \varnothing)$, where θ is an arbitrary positive constant, final states. We denote the set of all final states by $\mathbf{F}$.

From the definitions of the set $R(t)$ and function $K(t, Q)$ we have that

$$K(0, R(0)) = 0. \tag{4}$$

If some non-initial state (t, Q) realizes an immediate transition, i.e., transition by servicing a single object, from state (t', Q'), $(t', Q') \in N(t, Q)$, then over the time interval $[t', t]$ the processor services object $o_{\alpha(t', Q', t, Q)}$. Here $K(t', Q')$ is the minimal total penalty over all objects whose servicing had ended before we transitioned into state (t', Q'), and the penalty for object $o_{\alpha(t', Q', t, Q)}$ equals $a_{\alpha(t', Q', t, Q)}(t - t_{\alpha(t', Q', t, Q)})$. Due to the Bellman's principle [5] we get that

$$K(t, Q) = \min_{(t', Q') \in N(t, Q)} \left[K(t', Q') + a_{\alpha(t', Q', t, Q)}(t - t_{\alpha(t', Q', t, Q)}) \right]. \tag{5}$$

Obviously,

$$K_{opt} = \min_{(t,Q)\in\mathbf{F}}[K(t,Q)]. \tag{6}$$

Formulas (4)–(6) are dynamic programming relations intended to implement a forward dynamic programming procedure [6, p. 98] without considering states that are unreachable from the initial state. Computations of the values $K(t,Q)$ with these relations are done in the increasing order of the values of parameter t. The corresponding algorithm, which we call Algorithm **A**, operates as follows (we assume that the lists G_0–G_4 introduced in the algorithm description begin empty).

Step 1. Fix $K(0,R(0)) = 0$; add the corresponding record $[0,R(0),0]$ to the list G_0 (the list G_0 created for each considered pair (t,Q) over the algorithm's operation receives the record $[t,Q,K(t,Q)]$).

Step 2. Extract the initial state $(0,R(0))$. The resulting five-component records are added to the list G_1.

Step 3. Delete from list G_1 and move to list G_2 records with minimal value of the fourth component (parameter t); if list G_1 is empty then go to Step 6.

Step 4. For every pair (t,Q) present in the records of the list G_2 as their fourth and fifth components (note that a single pair (t,Q) can appear as the fourth and fifth components of several records): a) by formula (5), with the information contained in the list G_0 regarding already found values of the criterial function we compute the value $K(t,Q)$; here we find the state (t^*,Q^*) which realizes the minimum of the right-hand side in (6) and from which we should make an immediate transition into (t,Q) when implementing the optimal schedule in problem $Z(t,Q)$, and the corresponding index α^*, $\alpha^* = \alpha(t^*,Q^*,t,Q)$; b) add record $[t^*,Q^*,\alpha^*,t,Q,K(t,Q)]$ to the list G_3; add record $[t,Q,K(t,Q)]$ to the list G_0; if the pair (t,Q) is final then add another record $[t,\varnothing,K(t,\varnothing)]$ to the list G_4.

Step 5. 1. For each nonfinal state, a pair (t,Q) present in the records of list G_3 as their fourth and fifth components, perform the extraction procedure. The resulting five-component records are added to the list G_1. 2. Clear the list G_2 and go to Step 2.

Step 6. In the resulting list G_4, find the record $[t^+,Q^+,K(t^+,Q^+)]$ with the smallest value of the third component; $K(t^+,Q^+)$ is the optimal value of the criterion in Problem 1, $K(t^+,Q^+) = K_{opt}$.

Note that the list G_3 lets us easily find the optimal schedule of servicing objects for Problem 1.

4. SOLVING PROBLEM 1 WITH THE BRANCH-AND-BOUND METHOD

Implementation of the branch-and-bound method [7, Chap. 10; 8, Chap. 3] includes constructing a fragment of the search tree which is sufficient to find the optimal solution. Vertices of the tree correspond to states (t,Q) of the servicing system; in particular, its root corresponds to state $(0,R(0))$. In what follows we call vertex names the names of the corresponding states.

The computational procedure for the branch-and-bound method is completely defined by specifying methods for: a) branching; b) finding upper bounds for the total penalty (UB) in the resulting vertices of the search tree; c) finding lower bounds on the total penalty (LB) in the resulting vertices of the search tree; d) choosing the vertex for next branching.

Branching in a vertex is similar to extracting a state in the dynamic programming solution. We remind that to extract an arbitrary nonfinal state $S = (t',Q')$ means to generate all records defining states immediately following S; each of these records has the form $[t',Q',\alpha,t,Q]$, where α is the index of the object whose servicing transfers the system from state (t',Q') into state (t,Q). When branching by vertex (t',Q'), each record generated in the extraction of the state with the same name $[t',Q',\alpha,t,Q]$ is shown as an arc with label α which goes from vertex (t',Q') to the new vertex (t,Q).

When computing the upper and lower bounds at some vertex (t, Q), we use the fact that the sequence $\rho(t, Q)$ of objects o_i to be serviced that have already arrived to the system at time moment t and such that $i \notin Q$, is already known. This sequence is given by the labels of arcs that form the path from the root of the current search tree to the vertex (t, Q).

To get the bounds, we will use an algorithm that solves the following elementary problem.

Find the servicing schedule for a single processor that minimizes the total penalty for a set of objects $1, 2, \ldots, n$ in the situation when each object o_i is characterized by two values: τ_i is the servicing duration, and a_i is the penalty per unit of time spent in the servicing system; all objects are ready for servicing starting at time moment $t = 0$; there are no volume characteristics of the objects. The solving algorithm [9, Chap. 2, § 4] computes for every object o_i the value $\mu_i = a_i/\tau_i$; we call μ_i the μ-characteristic of an object. Then objects are ordered in decreasing order of their μ-characteristics; the resulting sequence is the optimal servicing schedule. This algorithm, which we call μ-algorithm, can be adapted as a heuristic algorithm for the generalization when objects for servicing arrive over time, i.e., for every object o_i we additionally know the time moment t_i when it is ready for servicing. At every subsequent decision making moment, from the collection of waiting objects we must choose and accept for immediate servicing an object with maximal μ-characteristic. We call the resulting schedule for the generalized problem a μ-schedule. By contradiction we show that if during the realization of a μ-schedule during the servicing of an object o_i there does not arrive an object o_k such that $\mu_k > \mu_i$, then the resulting μ-schedule is optimal.

To get an upper bound UB in a search tree vertex (t, Q), we supplement sequence $\rho(t, Q)$ with the following algorithm: in state (t, Q), we select from the collection $Q^*(t, Q)$ an index j that ensures the maximal possible value of the μ-characteristic; then we set that at time moment t servicing object o_j begins; we find the next object to service for every subsequent system state in the same way. Let $\rho^*(t, Q)$ be the resulting n-element schedule. Then $K(\rho^*(t, Q))$ is the upper bound UB corresponding to vertex (t, Q).

To find the lower bound LB for some vertex (t, Q) we do the following.

We assume $\rho(t, Q)$ as the initial part of the servicing schedule. Then we disregard volume characteristics of objects (it becomes unimportant which substream a certain object belongs to) and assume that all objects can be divided into parts. In state (t, Q), we choose from collection Q the index j of the object with maximal μ-characteristic. Servicing for object o_j starts at time moment t. Two cases are possible: 1) during the servicing of o_j the system does not receive an object o_k with a larger value of its μ-characteristics; 2) such an object does arrive in the system. In the first case we assume that servicing object o_j is realized without fractions and ends at time moment $t + \tau_j$. In the second case, we divide object o_j into two objects: $o_{j[1]}$ and $o_{j[2]}$. Here $\tau_{j[1]}$ and $\tau_{j[2]}$ are assumed to be equal to $t_k - t$ and $\tau_j - (t_k - t)$ respectively; $a_{j[1]}$ and $a_{j[2]}$ are set to $a_j\tau_{j[1]}/\tau_j$ and $a_j\tau_{j[2]}/\tau_j$ respectively. Note that the total penalty value for objects $o_{j[1]}$ and $o_{j[2]}$ in their sequential servicing over time interval $[t, t+\tau_j]$ is less than the penalty for object o_j when it is serviced over the same time interval. Servicing an object $o_{j[1]}$ ends at time moment t_k; object $o_{j[2]}$ is included in the collection of objects awaiting servicing at the decision making time t_k.

Acting in a similar fashion, we reconstruct the schedule from the state obtained at time moment $t + \tau_j$ when implementing the first case, and from the state obtained at time moment t_k when implementing the second case. The resulting schedule $\rho^{**}(t, Q)$ consists of $n + r$ elements, where r is the number of object divisions in the schedule (note that divided objects may be further subdivided later).

These object divisions expand the possible ways of servicing objects and reduces the minimal total penalty. Thus, $K(\rho^{**}(t, Q))$ is the lower bound LB corresponding to vertex (t, Q).

One can choose a vertex for the next branching with any standard technique; in particular, we can set that the open vertex with maximal rank is always subject to branching (rank of a vertex

Table 1

$t_i, i = \overline{1,n}$	$w_i = +1$			$w_i = -1$		
	a_i	τ_i	v_i	a_i	τ_i	v_i
$[t_{i-1}, t_{i-1} + 10]$	$[7, 15]$	$[8, 20]$	$[V^*/10, V^*/2]$	$[1, 7]$	$[1, 5]$	$[V^*/20, V^*/4]$

Table 2

n	**A**	**B**	n	**A**	**B**
10	0.013	0.000	15	208.760	0.502
11	0.059	0.001	16	832.329	1.560
12	0.466	0.006	17	–	9.542
13	2.124	0.028	18	–	43.459
14	18.678	0.107	19	–	478.901

is the number of arcs in the path from the tree root to this vertex). In our implementation fo the method, we choose for each next branching the open vertex with minimal value of the upper bound UB. If there are several such vertices, we choose the maximal rank vertex among them. We call this technology for solving Problem 1 Algorithm **B**.

To compare the performance of algorithms **A** and **B**, we have conducted computational experiments on test datasets; for every fixed value of n from 10 to 19, we performed a computation for one hundred streams O_n; parameters of objects in streams were defined randomly according to a normal distribution from the intervals shown in Table 1.

Average times that algorithms **A** and **B** took to solve the problem depending on the number of objects in the stream are shown in Table 2.

5. HOW TO REDUCE THE TIME NEEDED TO SYNTHESIZE SERVICING SCHEDULES

5.1. One way to improve performance includes solving Problem 1 with a hybrid technique that uses both dynamic programming relations and bounds characteristic for the branch-and-bound method. Here we implement a modification of Algorithm **A** where for newly obtained states one from time to time additionally computes estimates UB and LB. The lowest of the upper bounds UB obtained during computation is called the record (we let the initial record value to be $+\infty$, and it decreases during the computation). Obviously, extracting states whose lower bounds LB exceed the current record value does not make sense, so we can reduce the number of extracted states and thus improve performance.

In the modified version Algorithm **A** after each pth execution of Step 3, when p is a multiple of a given constant k (k is the algorithm's parameter), we go to an additional Step 3* instead of Step 4.

Step 3*. 1. For each nonfinal state, i.e., a pair (t, Q) present in records of list G_3 as the fourth and fifth components, execute the procedure for getting the upper and lower bounds $K(\rho^*(t, Q))$ and $K(\rho^{**}(t, Q))$ respectively. 2. If the minimal upper bound obtained on this step is less than the record value R, set R to this new value. 3. Delete from list G_3 records $[t^*, Q^*, \alpha^*, t, Q, K(t, Q)]$ such that $K(\rho^{**}(t, Q)) > R$.

Then the algorithm proceeds to Step 4.

5.2. Another way to reduce the complexity of servicing schedule synthesis is to pass to specializations of the considered model (similar to how it was done in [10] for a simpler problem).

We say that two objects have the same type if they belong to the same substream, have the same capacity, the same normative servicing duration, and equal penalties per unit of idle time.

We call Problem $1^{p,q}$ a special case of Problem 1 under the assumption that the numbers of object types in substreams O^+ and O^- do not exceed given constants p and q respectively (in known practical applications, constants p and q are rather small).

We will assume that types of objects are numbered: objects from substream O^+ belong to types $1, 2, \ldots, p$, while objects from substream O^- belong to types $p+1, p+2, \ldots, p+q$.

We characterize each set of objects U with a $(p+q)$-dimensional vector $D(U) = (d_1, d_2, \ldots, d_{p+q})$, where d_i is the number of objects of the ith type in the set U, $i = 1, 2, \ldots, p+q$. We call sets of objects U_1 and U_2 equal-valued if $D(U_1) = D(U_2)$. Note that in special case problems $Z(t, Q)$ that we have introduced when considering Problem 1, and for optimal values of their criteria the following holds: if sets Q_1 and Q_2 are equal-valued then values $K(t, Q_1)$ and $K(t, Q_2)$ coincide. Therefore, we define function $K^*(t, W)$, where W is a $(p+q)$-dimensional vector with integer nonnegative coordinates, as follows: $K^*(t, W) = K(t, Q)$, where Q is any set of objects such that $D(Q) = W$. Thus, $K^*(t, W)$ is the minimal possible total penalty over all arrived objects whose servicing is over by at the very latest the time moment t given that the set of arrived but not yet serviced objects by this time moment is characterized by vector W.

The basic recurrent relation (5) shown above for computing values of function $K^*(t, W)$ is modified in an obvious way, and the total number of considered vectors W that can serve as arguments for this function is bounded from above by n^{p+q-1}.

Under the assumption that $t_n \leqslant C$ for some constant C independent of n, which is a natural assumption for the considered class of problems, we get that the algorithm for solving Problem $1^{p,q}$ based on dynamic programming relations for fixed parameters p and q has computational complexity polynomial in the number of objects intended for servicing.

In applications, Problem $1^{p,q}$ often comes with additional information that limits the number of objects awaiting servicing. If we assume that at any decision making moment the number of objects in the system that are awaiting servicing cannot exceed a natural constant r, we get that the total number of vectors under consideration W that may serve as arguments for the function $K^*(t, W)$ is bounded from above by the value r^{p+q-1}, i.e., a constant that does not depend on n. The number of computations of function $K^*(t, W)$ needed to solve Problem 1 becomes linear in n, and the solving algorithm based on dynamic programming principles now has a computational complexity estimate which is quadratic in n.

6. CONCLUSION

In this work, we have presented a model of single-stage servicing for a binary stream of objects with a processor with a storage component. We have formulated for problem of synthesizing a servicing schedule that minimizes the total penalty over all objects; we have considered computational complexity estimates; based on the concepts of dynamic programming, branch and bound techniques, and their joint implementation we constructed decision algorithms. Despite our results regarding the NP-hardness of the problem in question, computations done on real life datasets took very reasonable time. For instance, in each practical problem with 17–19 objects optimal schedule synthesis could be done on a desktop PC (processor Intel Core 2 Duo 3.16 GHz, 4 Gb RAM) in 8–13 minutes.

ACKNOWLEDGMENTS

This work was supported by the Research Foundation of Volga State Academy of River Transport, project no. 02-2013.

APPENDIX

Proofs of Theorems 1 and 2 consist of polynomial reductions of the 3-partition problem [4, p. 283], which is NP-hard in the strong sense, to problems formulated in these Theorems.

The 3-partition problem is as follows. Consider a finite set of natural numbers $B = \{b_1, b_2, \ldots, b_{3n}\}$ such that each b_i belongs to interval $(T/4, T/2)$, and $\sum_{i=1}^{3n} b_i = nT$; here T is some natural number. The problem is whether the set B can be partitioned into n pairwise disjoint subsets $B_1, B_2, \ldots, B_n$ so that sums of numbers in each subset are equal. Note that if this question has a positive answer, each of the subsets B_i, $i = \overline{1,n}$, has three numbers whose sum equals T.

Proof of Theorem 1. We construct an instance of the servicing model $\boldsymbol{M}_0^*$ by an instance of the 3-partition problem as follows.

The collection O^+ consists of objects $o_1, o_2, \ldots, o_n$; collection O^-, of objects $o_{n+1}, o_{n+2}, \ldots, o_{4n}$ (objects from collection O^+ are intended to fill the reservoir up, objects from collection O^- are intended to be filled from the reservoir); all objects are ready for servicing at time moment $t = 0$. The reservoir volume equals T, at time moment $t = 0$ it is full, i.e., $V(0) = T$. Capacity of each object from collection O^+ is set to equal T; object o_{n+i} from collection O^- has capacity b_i, $i = 1, 2, \ldots, 3n$. Servicing durations for objects and penalties per unit of idle time are assigned arbitrarily (they are irrelevant for what follows).

Since the capacity of each object from collection O^+ equals T, servicing each subsequent object from O^+ can begin only when reservoir fillup is zero; as a result of servicing an object from O^+ reservoir fillup becomes maximal possible, i.e., equal to T. From time moment $t = 0$ to the moment when servicing of the first object from O^+ begins, a subset of objects from collection O^- defined by a certain collection of indices $M(1)$ such that $\sum_{j \in M(1)} b_j = T$ must be serviced. Then, between each $(i-1)$th and ith objects from collection O^+ (we can assume they are numbered in servicing order) we must service a subset of objects from collection O^- defined by some collection of indices $M(i)$ such that $\sum_{j \in M(i)} b_j = T$, $i = 2, 3, \ldots, n$. Therefore, we get that an admissible servicing schedule can be constructed if and only if the original 3-partition problem has a positive solution. This reduction is realized in time polynomial in the original problem's input size; the reduction does not generate numbers larger than T. This completes the proof of Theorem 1.

Proof of Theorem 2. We construct an instance of the servicing model $\boldsymbol{M}_0^{**}$ by an instance of the 3-partition problem as follows.

The collection O^+ consists of objects $o_1, o_2, \ldots, o_n$; collection O^-, of objects $o_{n+1}, o_{n+2}, \ldots, o_{4n}$ (objects from collection O^+ are intended to fill the reservoir up, objects collection O^- are intended to be filled from the reservoir). All objects are ready for servicing at time moment $t = 0$. The reservoir volume V^* exceeds $2T$, at time moment $t = 0$ the reservoir is full, i.e., $V(0) = V^*$. Capacity of each object from collection O^+ equals T; object O_{n+i} from collection O^- has capacity b_i, $i = 1, 2, \ldots, 3n$. We further let $a_1 = a_2 = \ldots = a_n = 1$; $a_{n+i} = 0$, $i = 1, 2, \ldots, 3n$; $\tau_1 = \tau_2 = \ldots = \tau_n = 1$; $\tau_{n+i} = b_i$, $i = 1, 2, \ldots, 3n$.

Unlike model $\boldsymbol{M}_0^*$, initial data of model $\boldsymbol{M}_0^{**}$ satisfy not only condition (1) but also inequality (2). Therefore, in model $\boldsymbol{M}_0^{**}$ the set of schedules is nonempty.

In order to get a lower bound on the optimal value of the criterion in Problem 1_0 arising in the model $\boldsymbol{M}_0^{**}$, we allow to split objects in substream O^-; thus we extend the set of possible solutions. The optimal value of the criterion in the new problem, which we call Problem $1_0'$, will not exceed the optimal value of the criterion in Problem 1_0. An object o_{n+j}, $j \in \{1, 2, \ldots, 3n\}$, can be split into two objects: $o_{n+j[1]}$ and $o_{n+j[2]}$. Here we let $\tau_{n+j[1]}$ and $\tau_{n+j[2]}$ to be equal to $\xi_j \tau_{n+j}$ and $(1-\xi_j)\tau_{n+j}$ respectively; $b_{n+j[1]}$ and $b_{n+j[2]}$, equal to $\xi_j b_{n+j}$ and $(1-\xi_j)b_{n+j}$ respectively. Here ξ_j is a constant from the interval $(0, 1)$. Problem $1_0'$ has an obvious solution. Since the capacity of each object from collection O^+ (penalties are imposed only for objects from O^+) equals T, servicing each subsequent

object from O^+ can begin only when the reservoir fillup is equal to $V^* - T$. Therefore, from time moment $t = 0$ up to reservoir fillup $V^* - T$ we can service any objects from the set O^- (it is possible here to split the last of the serviced objects). Then we service any object from collection O^+. Then, between $(i-1)$th and ith objects from collection O^+, $i = 2, 3, \ldots, n$, one has to service (with a possible splitting) a subset of objects from collection O^- defined by some collection of indices $M(i)$ such that $\sum_{j \in Mi} b_j = T$.

This organization of the servicing process ensures the total penalty

$$\mathbf{S} = (T+1) + (2T+2) + \ldots + (nT+n) = n(n+1)(T+1)/2;$$

$\mathbf{S}$ is a lower bound on the optimal value of the criterion in Problem $1'_0$, where it is impossible to split objects. This bound is reachable: the optimal value of the criterion in Problem $1'_0$ with unsplittable items equals $\mathbf{S}$ if and only if the set of natural numbers $B = \{b_1, b_2, \ldots, b_{3n}\}$ can be divided into n pairwise disjoint subsets $B_1, B_2, \ldots, B_n$ such that sums of numbers in each subset are equal. This reduction is realized in time polynomial in the original problem's input size; the reduction does not generate numbers larger than T. This completes the proof of Theorem 2.

REFERENCES

1. *Severnyi zavoz*, Wikipedia, http://ru.wikipedia.org/wiki/Severnyi_zavoz (accessed on 31.05.2013).
2. Tanaev, V.S., Gordon, V.S., and Shafranskii, Ya.M., *Teoriya raspisanii. Odnostadiinye sistemy* (Scheduling Theory. Single-Stage Systems), Moscow: Nauka, 1984.
3. Tanaev, V.S., Sotskov, Yu.N., and Strusevich, V.A., *Teoriya raspisanii. Mnogostadiinye sistemy* (Scheduling Theory. Multi-Stage Systems), Moscow: Nauka, 1989.
4. Garey, M.R. and Johnson, D.S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, San Francisco: Freeman, 1979. Translated under the title *Vychislitel'nye mashiny i trudnoreshaemye zadachi*, Moscow: Mir, 1982.
5. Bellman, R., *Dynamic Programming*, Princeton: Princeton Univ. Press, 1957. Translated under the title *Dinamicheskoe programmirovanie*, Moscow: Inostrannaya Literatura, 1960.
6. Pinedo, M.L., *Scheduling: Theory, Algorithms, and Systems*, New York: Springer, 2008.
7. Korbut, A.A. and Finkel'shtein, Yu.Yu., *Diskretnoe programmirovanie* (Discrete Programming), Moscow: Nauka, 1969.
8. Sigal, I.Kh. and Ivanova, A.P., *Vvedenie v prikladnoe diskretnoe programmirovanie* (Introduction to Applied Discrete Programming), Moscow: Nauka, 2007.
9. Tanaev, V.S. and Shkurba, V.V., *Vvedenie v teoriyu raspisanii* (Introduction to Scheduling Theory), Moscow: Nauka, 1975.
10. Kogan, D.I. and Fedosenko, Yu.S., The Dispatching Problem: Analysis of Computational Complexity and Polynomially Solvable Subclasses, *Diskret. Mat.*, 1966, vol. 8, no. 3, pp. 135–147.

This paper was recommended for publication by A.A. Lazarev, a member of the Editorial Board