Dynamic programming algorithms for producing food mixture packages by automatic combination weighers

Shinji IMAHORI*, Yoshiyuki KARUNO** and Kenju TATEISHI***

* Department of Computational Science and Engineering, Nagoya University
Furo-cho, Chikusa-ku, Nagoya 464-8603, Japan

** Department of Mechanical and System Engineering, Kyoto Institute of Technology
Matsugasaki, Sakyo-ku, Kyoto 606-8585, Japan
E-mail: karuno@kit.ac.jp

*** Graduate School of Science and Technology, Kyoto Institute of Technology
Matsugasaki, Sakyo-ku, Kyoto 606-8585, Japan

Received 17 December 2013

Abstract
The lexicographic bi-criteria combinatorial optimization problem to be discussed in this paper is a mathematical model of the food mixture packing performed by so-called automatic combination weighers, and it is described as follows. We are given a union $I = I_1 \cup I_2 \cup \cdots \cup I_m$ of $m$ sets of items, where for each $i = 1, 2, \ldots, m$, $I_i = \{I_{ik} \mid k = 1, 2, \ldots, n\}$ denotes a set of $n$ items of the $i$-th type and $I_{ik}$ denotes the $k$-th item of the $i$-th type. Each item $I_{ik}$ has an integral weight $w_{ik}$ and an integral priority $\gamma_{ik}$. The problem asks to find a union $I' = I'_1 \cup I'_2 \cup \cdots \cup I'_m$ of $m$ subsets of items where $I'_i \subseteq I_i$ so that the total weight of chosen items for $I'$ is no less than an integral target weight $T$, and the sum weight of chosen items of the $i$-th type for $I'_i$ is no less than an integral indispensable weight $b_i$. The total weight of chosen items for $I'$ is minimized as the primary objective, and further the total priority of chosen items for $I'$ is maximized as the second objective. For the case in which there are two types of items (i.e., $m = 2$), we propose an $O(nT)$ time dynamic programming algorithm, applying a linear search technique. We also conduct numerical experiments to demonstrate the empirical performance.

Key words: Engineering optimization, Food packaging, 0-1 integer programming, Dynamic programming, Pseudo-polynomial time algorithms

1. Introduction
In this paper, we discuss a combinatorial optimization problem arising in automated food packing systems known as automatic combination weighers (Ishida Co., Ltd., 2013). The automated food packing system possesses several hoppers with a weighing function. Some amount of food (such as a green pepper, a handful of potato chips, some pieces of candies, and so on) is thrown into each weighing hopper, and it is called an item with a weight. The combinatorial optimization problem to be discussed in this paper is a mathematical model of the mixture packing operation performed by the automated food packing system, and it is described as follows.

We are given a union $I = I_1 \cup I_2 \cup \cdots \cup I_m$ of $m$ sets of items, where for $i = 1, 2, \ldots, m$, $I_i = \{I_{ik} \mid k = 1, 2, \ldots, n\}$ denotes a set of $n$ items of the $i$-th type and $I_{ik}$ denotes the $k$-th item of the $i$-th type. Each item $I_{ik}$ has an integral weight $w_{ik}$ and an integral priority $\gamma_{ik}$. The problem asks to find a union $I' = I'_1 \cup I'_2 \cup \cdots \cup I'_m$ of $m$ subsets of items where $I'_i \subseteq I_i$ so that the total weight of chosen items for $I'$ is no less than an integral target weight $T$, and the sum weight of chosen items of the $i$-th type for $I'_i$ is no less than an integral indispensable weight $b_i$. The former is known as the target weight constraint, which is a hard constraint imposed on automatic combination weighers from a viewpoint of the service conscience (Morinaka, 2000). The latter is also a hard constraint of the mixture packing operation, which we call the indispensable weight constraint.

The chosen items for $I'$ are put into a single package at each mixture packing operation. The resulting empty hoppers
are supplied with new items, and each set \( I_i \) of items of the \( i \)-th type is updated by taking the union of the remaining items in \( I_i - I_i' \) and the new items of the \( i \)-th type, and hence the entire union \( I \) is also updated. The automated food packing system repeats the mixture packing operation to produce a large number of mixture packages one by one. Note that the automated food packing system always chooses some current items in hoppers without knowing the weights of next new items.

The total weight of chosen items for \( I' \) is minimized as the primary objective. This aims at minimizing the total amount of surplus in each mixture package, together with the target weight constraint. Further, the total priority of chosen items for \( I' \) is maximized as the second objective. During an operating run (i.e., a series of iterations of mixture packing operation), an item may be left in hopper for a long time before it is chosen to be packed (Kameoka and Nakatani, 2001). In order to prevent such an undesirable situation, the priority is introduced into each item, which is defined by a non-decreasing function of the duration in hopper of the item. We expect current items with longer durations in hoppers to be preferably chosen by the second objective. In fact, for some non-mixture packing operations (i.e., the automated food packing systems handle a single type of items), the previous numerical results have indicated the effectiveness of the second objective upon reducing the durations in hoppers of items (Karuno, et al., 2007, Karuno, et al., 2010, Imahori, et al., 2011). In the following section, the problem is going to be formulated as a lexicographic bi-criteria 0-1 integer programming problem of off-line setting.

In this paper, we focus on solving the case in which there are two types of items (i.e., \( m = 2 \)). We have already known that a dynamic programming algorithm for the duplex packing operation can be converted into an \( O(nT^2) \) time algorithm for the case of \( m = 2 \), which chooses simultaneously two disjoint subsets from a given set of items of a single type (Imahori, et al., 2010, Imahori, et al., 2012). The pseudo-polynomial time algorithm implies that the case of \( m = 2 \) is not strongly NP-hard, but it is weakly NP-hard (Garey and Johnson, 1979). In this paper, we first construct an \( O(mnT + mT^m) \) time algorithm for an arbitrary \( m \geq 2 \), and then improve the previous time complexity to \( O(nT) \) for the case of \( m = 2 \) by applying a linear search technique. We also conduct numerical experiments to demonstrate the empirical performance of the proposed \( O(nT) \) time algorithm.

2. Problem Formulation

For notational convenience, we assume that the automated food packing system possesses \( n \) weighing hoppers for each of \( m \) types of items by adding some fictitious items to the lists of the non-fictitious (i.e., actual) items. In result, it has totally \( mn \) weighing hoppers.

Let \( \mathcal{H} = \{H_{ik} \mid i = 1, 2, \ldots, m, \ k = 1, 2, \ldots, n\} \) denote the set of \( mn \) weighing hoppers, where \( H_{ik} \) stands for the hopper into which the current \( k \)-th item of the \( i \)-th type has been thrown (see Fig. 1). As in the previous section, \( I_{ik} \) denotes the current \( k \)-th item of the \( i \)-th type, and \( I = \{I_{ik} \mid i = 1, 2, \ldots, m, \ k = 1, 2, \ldots, n\} \) denotes the set of current \( mn \) items. Also, \( I_i = \{I_{ik} \mid k = 1, 2, \ldots, n\} \subset I \) denotes the set of current \( n \) items of the \( i \)-th type.

Let \( \ell_{\text{max}} \) denote the total number of mixture packages to be produced during an operating run, which is equivalent to the number of iterations of mixture packing operation during the operating run. Let \( \ell \) denote the current iteration number
of mixture packing operation \((1 \leq \ell \leq \ell_{\text{max}})\), and let \(\ell_{ik}\) denote the iteration number at which item \(I_{ik}\) has been thrown into the hopper (when it was empty). Then, we refer to \(d_{ik} = \ell - \ell_{ik} + 1\) as the duration in hopper of item \(I_{ik}\).

An instance of the food packing problem at each mixture packing operation consists of the following input data:

- \(I = \{I_k \mid i = 1, 2, \ldots, m, k = 1, 2, \ldots, n\}\): Set of the current \(mn\) items.
- \(w_{ik}\): Positive integer weight of non-fictitious item \(I_{ik}\). It is assumed to be zero for any fictitious item \(I_{ik}\).
- \(\gamma_{ik}:=d_{ik}\): Positive integer priority of non-fictitious item \(I_{ik}\). It is assumed to be zero for any fictitious item \(I_{ik}\).
- \(b_i\): Indispensable weight for the \(i\)-th type items in each mixture package, which is assumed to be a non-negative integer.
- \(T\): Target weight of each mixture package, which is assumed to be a positive integer.

The food packing problem at each mixture packing operation is formulated by using a 0-1 vector \(x = (x_{11}, x_{12}, \ldots, x_{1n}; x_{21}, x_{22}, \ldots, x_{2n}; \ldots; x_{m1}, x_{m2}, \ldots, x_{mn})\) of \(mn\) elements, where

\[
x_{ik} = \begin{cases} 
1 & \text{if item } I_{ik} \text{ is chosen,} \\
0 & \text{otherwise.}
\end{cases}
\]  

\textbf{MIXTURE\_LEXICO}

\[
\begin{align*}
\text{minimize} & \quad f(x) = \sum_{i=1}^{m} \sum_{k=1}^{n} w_{ik} x_{ik} \quad \text{as the primary objective,} \\
\text{maximize} & \quad g(x) = \sum_{i=1}^{m} \sum_{k=1}^{n} \gamma_{ik} x_{ik} \quad \text{as the second objective,} \\
\text{subject to} & \quad \sum_{i=1}^{m} \sum_{k=1}^{n} w_{ik} x_{ik} \geq T, \\
& \quad \sum_{i=1}^{m} \sum_{k=1}^{n} \gamma_{ik} x_{ik} \geq b_i, \quad i = 1, 2, \ldots, m, \\
& \quad x_{ik} \in \{0, 1\}, \quad i = 1, 2, \ldots, m, \; k = 1, 2, \ldots, n.
\end{align*}
\]

The target weight and indispensable weight constraints for each mixture package are represented by Eqs. (4) and (5), respectively. The binary constraints of variables \(x_{ik}\) are expressed by Eq. (6). A solution \(x = (x_{11}, x_{12}, \ldots, x_{1n}; x_{21}, x_{22}, \ldots, x_{2n}; \ldots; x_{m1}, x_{m2}, \ldots, x_{mn})\) satisfying Eqs. (4)–(6) is referred to as a feasible solution of problem MIXTURE\_LEXICO.

The primary objective of Eq. (2) aims at attaining the total weight of chosen items for each mixture package as close to the target weight \(T\) as possible, together with Eq. (4) (i.e., the target weight constraint). The second objective of Eq. (3) is introduced so that the items with longer durations in hoppers are preferably chosen. (Recall that we solve the food packing problem of off-line setting at each mixture packing operation.)

In order to omit some trivial cases, we assume that

\[
0 < w_{ik}^{(i)} = \max_{1 \leq k \leq n} \{w_{ik}\} < T, \quad i = 1, 2, \ldots, m, \\
\]

\[
b_i \leq \sum_{k=1}^{n} w_{ik}, \quad i = 1, 2, \ldots, m, \\
\]

\[
\sum_{i=1}^{m} b_i \leq T \leq \sum_{i=1}^{m} \sum_{k=1}^{n} w_{ik}.
\]

We should remark that the case of \(x_{ik}^{(i)} = b_i = T\) can be solved in \(O(n(b_1 + w_{ik}^{(i)})) = O(mnT)\) time by considering every non-mixture food packing problem for \(m\) types of items independently (Imahori, et al., 2011). For notational convenience, we also define

\[
w_{\text{max}} = \max_{1 \leq i \leq m, 1 \leq k \leq n} \{w_{ik}\} = \max \{\max \{w_{ik}\}\} \quad (< T).
\]

For a given instance of problem MIXTURE\_LEXICO, let \(f^*\) denote the minimum of the total weight of chosen items in a feasible solution, and let \(x = \hat{x}\) denote a feasible solution which attains the minimum of the total weight (i.e., \(f^* = f(\hat{x})\)). An optimal solution \(x = x^*\) is defined as a feasible solution such that it satisfies \(f(x^*) = f^*\) and it maximizes the total priority among feasible solutions with the minimum total weight \(f^*\) (i.e., it satisfies \(g(x^*) \geq g(\hat{x})\) for any feasible solution \(\hat{x}\) with \(f(\hat{x}) = f^*\)). We call \(g^* = g(x^*)\) the conditionally maximum total priority. Problem MIXTURE\_LEXICO asks to find an optimal solution \(x = x^*\). We call the problem MIXTURE\_PRIMAL if we are asked to find a feasible solution \(x = \hat{x}\) with the minimum total weight \(f^*\) (i.e., if the second objective of Eq. (3) is disregarded).
3. Algorithms

As mentioned before, for problem MIXTURE_LEXICO with \( m = 2 \) (i.e., the case in which there are two types of items), a dynamic programming algorithm designed for the duplex food packing problem in the previous papers (Imahori, et al., 2010, Imahori, et al., 2012) can be converted into an \( O(nT^2) \) time algorithm (see Appendix). Another mixture packing system handling two types of items has also been considered by Kameoka et al. (2000), in which there is exactly one current item of the first type and it must be chosen. However, from our viewpoint of mathematical modeling, the packing operation may be suitable to be categorized into the class of non-mixture ones.

In this paper, we propose a new pseudo-polynomial time algorithm for problem MIXTURE_LEXICO with \( m = 2 \), which improves the previous time complexity to \( O(nT) \). The proposed algorithm first solves a non-mixture food packing problem for each type of items individually by a dynamic programming procedure. In the solution process, the dynamic programming procedure maintains \( O(T) \) partial solutions for each non-mixture food packing problem in \( O(nT) \) time. (We are going to provide the mathematical definition of partial solutions in the following subsection, although the definition is quite natural.) Then the proposed algorithm seeks for the best pair of partial solutions for two types of items among \( O(T^2) \) pairs in \( O(T) \) time by a linear search technique.

3.1. Partial Problems

We first define a partial food packing problem for each type of items, which is a non-mixture food packing problem. For \( i = 1, 2, \ldots, m \), let \( x_i = (x_{i1}, x_{i2}, \ldots, x_{in}) \) denote the \( i \)-th 0-1 vector of a solution \( x \) of problem MIXTURE_LEXICO, which we simply call a partial solution (for items of the \( i \)-th type). We may express the solution by \( x = (x_1; x_2; \cdots; x_m) \), instead of using its \( mn \) elements. Note that each objective function of the total weight \( f(x) \) and total priority \( g(x) \) of problem MIXTURE_LEXICO can be written by a separable form (see Eqs. (2) and (3)). Each of \( m \) partial food packing problems is also a lexicographic bi-criteria 0-1 integer programming problem.

**PARTIAL_LEXICO (for items of the \( i \)-th type)**

\[
\begin{align*}
\text{minimize} & \quad f_i(x_i) = \sum_{k=1}^{n} w_{ik} x_{ik} \quad \text{as the primary objective,} \\
\text{maximize} & \quad g_i(x_i) = \sum_{k=1}^{n} y_{ik} x_{ik} \quad \text{as the second objective,} \\
\text{subject to} & \quad \sum_{k=1}^{n} w_{ik} x_{ik} \geq b_i, \\
& \quad x_{ik} \in \{0, 1\}, \quad k = 1, 2, \ldots, n. 
\end{align*}
\]

A partial solution \( x_i \) satisfying Eqs. (13) and (14) is a feasible solution for problem PARTIAL_LEXICO (for items of the \( i \)-th type), and further we refer to it as a partially feasible solution of problem MIXTURE_LEXICO. We remark that a solution \( x = (x_1; x_2; \cdots; x_m) \) consisting of \( m \) partially feasible solutions for all item types may be infeasible for problem MIXTURE_LEXICO since the \( x \) may fail to meet the target weight constraint of Eq. (4). For an instance of PARTIAL_LEXICO (for items of the \( i \)-th type), let \( f_i' \) denote the minimum of the sum weight of chosen items in a feasible solution of PARTIAL_LEXICO (for items of the \( i \)-th type), and let \( x_i' \) denote a feasible solution which attains the minimum, i.e., \( f_i' = f_i(x_i') \).

Let \( x^* = (x_1^*; x_2^*; \cdots; x_m^*) \) be optimal solution of problem MIXTURE_LEXICO, and let \( f_i^* = f_i(x_i^*) \). Then it holds \( f_* = f_1^* + f_2^* + \cdots + f_m^* \). Note that for each \( i = 1, 2, \ldots, m \), the partial solution \( x_i^* \) is also feasible for problem PARTIAL_LEXICO (for items of the \( i \)-th type).

**Lemma 1.** For a given instance of problem MIXTURE_LEXICO, a solution \( x^* = (x_1^*; x_2^*; \cdots; x_m^*) \) consisting of \( m \) partially feasible solutions for all item types with \( f_i' = f_i(x_i') \) and an optimal solution \( x^* = (x_1^*; x_2^*; \cdots; x_m^*) \) with \( f_i^* = f_i(x_i^*) \) satisfy that for each \( i = 1, 2, \ldots, m \),

\[
b_i \leq f_i' \leq f_i^* \leq T - \sum_{j=1}^{m} b_j + w_{\text{max}}^0 - 1. 
\]

**Proof.** For each \( i = 1, 2, \ldots, m \), partial solutions \( x_i' \) and \( x_i^* \) are feasible for problem PARTIAL_LEXICO (for items of the \( i \)-th type), and hence it holds \( \min(f_i', f_i^*) \geq b_i \) (see Eq. (13)). By the minimality of \( f_i' \) for problem PARTIAL_LEXICO (for items of the \( i \)-th type), it is no more than \( f_i^* \). Thus, we have \( b_i \leq f_i' \leq f_i^* \).
Further, for each $i = 1, 2, \ldots, m$, by the optimality of $x^*$, it holds that either $T > f(x^*) - w_{ik}$ or $b_i > f_{ik}^* - w_{ik}$ for any $k$ with $w_{ik} > 0$ and $x_{ik}^* = 1$. Hence we have $T > f(x^*) - w_{ik} = (f_1^* + f_2^* + \cdots + f_n^*) - w_{ik} \geq (b_1 + \cdots + b_{i-1} + f_{i+1}^* + b_{i+1} + \cdots + b_n) - w_{ik} = f_{i}^* + \sum_{j=1}^{n} b_j - w_{ik}$ from the former, and $f_{i}^* - w_{ik} < b_i \leq (T - \sum_{j=1}^{n} b_j) + b_i$ from the latter together with assumption $\sum_{j=1}^{n} b_j \leq T$ (see Eq. (9)), which completes the proof. □

3.2. Dynamic Programming Procedure

For notational convenience, we define the upper bound on the optimal sum weight $f_{ik}^*$ of chosen items of the $i$-th type provided in Lemma 1 as follows:

$$B_i = \min \left\{ T - \sum_{j=1}^{n} b_j + \frac{w_{ik}}{1}, \sum_{i=1}^{n} w_{ik} \right\}. \tag{16}$$

Note that it holds $O(B_i) = O(T)$ by assumption $T > w_{\text{max}}$ (see Eq. (10)).

In order to obtain promising partially feasible solutions for each item type, we design a dynamic programming procedure for partial,LEXICO (for items of the $i$-th type). Without loss of generality, we assume that the first item of each type is a non-fictitious item (i.e., it holds $w_{11} > 0$ and $\gamma_{11} > 0$). The following 0-1 \textit{state variables} are defined for $k = 1, 2, \ldots, n$ and $p = 0, 1, \ldots, B_i$:

$$u_{ik}(p) = 1 \iff \text{ There exists a partial 0-1 vector } (x_{i1}, x_{i2}, \ldots, x_{ik}) \text{ such that } \sum_{j=1}^{k} w_{ij} x_{ij} = p.$$

$$u_{ik}(p) = 0 \iff \text{ Such a partial 0-1 vector does not exist.}$$

The \textit{priority recording variables} $v_{ik}(p)$ for $k = 1, 2, \ldots, n$ and $p = 0, 1, \ldots, B_i$ are also defined to maintain the conditionally maximum of total priority $\sum_{j=1}^{k} \gamma_{ij} x_{ij}$ of the partial 0-1 vector $(x_{i1}, x_{i2}, \ldots, x_{ik})$ such that $\sum_{j=1}^{k} w_{ij} x_{ij} = p$. Further, two kinds of additional variables $\sigma_{ik}(p)$ and $\tau_{ik}(p)$ for $k = 1, 2, \ldots, n$ and $p = 0, 1, \ldots, B_i$ are introduced to make each priority recording variable $v_{ik}(p)$ identify the conditionally maximum for the partial 0-1 vector $(x_{i1}, x_{i2}, \ldots, x_{ik})$ such that $\sum_{j=1}^{k} w_{ij} x_{ij} = p$. For each priority recording variable $v_{ik}(p)$, at most two possible alternatives should be regarded as shown in Eqs. (19)–(22) later, and each $v_{ik}(p)$ takes the maximum of $\sigma_{ik}(p)$ and $\tau_{ik}(p)$.

The 0-1 state variables and priority recording variables are computed by the following dynamic programming recursives: For $p = 0, 1, \ldots, B_i$,

$$u_{i1}(p) = \begin{cases} 1 & \text{if } p \in \{0, w_{1i}\}, \\ 0 & \text{otherwise}, \end{cases} \tag{17}$$

$$v_{i1}(p) = \begin{cases} 0 & \text{if } p = 0, \\ \gamma_{1i} & \text{if } p = w_{1i}, \\ -1 & \text{otherwise}, \end{cases} \tag{18}$$

and for $k = 2, 3, \ldots, n$ and $p = 0, 1, \ldots, B_i$,

$$u_{ik}(p) = \begin{cases} 1 & \text{if } (u_{ik-1}(p) = 1), \text{ or } (w_{ik} > 0, p - w_{ik} \geq 0 \text{ and } u_{ik-1}(p - w_{ik}) = 1), \\ 0 & \text{otherwise}, \end{cases} \tag{19}$$

$$\sigma_{ik}(p) = \begin{cases} v_{ik-1}(p) & \text{if } u_{ik-1}(p) = 1, \\ -1 & \text{otherwise}, \end{cases} \tag{20}$$

$$\tau_{ik}(p) = \begin{cases} v_{ik-1}(p - w_{ik}) + \gamma_{ik} & \text{if } w_{ik} > 0, p - w_{ik} \geq 0 \text{ and } u_{ik-1}(p - w_{ik}) = 1, \\ -1 & \text{otherwise}, \end{cases} \tag{21}$$

$$v_{ik}(p) = \max \{ \sigma_{ik}(p), \tau_{ik}(p) \}. \tag{22}$$

In this paper, we call the dynamic programming procedure PartialDP (for items of the $i$-th type). The time complexity of $O(nT)$ and correctness are sketched by following the manner of Imahori et al. (2011).

The computation of all variables $u_{ik}(p), v_{ik}(p), \sigma_{ik}(p)$ and $\tau_{ik}(p)$ by Eqs. (17)–(22) requires $O(nB_i) = O(nT)$ time since it holds $O(B_i) = O(T)$ (see Eq. (16)). The condition of $w_{ik} > 0$ appearing in Eqs. (19) and (21) is put to exclude any fictitious item from the resulting solution. We also remark that in an implementation of the procedure, it suffices to prepare an $O(1)$ space for maintaining the additional variables $\sigma_{ik}(p)$ and $\tau_{ik}(p)$ temporarily, since these variables are referred only for the $v_{ik}(p)$ in Eq. (22).

After the computation, we can find all values of $p = \hat{p}$ such that $u_{ik}(\hat{p}) = 1$ and $b_i \leq \hat{p} \leq B_i$ in addition $O(B_i) = O(T)$ time. Let $S_i = \{ \hat{p} \mid u_{ik}(\hat{p}) = 1, b_i \leq \hat{p} \leq B_i \}$ and let $s_i = |S_i|$. Then it holds $s_i = O(T)$, and for each $\hat{p} \in S_i$, we can
construct a corresponding partial solution $\tilde{x}_i$ with $f_i(\tilde{x}_i) = \tilde{p}$ in further additional $O(n)$ time. This means that a sequence $\tilde{x}_i^{(1)}, \tilde{x}_i^{(2)}, \ldots, \tilde{x}_i^{(k)}$ of $s_i (= |S_i|)$ partial solutions can be obtained in $O(nT)$ time by procedure Partial_DP (for items of the $i$-th type) such that it satisfies

$$b_i \leq f_i^{(1)} = f_i(\tilde{x}_i^{(1)}) < f_i(\tilde{x}_i^{(2)}) < \cdots < f_i(\tilde{x}_i^{(k)}) \leq B_i.$$  

(23)

The priority recording variable $u_k(p)$ takes the maximum of two additional values $\sigma_k(p)$ and $\tau_k(p)$ in Eq. (22), and for each $\tilde{p} \in S_i$ and for a corresponding partial solution $\tilde{x}_i$ with $f_i(\tilde{x}_i) = \tilde{p}$, the total priority $g(\tilde{x}_i)$ of chosen items of the $i$-th type in the $\tilde{x}_i$ is the maximum among ones of partial solutions $x_i$ with $f_i(x_i) = \tilde{p}$. Hence, we also see that it holds $v_k(p) = g(\tilde{x}_i)$.

In Table 1, we show the behavior of procedure Partial_DP for an example with five items of the first type. The entries $u_k(p)$ and $v_k(p)$ are computed by Eqs. (17)–(22). For $k = 1$, the setting of additional 0-1 variables $x_{1k}(p)$ for the backtracking process is obvious. For $k \geq 2$, each entry $x_{1k}(p)$ is set to be zero if it holds $u_{k+1}(p) = 1$ due to $u_{k+1}(p) = 1$, while it is set to be one if it holds $u_{k+1}(p) = 1$ due to $w_{ik} > 0, p - w_{ik} > 0$ and $u_{k+1}(p) = 1$. If the conditions in Eq. (19) are both satisfied, the $x_{1k}$ depends upon the choice of the corresponding $v_k(p)$ in Eq. (22). That is, it is set to be zero if $v_k(p) = \sigma_k(p)$, while it is set to be one if $v_k(p) = \tau_k(p)$. Equation (22) breaks ties arbitrarily. For the first type of items, we have six partial solutions $\tilde{x}_1^{(1)} = (1, 0, 1, 0, 0), \tilde{x}_1^{(2)} = (1, 0, 0, 1, 0), \tilde{x}_1^{(3)} = (1, 0, 1, 1, 0), \tilde{x}_1^{(4)} = (1, 0, 0, 1, 1), \tilde{x}_1^{(5)} = (1, 0, 1, 1, 1)$ and $\tilde{x}_1^{(6)} = (1, 0, 1, 1, 1)$, since $S_1 = \{p \mid u_k(p) = 1, b_1 = 9 \leq \tilde{p} \leq B_1 = 21\} = \{10, 12, 14, 16, 18, 20\}$ (see Eq. (23)).

### Table 1. Behavior of Partial_DP for an Example

| $p$ | $u_{12}$ | $u_{13}$ | $u_{14}$ | $u_{15}$ | $u_{16}$ | $u_{17}$ | $u_{18}$ | $u_{19}$ | $u_{20}$ |
|-----|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| 0   | 1        | 0        | 0        | 0        | 1        | 0        | 0        | 1        | 0        |
| 1   | 0        | -1       | 0        | -1       | 0        | -1       | 0        | -1       | 0        |
| 2   | 0        | -1       | 0        | -1       | 0        | -1       | 0        | -1       | 0        |
| 3   | 0        | -1       | 0        | -1       | 0        | -1       | 0        | -1       | 0        |
| 4   | 0        | -1       | 0        | -1       | 0        | -1       | 0        | -1       | 0        |
| 5   | 0        | -1       | 0        | -1       | 0        | -1       | 0        | -1       | 0        |
| 6   | 0        | -1       | 0        | -1       | 0        | -1       | 0        | -1       | 0        |
| 7   | 0        | -1       | 0        | -1       | 0        | -1       | 0        | -1       | 0        |
| 8   | 1        | 5        | 1        | 5        | 0        | 1        | 5        | 0        | 1        |
| 9   | 0        | -1       | 0        | -1       | 0        | -1       | 0        | -1       | 0        |
| 10  | 0        | -1       | 0        | -1       | 0        | -1       | 0        | -1       | 0        |
| 11  | 0        | -1       | 0        | -1       | 0        | -1       | 0        | -1       | 0        |
| 12  | 0        | -1       | 0        | -1       | 0        | -1       | 0        | -1       | 0        |
| 13  | 0        | -1       | 0        | -1       | 0        | -1       | 0        | -1       | 0        |
| 14  | 0        | -1       | 0        | -1       | 0        | -1       | 0        | -1       | 0        |
| 15  | 0        | -1       | 0        | -1       | 0        | -1       | 0        | -1       | 0        |
| 16  | 0        | -1       | 0        | -1       | 0        | -1       | 0        | -1       | 0        |
| 17  | 0        | -1       | 0        | -1       | 0        | -1       | 0        | -1       | 0        |
| 18  | 0        | -1       | 0        | -1       | 0        | -1       | 0        | -1       | 0        |
| 19  | 0        | -1       | 0        | -1       | 0        | -1       | 0        | -1       | 0        |
| 20  | 0        | -1       | 0        | -1       | 0        | -1       | 0        | -1       | 0        |
| 21  | 0        | -1       | 0        | -1       | 0        | -1       | 0        | -1       | 0        |

$m = 2, n = 5, T = 13, b_1 = 1, b_{max} = 20$.

The daggers † indicate the backtracking process for $\tilde{x}_i^{(1)} = (1, 0, 1, 0, 0)$.

### 3.3. Integrating Partial Solutions

For a given instance of problem MIXTURE_LEXICO, suppose that we have obtained a sequence $\tilde{x}_i^{(1)}, \tilde{x}_i^{(2)}, \ldots, \tilde{x}_i^{(k)}$ of $s_i$ partial solutions of PARTIAL_LEXICO (for items of the $i$-th type) in Eq. (23) for every item type by applying procedure Partial_DP (for items of the $i$-th type). For each $i = 1, 2, \ldots, m$, let $X_i = \{\tilde{x}_i^{(1)}, \tilde{x}_i^{(2)}, \ldots, \tilde{x}_i^{(k)}\}$ denote the set of the $s_i$ partial solutions of PARTIAL_LEXICO (for items of the $i$-th type). The computation requires $O(m) \times O(nT) = O(mnT)$ time to obtain $X_i$ for all $i = 1, 2, \ldots, m$.

We then define the set of all combinations of $m$ partial solutions covering over all the types of items by $X = \times X_1 \times X_2 \times \cdots \times X_m$. Each combination in $X$ corresponds to a solution $\tilde{x}$ of problem MIXTURE_LEXICO, and the set $X$ contains an optimal solution $x^*$ by Lemma 1. Since it holds $|X| = s = O(T)$, we have $|X| = O(T^m)$. It is obvious that for a combination $\tilde{x} = (\tilde{x}_1; \tilde{x}_2; \cdots; \tilde{x}_m)$ in $X$, the objective function values $f(\tilde{x})$ and $g(\tilde{x})$ can be known in $O(m)$ time since
the sum weight \( f_i(x_i) \) and sum priority \( g_i(x_i) \) have already been computed by the dynamic programming procedure for \( i = 1, 2, \ldots, m \). Hence, it suffices to find the best combination among the \( O(T^m) \) combinations in \( X \) from the viewpoint of the lexicographic bi-criteria (see Eqs. (2) and (3)), checking the target weight constraint (see Eq. (4)). It takes \( O(mT^m) \) time. Therefore, we have the following lemma:

**Lemma 2.** Problem \textsc{Mixture_lexico} can be solved in \( O(mnT + mT^m) \) time.

### 3.4. Applying a Linear Search Technique

In this subsection, we concentrate our attention on the case of \( m = 2 \). For notational simplicity, let \( \lambda = (\lambda_1, \lambda_2, \ldots, \lambda_{s_1}) \) and \( \mu = (\mu_1, \mu_2, \ldots, \mu_{s_2}) \) denote two (non-empty) lists of non-negative integers such that

\[
\begin{align*}
\lambda_1 &< \lambda_2 < \cdots < \lambda_{s_1} \leq B_1, \\
\mu_1 &< \mu_2 < \cdots < \mu_{s_2} \leq B_2, \\
\lambda_1 + \mu_1 &\geq T,
\end{align*}
\]

and \( \lambda_1 \mu_1 \geq T \). We are asked to find a pair \( \{\lambda_1, \mu_1\} \) such that it meets \( \lambda_1 + \mu_1 \geq T \), and minimizes \( \lambda_1 + \mu_1 \).

Suppose that it holds \( \lambda_1 + \mu_1 < T \); otherwise, \( \{\lambda_f, \mu_f\} = \{\lambda_1, \mu_1\} \).

The following linear search procedure finds a required pair \( \{\lambda_f, \mu_f\} \) in \( O(s_1 + s_2) \) time, where the incumbent value of the sum of two integers is recorded in parameter \( z \), and the search counters for lists \( \lambda \) and \( \mu \) are denoted by \( c_1 \) and \( c_2 \), respectively.

**Procedure BLS (Base of Linear Search)**

**Step 1** (Initialization): Set \( z := +\infty, c_1 := 1 \) and \( c_2 := s_2 \).

**Step 2** (Termination Test): If it holds either \( c_1 > s_1 \) or \( c_2 < 1 \), output the pair \( \{\lambda_f, \mu_f\} \), and the searching process is terminated.

**Step 3** (Checking the Target Weight Constraint): If the current pair satisfies \( \lambda_{c_1} + \mu_{c_2} \geq T \), go to Step 4. Otherwise, set \( c_1 := c_1 + 1 \) and return to Step 2.

**Step 4** (Updating the Best Pair): If the current pair satisfies \( \lambda_{c_1} + \mu_{c_2} < z \), set \( z := \lambda_{c_1} + \mu_{c_2}, f^* := c_1 \) and \( f^* := c_2 \). Set \( c_2 := c_2 - 1 \) and return to Step 2.

By the above procedure, we have finished the preparation for applying a linear search technique to the case of \( m = 2 \), and we now move to the original description of the food packing problem. Recall that the sequence \( z_1^{(1)}, z_2^{(1)}, \ldots, z_{s_1}^{(1)} \) of \( s_1 \) partial solutions of \textsc{Partial_lexico} (for items of the \( i \)-th type) in Eq. (23) has been sorted in the increasing order of their values of function \( f_i \) for both \( i = 1 \) and \( i = 2 \). For problem \textsc{Mixture_lexico} with \( m = 2 \), we can utilize the property to find the best combination of two partial solutions covering the two types of items by applying the linear search technique to the two lists of partial solutions in \( O(s_1 + s_2) \) time, instead of checking all \( O(T^2) \) combinations in \( X_1 \times X_2 \). It is also easy to incorporate the second objective, i.e., the total priority, in procedure BLS. Therefore, together with Lemma 2, we obtain the following theorem:

**Theorem 1.** Problem \textsc{Mixture_lexico} with \( m = 2 \) can be solved in \( O(nT) \) time.

### 4. Numerical Results

In this section, we demonstrate the empirical performance of the proposed \( O(nT) \) time algorithm for problem \textsc{Mixture_lexico} with \( m = 2 \). The instances to be tested are randomly generated as follows:

- The number of hoppers for each type of items: \( n \in [10, 20, 30, 40] \).
- Integer weights: \( w_i \) values are uniformly random integers either in \( [35, 65] \) or in \( [70, 130] \).
- Target weight: \( T \in (400, 800, 1200) \).
- Indispensable weights: \( h_1, h_2 \in [0.375T, 0.4375T, 0.5T] \).
- The number of iterations of mixture packing operation per operation run: \( \ell_{\text{max}} = 10000 \).

The program is written in C, and it is run on a desktop personal computer with Windows 8 (64bit), Intel Core i7 3770 CPU (3.40GHz), 500GB HDD and 4GB memory. Note that by giving a constant priority to each item in the proposed \( O(nT) \) time algorithm, we can equivalently solve the instances of problem \textsc{Mixture_primal}.

In all the tables, each of the data indicates the mean value for ten operating runs of producing \( \ell_{\text{max}} \) packages (and hence 10000×10 packages are produced to obtain each of the data). The notations used in the tables have the following meanings:
CPU: The execution time of an algorithm required to obtain an optimal solution at each mixture packing operation.

NET: The mean value of the total weight over \( t_{\text{max}} \) packages.

\( D_{\text{max}} \): The maximum duration over all items thrown into the automated food packing system during an operating run.

\( D_{\text{mean}} \): The mean duration over all items thrown into the automated food packing system during an operating run.

Table 2 shows the execution times of the previous \( O(nT^2) \) time algorithm and the proposed \( O(nT) \) time algorithm. In the implementation, the algorithms include the dynamic programming procedures and some initializing steps as well to update the set of current 2n items at each mixture packing operation. We observe that the proposed \( O(nT) \) time algorithm improves not only the theoretical time complexity, but also reduces the actual execution time significantly. Typical automated food packing systems for a single type of items possess around twenty weighing hoppers (Ishida Co., Ltd., 2013), and the fastest one produces about 200 packages per minute at the maximum, i.e., approximately 300 milliseconds per packing operation, which spends most time in measuring the weights of next new items accurately, and only a few milliseconds may be left for choosing a subset of current items at each packing operation. From the practical viewpoint, the improvement of the proposed \( O(nT) \) time algorithm is important. Note that for the case of \( T = b_1 + b_2 \), it is not necessary for the proposed \( O(nT) \) time algorithm to call the linear search procedure.

Table 2  Execution Time Comparison between the Previous and Proposed Algorithms

| CPU [msec] | \( n \) | \( O(nT^2) \) Time Algorithm | \( O(nT) \) Time Algorithm |
|------------|--------|-----------------------------|-----------------------------|
| \( nT = 10 \) | \( O(nT^2) \) | 22 | 0.08 |
| \( nT = 20 \) | \( O(nT^2) \) | 55 | 0.18 |
| \( nT = 30 \) | \( O(nT^2) \) | 91 | 0.29 |
| \( nT = 40 \) | \( O(nT^2) \) | 127 | 0.38 |

\( w_{ik} \in [35, 65], T = 400, b_1 = 150, b_2 = 150 \)

| CPU [msec] | \( n \) | \( O(nT^2) \) Time Algorithm | \( O(nT) \) Time Algorithm |
|------------|--------|-----------------------------|-----------------------------|
| \( nT = 10 \) | \( O(nT^2) \) | 18 | 0.07 |
| \( nT = 20 \) | \( O(nT^2) \) | 45 | 0.17 |
| \( nT = 30 \) | \( O(nT^2) \) | 74 | 0.26 |
| \( nT = 40 \) | \( O(nT^2) \) | 105 | 0.35 |

\( w_{ik} \in [35, 65], T = 400, b_1 = 175, b_2 = 175 \)

| CPU [msec] | \( n \) | \( O(nT^2) \) Time Algorithm | \( O(nT) \) Time Algorithm |
|------------|--------|-----------------------------|-----------------------------|
| \( nT = 10 \) | \( O(nT^2) \) | 14 | 0.06 |
| \( nT = 20 \) | \( O(nT^2) \) | 36 | 0.15 |
| \( nT = 30 \) | \( O(nT^2) \) | 59 | 0.23 |
| \( nT = 40 \) | \( O(nT^2) \) | 82 | 0.32 |

\( w_{ik} \in [35, 65], T = 400, b_1 = 200, b_2 = 200 \)

Table 3 shows the execution time of the proposed \( O(nT) \) time algorithm for larger target weights \( T = 800 \) and \( T = 1200 \). The execution time is at most 1 [msec] except for the entry with \( n = 40, b_1 = b_2 = 450 \) and \( T = 1200 \), and it seems to change almost linearly either in the number \( n \) of items of each type or the target weight \( T \).

Table 4 shows the results on the total weight of chosen items in an optimal solution by the proposed \( O(nT) \) time algorithm. For both problems MIXTURE\_PRIMAL and MIXTURE\_LEXICO, the total weight in each entry with \( T > b_1 + b_2 \) is closer to the target weight \( T = 800 \) than that in the corresponding entry with \( T = b_1 + b_2 \). We see the reason when, for example, we recall that for the instances with \( T = b_1 + b_2 \), the proposed \( O(nT) \) time algorithm does not call the linear search procedure. However anyway, the relative difference from the target weight on the average is at most \((804.8-800)/800 = 0.6\%\).

Table 5 shows the maximum duration \( D_{\text{max}} \) and mean duration \( D_{\text{mean}} \) obtained by the proposed \( O(nT) \) time algorithm. Each operating run consists of 10000 iterations of mixture packing operation (i.e., \( t_{\text{max}} = 10000 \)). The \( D_{\text{max}} \) is from 2968 to 9993 on the average for the instances of problem MIXTURE\_PRIMAL with \( n \in \{20, 30, 40\} \). On the other hand, the \( D_{\text{max}} \) is less than 76 on the average for the corresponding instances of problem MIXTURE\_LEXICO. Similar results with respect to the previous non-mixture algorithms have been observed (Karuno, et al., 2007, Karuno, et al., 2010, Imahori, et al., 2011). That is, the second objective of maximizing the total priority is effective upon reducing the durations in hoppers of items also for the mixture packing operation.
Table 3  Execution Time of the Proposed Algorithm with the Numbers of Hoppers and Target Weights

| CPU [msec] | Indispensable Weights, $b_1$ and $b_2$ |
|-----------|----------------------------------------|
| $n$       | 300         | 350         | 400         |
| 10        | 0.14        | 0.13        | 0.11        |
| 20        | 0.34        | 0.31        | 0.27        |
| 30        | 0.54        | 0.49        | 0.43        |
| 40        | 0.73        | 0.67        | 0.58        |

$w_{ik} \in [70, 130], T = 800$

| $n$       | 450         | 525         | 600         |
|-----------|----------------------------------------|
| 10        | 0.19        | 0.17        | 0.15        |
| 20        | 0.48        | 0.44        | 0.40        |
| 30        | 0.77        | 0.70        | 0.63        |
| 40        | 1.05        | 0.96        | 0.86        |

$w_{ik} \in [70, 130], T = 1200$

Table 4  Total Weight of a Mixture Package by the Proposed Algorithm

**NET**

| MIXTURE_PRIMAL | Indispensable Weights, $b_1$ and $b_2$ |
|----------------|----------------------------------------|
| $n$            | 300         | 350         | 400         |
| 10             | 800.0       | 800.0       | 804.8       |
| 20             | 800.0       | 800.0       | 803.5       |
| 30             | 800.0       | 800.0       | 803.4       |
| 40             | 800.0       | 800.0       | 803.4       |

$w_{ik} \in [70, 130], T = 800$

| MIXTURE_LEXICO | Indispensable Weights, $b_1$ and $b_2$ |
|----------------|----------------------------------------|
| $n$            | 300         | 350         | 400         |
| 10             | 800.0       | 800.0       | 804.3       |
| 20             | 800.0       | 800.0       | 801.5       |
| 30             | 800.0       | 800.0       | 800.7       |
| 40             | 800.0       | 800.0       | 800.4       |

$w_{ik} \in [70, 130], T = 800$

Table 5  Maximum and Mean Durations

| MIXTURE_PRIMAL | Indispensable Weights, $b_1$ and $b_2$ |
|----------------|----------------------------------------|
| $n$            | 300         | 350         | 400         |
| 10             | 498 [2.5]   | 144 [2.5]   | 88 [2.5]    |
| 20             | 9186 [5.0]  | 2968 [5.0]  | 4172 [5.0]  |
| 30             | 9897 [7.5]  | 8736 [7.5]  | 9274 [7.4]  |
| 40             | 9993 [10.0] | 9977 [10.0] | 9949 [9.9]  |

$w_{ik} \in [70, 130], T = 800$

| MIXTURE_LEXICO | Indispensable Weights, $b_1$ and $b_2$ |
|----------------|----------------------------------------|
| $n$            | 300         | 350         | 400         |
| 10             | 5.1 [2.5]   | 5.8 [2.5]   | 39.9 [2.5]  |
| 20             | 9.0 [5.0]   | 9.0 [5.0]   | 61.4 [5.0]  |
| 30             | 12.0 [7.5]  | 12.3 [7.5]  | 70.7 [7.5]  |
| 40             | 15.6 [10.0] | 15.8 [10.0] | 75.2 [9.9]  |

$w_{ik} \in [70, 130], T = 800$

5. Conclusions

In this paper, we considered a lexicographic bi-criteria combinatorial optimization problem, which is a mathematical model of the food mixture packing performed by so-called automatic combination weighers. For the case in which the
number of types of items is two, i.e., \( m = 2 \), an \( O(nT^2) \) time dynamic programming algorithm had been designed. In this paper, we first constructed an \( O(mnT + mT^m) \) time algorithm for an arbitrary \( m \geq 2 \), and then improved the previous time complexity to \( O(nT) \) for the case of \( m = 2 \) by applying a linear search technique. We also conducted numerical experiments, and observed that the proposed \( O(nT) \) time algorithm ran significantly faster than the previous \( O(nT^2) \) time algorithm.

### Acknowledgment

This research was partially supported by a Grant-in-Aid for Scientific Research (C) of the Japan Society for the Promotion of Science (JSPS). We gave a presentation on a part of the theoretical results at International Symposium on Scheduling 2013 (ISS 2013), JSME/SSJ, July 2013, Tokyo, Japan (Imahori and Karuno, 2013). This refined version includes an improved upper bound on the optimal sum weights and a new computational section.

### Appendix

The previous \( O(nT^2) \) time algorithm for problem MIXTURE.LEXICO with \( m = 2 \) is a straightforward modification of an \( O(nT^2) \) time dynamic programming algorithm for the duplex food packing problem (Imahori, et al., 2010, Imahori, et al., 2012).

For \( k = 1, 2, \ldots, n \), \( p = 0, 1, \ldots, B_1 \) and \( q = 0, 1, \ldots, B_2 \), the 0-1 state variables \( u_k(p, q) \) are defined as follows:

\[
u_k(p, q) = \begin{cases} 
1 & \text{if } (p = 0 \text{ and } q = 0), \\
& \text{if } (p = 0 \text{ and } q = w_{21}), \\
& \text{if } (p = w_{11} \text{ and } q = 0), \\
& \text{if } (p = w_{11} \text{ and } q = w_{21}), \\
0 & \text{otherwise},
\end{cases}
\]

(24)

\[
u_1(p, q) = \begin{cases} 
0 & \text{if } (p = 0 \text{ and } q = 0), \\
\gamma_{21} & \text{if } (p = 0 \text{ and } q = w_{21}), \\
\gamma_{11} & \text{if } (p = w_{11} \text{ and } q = 0), \\
\gamma_{11} + \gamma_{21} & \text{if } (p = w_{11} \text{ and } q = w_{21}), \\
-1 & \text{otherwise},
\end{cases}
\]

(25)

and for \( k = 2, 3, \ldots, n \), \( p = 0, 1, \ldots, B_1 \) and \( q = 0, 1, \ldots, B_2 \),

\[
u_k(p, q) = \begin{cases} 
1 & \text{if } u_{k-1}(p, q) = 1, \\
& \text{if } (w_{2k} > 0, q - w_{2k} \geq 0 \text{ and } u_{k-1}(p, q - w_{2k}) = 1), \\
& \text{if } (w_{1k} > 0, p - w_{1k} \geq 0 \text{ and } u_{k-1}(p - w_{1k}, q) = 1), \\
& \text{if } (w_{1k} > 0, w_{2k} > 0, p - w_{1k} \geq 0, q - w_{2k} \geq 0 \text{ and } u_{k-1}(p - w_{1k}, q - w_{2k}) = 1), \\
0 & \text{otherwise},
\end{cases}
\]

(26)

\[
\phi_k(p, q) = \begin{cases} 
u_{k-1}(p, q) & \text{if } u_{k-1}(p, q) = 1, \\
-1 & \text{otherwise},
\end{cases}
\]

(27)

\[
\psi_k(p, q) = \begin{cases} 
u_{k-1}(p, q - w_{2k}) + \gamma_{2k} & \text{if } (w_{2k} > 0, q - w_{2k} \geq 0 \text{ and } u_{k-1}(p, q - w_{2k}) = 1), \\
-1 & \text{otherwise},
\end{cases}
\]

(28)
\[ \sigma_k(p, q) = \begin{cases} v_{k-1}(p - w_{1k}, q) + \gamma_{1k} & \text{if } (w_{1k} > 0, p - w_{1k} \geq 0 \text{ and } u_{k-1}(p - w_{1k}, q) = 1), \\ -1 & \text{otherwise}, \end{cases} \]

(29)

\[ \tau_k(p, q) = \begin{cases} v_{k-1}(p - w_{1k}, q - w_{2k}) + \gamma_{1k} + \gamma_{2k} & \text{if } (w_{1k} > 0, p - w_{1k} \geq 0, q - w_{2k} \geq 0 \text{ and } u_{k-1}(p - w_{1k}, q - w_{2k}) = 1), \\ -1 & \text{otherwise}, \end{cases} \]

(30)

\[ v_k(p, q) = \max \{ \phi_k(p, q), \psi_k(p, q), \sigma_k(p, q), \tau_k(p, q) \}. \]

(31)

The computation of all the \(u_k(p, q), v_k(p, q), \psi_k(p, q), \sigma_k(p, q) \) and \(\tau_k(p, q)\) requires \(O(n \times B_1 \times B_2) = O(nT^2)\) time (see Eq. (16)). After the computation, we find a minimum of \(p + q\) such that it satisfies \(u_0(p, q) = 1, p + q \geq T, p \geq b_1\) and \(q \geq b_2\). Let \(\hat{p} + \hat{q}\) be the minimum. Then we can easily see that \(f^* = \hat{p} + \hat{q}\), where we regard \(f^*_1 = \hat{p}\) and \(f^*_2 = \hat{q}\).

Let \(S = \{(\hat{p}, \hat{q}) \mid u_0(\hat{p}, \hat{q}) = 1, \hat{p} + \hat{q} = f^* (\geq T), b_1 \leq \hat{p} \leq B_1, b_2 \leq \hat{q} \leq B_2\}\) denote the set of pairs of \(\hat{p}\) and \(\hat{q}\), and let \((p^*, q^*) \in S\) denote a pair such that it satisfies \(v_{\ell}(p^*, q^*) \geq v_0(\hat{p}, \hat{q})\) for any pair \((\hat{p}, \hat{q}) \in S\). Note that it holds \(|S| = O(B_1 \times O(B_2) = O(T^2)\). We can find the minimum \(p^* + q^*\) in \(O(T^2)\) time by checking \(O(T^2)\) variables of \(u_\ell(p, q)\) and \(v_\ell(p, q)\) with \(b_1 \leq p \leq B_1\) and \(b_2 \leq q \leq B_2\), which have already been computed.

From the above discussion, it is obvious that the time complexity of the dynamic programming is evaluated as \(O(nT^2)\) time. The correctness of the recursive and consecutive procedure can be shown by the similar argument to that of Imahori et al. (2010).

References

Garey, M.R. and Johnson, D.S., Computers and Intractability: A Guide to the Theory of NP-Completeness (1979), W.H. Freeman, San Francisco.

Imahori, S. and Karuno, Y., Pseudo-polynomial time algorithms for food mixture packing by automatic combination weighers, Proceedings of International Symposium on Scheduling 2013 (ISS 2013), JSME No. 13-202 (2013), pp. 59–64.

Imahori, S., Karuno, Y., Nagamochi, H. and Wang, X., Optimization problems and algorithms in double-layered food packing systems, The Fifth IEEE Conference on Industrial Informatics (INDIN 2010), IEEE Xplore (2010), pp. 810–815.

Imahori, S., Karuno, Y. and Yoshimoto, Y., Duplex and quasi-duplex operations in automated food packing systems, The Fifth IEEE/SICE International Symposium on System Integration (SII 2012), IEEE Xplore (2012), pp. 810–815.

Imahori, S., Karuno, Y. and Yoshimoto, Y., Dynamic programming algorithms for combinatorial food packing problems, International Journal of Biometrics, Vol. 3 (2011), pp. 228–245.

Ishida Co., Ltd., Weighing and Packaging, available from <http://www.ishida.co.jp/>., (accessed on 12 December, 2013) (in Japanese).

Kameoka, K., and Nakatani, M., Feed control criterion for a combination weigher and its effects, Transactions of the Society of Instrument and Control Engineers, Vol. 37 (2001), pp. 911–915 (in Japanese).

Kameoka, K., Nakatani, M. and Inui, N., Phenomena in probability and statistics found in a combinatorial weigher, Transactions of the Society of Instrument and Control Engineers, Vol. 36 (2000), pp. 388–394 (in Japanese).

Karuno, Y., Nagamochi, H. and Wang, X., Bi-criteria food packing by dynamic programming, Journal of the Operations Research Society of Japan, Vol. 50 (2007), pp. 376–389.

Karuno, Y., Nagamochi, H. and Wang, X., Optimization problems and algorithms in double-layered food packing systems, Journal of Advanced Mechanical Design, Systems, and Manufacturing, Vol. 4 (2010), pp. 605–615.

Morinaka, H., Automatic combination weigher for product foods, Journal of the Japan Society of Mechanical Engineers, Vol. 103 (2000), pp. 130–131 (in Japanese).