PSEUDO-POLYNOMIAL TIME ALGORITHMS FOR COMBINATORIAL FOOD MIXTURE PACKING PROBLEMS

SHINJI IMAHORI
Faculty of Science and Engineering
Chuo University
Kasuga 1-13-27, Bunkyo-ku, Tokyo 112-8551, Japan

YOSHIYUKI KARUNO*
Faculty of Mechanical Engineering
Kyoto Institute of Technology
Matsugasaki, Sakyo-ku, Kyoto 606-8585, Japan

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

(Communicated by Ruhul Sarker)

Abstract. A union $I = I_1 \cup I_2 \cup \cdots \cup I_m$ of $m$ sets of items is given, 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 $p_{ik}$. The food mixture packing problem to be discussed in this paper asks to find a union $I' = I'_1 \cup I'_2 \cup \cdots \cup I'_m$ of $m$ subsets of items so that for each $i = 1, 2, \ldots, m$, the sum weight of chosen items of the $i$-th type for $I'_i \subseteq I_i$ is no less than an integral indispensable bound $b_i$, and the total weight of chosen items for $I'$ is no less than an integral target weight $t$. 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. In this paper, the known time complexity $O(mnt + mt^2m)$ is improved to $O(mnt + mt^2)$ for an arbitrary $m \geq 3$ by a two-stage constitution algorithm with dynamic programming procedures. The improved time complexity figures out the weak NP-hardness of the food mixture packing problem.

1. Introduction. In this paper, we treat a combinatorial optimization problem arising in actual automated food packing systems known as automatic combination weighers or multi-head weighers (e.g., see Morinaka [12], Wurdemann et al. [13], and web sites [6, 14] of manufacturers). The automated food packing system possesses several hoppers with 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 recent mathematical model

2010 Mathematics Subject Classification. Primary: 90C27, 90C39; Secondary: 68Q25.

Key words and phrases. Combinatorial optimization, dynamic programming, pseudo-polynomial time algorithms, mathematical modeling, food packaging quality.

* Corresponding author: Yoshiyuki Karuno.
of the mixture packing operation (see Imahori and Karuno [2]) 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 $p_{ik}$. The problem asks to find a union $I' = I'_1 \cup I'_2 \cup \cdots \cup I'_m$ of $m$ subsets of items so that for each $i = 1, 2, \ldots, m$, the sum weight of chosen items of the $i$-th type for $I'_i \subseteq I_i$ is no less than an integral indispensable bound $b_i$, and the total weight of chosen items for $I'$ is no less than an integral target weight $t$. The latter is known as the target weight constraint, which is a hard constraint imposed on automatic combination weighers from a viewpoint of the service conscience (see Morinaka [12] again). The former is also a hard constraint of the mixture packing operation, which we call the indispensable bound 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 (e.g., see Kameoka and Nakatani [7]). In order to prevent such an undesirable situation, the priority may be introduced into each item (e.g., see Karuno et al. [9]), which is defined by a non-decreasing function of the duration in hopper of the item. Current items with longer durations in hoppers are expected 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 (e.g., see Imahori et al. [3], Imahori et al. [4], Karuno et al. [10], Karuno et al. [11]).

In the following section, the food packing problem is going to be described by a lexicographic bi-criteria 0-1 integer programming formulation, which is obviously NP-hard (e.g., see Garey and Johnson [1] for the theory of NP-hardness). Imahori and Karuno [2] have shown that the case of $m = 2$ can be solved in $O(nt)$ time, which implies that the case of $m = 2$ is not strongly NP-hard, but it is weakly NP-hard (also see Garey and Johnson [1]). The pseudo-polynomial time algorithm was constructed by combining an $O(mnt + mt^m)$ time algorithm for an arbitrary $m \geq 2$ with a linear search technique. Afterward, Imahori et al. [5] have proposed an improved implementation of the $O(nt)$ time algorithm for the case of $m = 2$. They concluded that the empirical performance for the case of $m = 2$ indicated its applicability to practical situation. Another mixture packing system
handling two types of items has also been considered by Kameoka et al. [8]. However, from the viewpoint of mathematical modeling, the packing operation may be suitable to be categorized into the class of non-mixture ones, since there is exactly one current item of the first type and it must be chosen.

In this paper, we show that for an arbitrary $m \geq 3$, the known time complexity $O(mnt + mt^2)$ can be improved into $O(mnt + mt^2)$ by a two-stage constitution algorithm with dynamic programming procedures. The proposed two-stage constitution algorithm utilizes a novel and simple $O(t)$ upper bound on the minimum total weight. The improved time complexity is polynomial in $m$ and $n$, which figures out the weak NP-hardness of the case for an arbitrary $m \geq 3$.

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 non-fictitious (i.e., actual) items. In result, it has totally $mn$ weighing hoppers. As in the previous section, for each $i = 1, 2, \ldots, m$, $I_i = \{I_{ik} \mid k = 1, 2, \ldots, n\}$ denotes a set of current $n$ items of the $i$-th type, and $I_{ik}$ denotes the current $k$-th item of the $i$-th type (see Fig. 1). Also, $I = \{I_{ik} \mid i = 1, 2, \ldots, m, k = 1, 2, \ldots, n\}$ denotes the set of current $mn$ items.

![Figure 1. Mixture packing operation with $mn$ weighing hoppers](image)

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:

- $I = \bigcup_{i=1}^{m} I_i$ : a union 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.
- $w_{ik}$ : a positive integer weight of non-fictitious item $I_{ik}$. It is assumed to be zero for any fictitious item $I_{ik}$.
- $p_{ik} (= d_{ik})$ : a positive integer priority of non-fictitious item $I_{ik}$. It is assumed to be zero for any fictitious item $I_{ik}$. 
• $b_i$: an indispensable bound for the $i$-th type items in each mixture package, which is assumed to be a non-negative integer.
• $t$: a target weight of each mixture package, which is assumed to be a positive integer.

For each $i = 1, 2, \ldots, m$ and set $I_i = \{I_{ik} \mid k = 1, 2, \ldots, n\}$ of items of the $i$-th type, let $x_i = (x_{i1}, x_{i2}, \ldots, x_{in})$ denote a 0-1 vector, where for $k = 1, 2, \ldots, n,$

$$x_{ik} = \begin{cases} 1 & \text{if item } I_{ik} \text{ is chosen,} \\ 0 & \text{otherwise,} \end{cases} \quad (1)$$

and let $x = (x_1; x_2; \cdots; x_m)$ denote the whole list of 0-1 vectors. The food packing problem at each mixture packing operation is formulated by using the list $x$ of $m$ 0-1 vectors as a solution. Note that a list $x$ consists of $mn$ elements, and we are sometimes going to mean the list $x$ by $mn$ 0-1 variables, instead of $m$ 0-1 vectors, in order to follow a conventional 0-1 integer programming formulation. Although the $x$ may also be viewed as an $m \times n$ matrix, we emphasize it as a list of $m$ 0-1 vectors. We do not utilize any mathematical feature of matrices for the $x$, and the algorithm to be designed in this paper is going to pick out the partial list of 0-1 vectors from the first type to the $i$-th type.

The lexicographic bi-criteria 0-1 integer programming formulation of the food packing problem at each mixture packing operation is provided as follows:

**MIXTURE\_LEXICO**

\[
\begin{align*}
\text{minimize} \quad & f(x) = \sum_{i=1}^{m} \sum_{k=1}^{n} w_{ik} x_{ik} \quad \text{as the primary objective}, \\
\text{maximize} \quad & g(x) = \sum_{i=1}^{m} \sum_{k=1}^{n} p_{ik} x_{ik} \quad \text{as the second objective}, \\
\text{subject to} \quad & \sum_{i=1}^{m} \sum_{k=1}^{n} w_{ik} x_{ik} \geq t, \quad (4) \\
& \sum_{k=1}^{n} w_{ik} x_{ik} \geq b_i, \quad i = 1, 2, \ldots, m, \quad (5) \\
& x_{ik} \in \{0, 1\}, \quad i = 1, 2, \ldots, m, \quad k = 1, 2, \ldots, n. \quad (6)
\end{align*}
\]

The target weight and indispensable bound 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 $x = (x_1; x_2; \cdots; x_m)$ 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 current 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.)
Let $\mu_i = \max_{1 \leq k \leq n} \{w_{ik}\}$ denote the maximum weight of an item of the $i$-th type. In order to omit some trivial cases, the followings are assumed:

$$0 < \mu_i \left( = \max_{1 \leq k \leq n} \{w_{ik}\} \right) < t, \quad i = 1, 2, \ldots, m, \quad (7)$$

$$b_i \leq \sum_{k=1}^{n} w_{ik}, \quad i = 1, 2, \ldots, m, \quad (8)$$

$$\sum_{i=1}^{m} b_i < t < \sum_{i=1}^{m} \sum_{k=1}^{n} w_{ik}. \quad (9)$$

As a trivial case, it should here be remarked that the case of $\sum_{i=1}^{m} b_i \geq t$ can be solved in $\sum_{i=1}^{m} O(n(b_i + \mu_i)) = O(mnt)$ time by considering $m$ instances of the (non-mixture) food packing problem independently (see Imahori et al. [3]). Further, let $w_{\text{max}} = \max_{1 \leq i \leq m} \{\mu_i\}$ denote the maximum weight among all $mn$ items in $\mathcal{I}$. Then, by the first assumption in the above (i.e., Eq. (7)), it satisfies that

$$w_{\text{max}} \left( = \max_{1 \leq i \leq m} \{\max_{1 \leq k \leq n} \{w_{ik}\}\} \right) < t. \quad (10)$$

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 that 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\_PRIMARY 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. Promising partial solutions for each type of items. As mentioned in the first section, we improve the known time complexity $O(mnt + mt^m)$ to $O(mnt + mt^2)$ for problem MIXTURE\_LEXICO with an arbitrary $m \geq 3$, which implies that the food packing problem is not strongly NP-hard, but it is weakly NP-hard. The algorithm to be designed in this paper can be seen as a two-stage constitution algorithm with dynamic programming procedures. Since the proposed algorithm utilizes promising partial solutions for each type of items as in our previous papers (see Imahori and Karuno [2], Imahori et al. [5]), we first in this section review the mathematical definition. Then we explain that the partial solutions are obtained by a dynamic programming procedure, which runs in $O(nt)$ time and maintains $O(t)$ promising partial solutions for each type of items. In the following sections, we are going to regard the $O(nt)$ time dynamic programming as the first stage procedure of the proposed algorithm.

3.1. Partial food packing problems. Let us consider the partial food packing problem for each type of items, which can be viewed as a non-mixture food packing problem. For each $i = 1, 2, \ldots, m$, we refer to the $i$-th 0-1 vector $x_i = (x_{i1}, x_{i2}, \ldots, x_{in})$ of a solution $x$ of problem MIXTURE\_LEXICO as a partial solution (for items of the $i$-th type). 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)**

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

A partial solution \( x_i = (x_{i1}, x_{i2}, \ldots, x_{im}) \) 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 the \( i \)-th type of problem MIXTURE-LEXICO. 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 \( x_i = x_i^+ \) denote a feasible solution that attains the minimum sum weight, i.e., \( f_i^+ = f_i(x_i^+) \), and maximizes the total priority among feasible solutions with the minimum sum weight \( f_i^+ \), i.e., it satisfies \( g_i(x_i^+) \geq g_i(\hat{x}_i) \) for any feasible solution \( \hat{x}_i \) of problem PARTIAL-LEXICO (for items of the \( i \)-th type) with \( f_i(\hat{x}_i) = f_i^+ \). We call the \( x_i^+ \) a *partially optimal* solution of the \( i \)-th type of problem MIXTURE-LEXICO.

### 3.2. Upper bounds on the optimal sum weight

Let \( x^* = (x_1^*; x_2^*; \ldots; x_m^*) \) be an optimal solution of problem MIXTURE-LEXICO, where \( x_i^* = (x_{i1}^*, x_{i2}^*, \ldots, x_{im}^*) \) denotes the \( i \)-th 0-1 vector for \( i = 1, 2, \ldots, m \), and let \( f_i^* = f_i(x_i^*) \) denote the *optimal sum weight* of the \( i \)-th item type. (Recall that we refer to \( f_i^* \) as the minimum sum weight for problem PARTIAL-LEXICO (for items of the \( i \)-th type).) Then the minimum total weight satisfies \( f(x^*) = f^* = f_1^* + f_2^* + \cdots + f_m^* \). 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). The following lemma has been proved by Imahori et al. [5]:

**Lemma 3.1.** For an instance of problem MIXTURE-LEXICO, and for each item type \( i = 1, 2, \ldots, m \), let \( x_i^+ \) denote a partially optimal solution of the \( i \)-th type with \( f_i(x_i^+) = f_i^+ \). Then an optimal solution \( x^* = (x_1^*; x_2^*; \ldots; x_m^*) \) where \( f_i(x_i^*) = f_i^* \) for each \( i = 1, 2, \ldots, m \) satisfies that

\[
b_i \leq f_i^+ \leq f_i^* < \left( t - \sum_{j=1}^{m} b_j + b_i \right) + \mu_i.
\]

Although the optimal sum weight \( f_i^* \) of chosen items of the \( i \)-th type is also bounded by \( \sum_{k=1}^{n} w_{ik} \) from the above, this trivial upper bound may not be evaluated as \( O(t) \). On the other hand, the upper bound provided in Lemma 3.1 is surely \( O(t) \) by assumption \( t > w_{\text{max}} \geq \mu_i \) (see Eq. (10)). At the first stage of the proposed
algorithm, the minimum of the two upper bounds, i.e.,
\[
\lambda_i = \min\left\{ t - \sum_{j=1}^{m} b_j + b_i, \mu_i - 1, \sum_{k=1}^{n} w_{ik} \right\},
\]

is going to be used. Note that it holds \(O(\lambda_i) = O(t)\).

Instead of reviewing the proof by Imahori et al. [5], let us construct a feasible solution for problem MIXTURE.LEXICO. From Eq. (9), we here suppose that \(t - \sum_{j=1}^{m} b_j > 0\). Then, it clearly holds \(t - \sum_{j=1}^{m} b_j > b_i\) for any \(i = 1, 2, \ldots, m\). Let
\[
S_1 = \left\{ i \mid \sum_{k=1}^{n} w_{ik} \geq t - \sum_{j=1}^{m} b_j + b_i, \ i \in \{1, 2, \ldots, m\} \right\},
\]

and let
\[
S_2 = \left\{ i \mid \sum_{k=1}^{n} w_{ik} < t - \sum_{j=1}^{m} b_j + b_i, \ i \in \{1, 2, \ldots, m\} \right\},
\]

which partitions the index set of \(m\) types into two disjoint subsets. For each \(i \in S_1\), let \(\kappa_i\) define the index of an item of the \(i\)-th type such that
\[
\kappa_i = \min\left\{ \kappa \in \{1, 2, \ldots, n\} \mid \sum_{k=1}^{\kappa} w_{ik} \geq t - \sum_{j=1}^{m} b_j + b_i \right\}, \ (i \in S_1),
\]

and for \(i \in S_2\), let
\[
\kappa_i = n, \quad (i \in S_2).
\]

Consider a 0-1 vector \(x_i = (x_{i1}, x_{i2}, \ldots, x_{in})\) for each \(i = 1, 2, \ldots, m\) defined by
\[
x_{ik} = \begin{cases} 1 & \text{for } k = 1, 2, \ldots, \kappa_i, \\ 0 & \text{for } k = \kappa_i + 1, \kappa_i + 2, \ldots, n, \end{cases}
\]

which is a feasible solution for problem PARTIAL.LEXICO (for items of the \(i\)-th type) since \(\sum_{k=1}^{\kappa_i} w_{ik} \geq t - \sum_{j=1}^{m} b_j + b_i\) for each \(i \in S_1\) (see Eq. (17)) and \(\sum_{k=1}^{\kappa_i} w_{ik} = \sum_{k=1}^{n} w_{ik} \geq b_i\) for each \(i \in S_2\) (see Eq. (8)). It is left to show that the solution \(x = (x_1; x_2; \ldots; x_m)\) consisting of the \(m\) 0-1 vectors defined by Eq. (19) satisfies the target weight constraint (see Eq. (4)). If \(S_1 = \emptyset\), the solution \(x\) chooses all \(mn\) items, and by assumption (see Eq. (9)), it obviously meets the target weight constraint for a given instance of problem MIXTURE.LEXICO. Now, suppose that \(S_1 \neq \emptyset\). Then we have
\[
\sum_{i=1}^{m} \sum_{k=1}^{\kappa_i} w_{ik} = \sum_{i \in S_1} \sum_{k=1}^{\kappa_i} w_{ik} + \sum_{i \in S_2} \sum_{k=1}^{n} w_{ik}
\]
\[
\geq \sum_{i \in S_1} \left( t - \sum_{j=1}^{m} b_j + b_i \right) + \sum_{i \in S_2} b_i
\]
\[
= |S_1| \left( t - \sum_{j=1}^{m} b_j \right) + \sum_{j=1}^{m} b_j
\]
\[
= t + \left( |S_1| - 1 \right) \left( t - \sum_{j=1}^{m} b_j \right)
\]
\[
\geq t,
\]
which implies that the solution \( x \) is feasible for an instance of problem MIXTURE_LEXICO. Note that the feasible solution \( x = (x_1; x_2; \cdots; x_m) \) satisfies

\[
\sum_{k=1}^{s_i} w_{ik} = \sum_{k=1}^{s_i-1} w_{ik} + w_{i,s_i} < \left( t - \sum_{j=1}^{m} b_j + b_i \right) + \mu_i, \quad i = 1, 2, \ldots, m. \tag{20}
\]

3.3. Dynamic programming procedure at the first stage. In order to obtain promising partially feasible solutions for each item type, a dynamic programming procedure is employed for problem PARTIAL_LEXICO (for items of the \( i \)-th type). Also in this paper, we call the dynamic programming procedure PartialDP (for items of the \( i \)-th type) (see Imahori et al. [5]).

Without loss of generality, we assume that the first item of each type is a non-fictitious item (i.e., it holds \( w_{i1} > 0 \) and \( p_{i1} > 0 \)). The following 0-1 state variables are defined for \( k = 1, 2, \ldots, n \) and \( q = 0, 1, \ldots, \lambda_i \) (n.b., the upper bound \( \lambda_i \) on the optimal sum weight of the \( i \)-th item type was defined in Eq. (16)):

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

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

The priority recording variables \( v_{ik}(q) \) for \( k = 1, 2, \ldots, n \) and \( q = 0, 1, \ldots, \lambda_i \) are also defined to maintain the conditionally maximum of total priority \( \sum_{j=1}^{k} p_{ij}x_{ij} \) of the partial 0-1 vector \( (x_{i1}, x_{i2}, \ldots, x_{ik}) \) with \( \sum_{j=1}^{k} w_{ij}x_{ij} = q \). Further, two kinds of temporal variables \( \theta_0 \) and \( \theta_1 \) are introduced to make each priority recording variable \( v_{ik}(q) \) identify the conditionally maximum for the partial 0-1 vector \( (x_{i1}, x_{i2}, \ldots, x_{ik}) \) with \( \sum_{j=1}^{k} w_{ij}x_{ij} = q \). (For each priority recording variable \( v_{ik}(q) \), at most two possible alternatives should be regarded as shown in Eqs. (23)–(26) later, and each \( v_{ik}(q) \) takes the maximum of \( \theta_0 \) and \( \theta_1 \).)

The 0-1 state variables and priority recording variables are computed by the following dynamic programming recursives: For \( q = 0, 1, \ldots, \lambda_i \),

\[
u_{i1}(q) = \begin{cases} 1 & \text{if } q \in \{0, w_{i1}\}, \\ 0 & \text{otherwise}, \end{cases} \tag{21}
\]

\[
u_{i1}(q) = \begin{cases} 0 & \text{if } q = 0, \\ p_{i1} & \text{if } q = w_{i1}, \\ -1 & \text{otherwise}, \end{cases} \tag{22}
\]

and for \( k = 2, 3, \ldots, n \) and \( q = 0, 1, \ldots, \lambda_i \),

\[
u_{ik}(q) = \begin{cases} 1 & \text{if } (u_{i,k-1}(q) = 1), \text{ or } \text{if } (w_{ik} > 0, q - w_{ik} \geq 0 \text{ and } u_{i,k-1}(q - w_{ik}) = 1), \\ 0 & \text{otherwise,} \end{cases} \tag{23}
\]

\[
u_{i1}(q) = \begin{cases} v_{i,k-1}(q) & \text{if } u_{i,k-1}(q) = 1, \\ -1 & \text{otherwise}, \end{cases} \tag{24}
\]

\[
u_{ik}(q) = \begin{cases} v_{i,k-1}(q - w_{ik}) + p_{ik} & \text{if } (w_{ik} > 0, q - w_{ik} \geq 0 \text{ and } u_{i,k-1}(q - w_{ik}) = 1), \\ -1 & \text{otherwise}, \end{cases} \tag{25}
\]

\[
u_{ik}(q) = \max \{\theta_0, \theta_1\}. \tag{26}
\]
The correctness is basically followed by Imahori et al. [3]. The condition of $w_{ik} > 0$ appearing in Eqs. (23) and (25) is put to exclude any fictitious item from the resulting solution (see Imahori and Karuno [2]). In an implementation of the dynamic programming procedure, it suffices to prepare an $O(1)$ space for maintaining the temporal variables $\theta_0$ and $\theta_1$ (see Imahori et al. [5]), since these variables are referred only for each $v_{ik}(q)$ in Eq. (26). Computing all variables $u_{ik}(q)$ and $v_{ik}(q)$ by Eqs. (21)–(26) obviously requires $O(n\lambda_i) = O(nt)$ time (see Eq. (16)).

After computing the 0-1 state variables $u_{ik}(q)$ and priority recording variables $v_{ik}(q)$ by Eqs. (21)–(26), all values of $q = \hat{q}$ such that $u_{in}(\hat{q}) = 1$ and $b_i \leq \hat{q} \leq \lambda_i$ can be found in additional $O(\lambda_i) = O(t)$ time. Let $s_i = \{\hat{q} \mid u_{in}(\hat{q}) = 1, b_i \leq \hat{q} \leq \lambda_i\}$ and let $s_i = |S_i|$. Then it holds $s_i = O(\lambda_i) = O(t)$, and for each $\hat{q} \in S_i$, we can construct a corresponding partial solution $\tilde{x}_i$ with $f_i(\tilde{x}_i) = \hat{q}$ in further additional $O(n)$ time.

Further, the priority recording variable $v_{ik}(q)$ takes the maximum of two temporal values $\theta_0$ and $\theta_1$ in Eq. (26). For each $\hat{q} \in S_i$ and for a corresponding partial solution $\tilde{x}_i$ with $f_i(\tilde{x}_i) = \hat{q}$, the total priority $g_i(\tilde{x}_i)$ of chosen items of the $i$-th type in the $\tilde{x}_i$ is the maximum among ones of partial solutions $x_i$ with $f_i(x_i) = \hat{q}$. Hence, we also see that it holds $v_{in}(\hat{q}) = g_i(\tilde{x}_i)$. We therefore obtain the following lemma [5]:

**Lemma 3.2.** Procedure PartialDP (for items of the $i$-th type) can obtain an ordered set $X_i = \{\tilde{x}_i^{(1)}, \tilde{x}_i^{(2)}, \ldots, \tilde{x}_i^{(s_i)}\}$ of $s_i$ promising partial solutions in $O(nt)$ time such that it satisfies

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

(27)

Note that it holds $x_i^+ = \tilde{x}_i^{(1)}$, and that no $O(s_i \log s_i) = O(t \log t)$ time sorting procedure is required for obtaining the ordered set $X_i$ of $s_i$ promising partial solutions.

3.4. Novel upper bounds on the minimum total weight. It should be remarked that a solution $x^+ = (x_1^+: x_2^+: \cdots : x_m^+)$ consisting of $m$ partially optimal solutions for all item types may be infeasible for a given instance of problem MIXTURE_LEXICO, since the solution $x^+$ may fail to meet the target weight constraint of Eq. (4). For the case in which solution $x^+$ meets the target weight constraint (and hence it is a feasible solution for a given instance of problem MIXTURE_LEXICO), we can prove the following lemma:

**Lemma 3.3.** For an instance of problem MIXTURE_LEXICO, and for each $i = 1, 2, \ldots, m$, let $x_i^+$ denote a partially optimal solution of the $i$-th type with $f_i(x_i^+) = f_i^+$. If it holds

$$\sum_{i=1}^m f_i^+ \geq t,$$

then $x^+ = (x_1^+: x_2^+: \cdots : x_m^+)$ is an optimal solution of problem MIXTURE_LEXICO.

**Proof.** Let $x^* = (x_1^*: x_2^*: \cdots : x_m^*)$ denote an optimal solution. By the feasibility of solution $x^+$ for the given instance of problem MIXTURE_LEXICO and the minimality of $f_i(x_i^+)$ for the $i$-th item type (see Lemma 3.1), we have $\sum_{i=1}^m f_i(x_i^+) = \sum_{i=1}^m f_i(x_i^*)$, and have

$$f_i(x_i^+) = f_i(x_i^*), \quad i = 1, 2, \ldots, m.$$
By the definition of $x_i^\perp$ and the above equation, we also obtain

$$g_i(x_i^\perp) \geq g_i(x_i^\ast), \quad i = 1, 2, \ldots, m.$$  

Hence, it holds

$$\sum_{i=1}^m g_i(x_i^\perp) \geq \sum_{i=1}^m g_i(x_i^\ast) = g^\ast,$$

which means that solution $x^\perp$ is also optimal for the given instance of problem MIXTURE_LEXICO. \hfill \Box

This property implies that problem MIXTURE_LEXICO is solved in $O(mnt)$ time by computing a partially optimal solution of every item type individually if solution $x^\perp$ consisting of $m$ partially optimal solutions meets the target weight constraint. Hence, we concentrate on the opposite case in which the solution $x^\perp$ consisting of $m$ partially optimal solutions fails to meet the target weight constraint. The following lemma provides another upper bound on the minimum total weight, which are going to be used at the second stage of the proposed algorithm.

**Lemma 3.4.** For an instance of problem MIXTURE_LEXICO, and for each $i = 1, 2, \ldots, m$, let $x_i^\perp$ denote a partially optimal solution of the $i$-th type with $f_i(x_i^\perp) = f_i^\perp$. If it holds

$$\sum_{i=1}^m f_i^\perp < t,$$

then an optimal solution $x^\ast$ satisfies

$$f(x^\ast) = f^\ast < t + w_{\text{max}},$$

where $w_{\text{max}}$ denotes the maximum among all $mn$ weights of items.

**Proof.** Let $x^\perp = (x_1^\perp; x_2^\perp; \ldots; x_m^\perp)$ be a solution consisting of $m$ partially optimal solutions of all item types, and let $I^\perp (\subset I)$ denote a subset of items that are chosen in solution $x^\perp$. Recall that solution $x^\perp$ is infeasible for the given instance of problem MIXTURE_LEXICO. By the assumption of Eq. (9), at least one item is not chosen for $I^\perp$.

We construct a feasible solution $x'$ with the solution $x^\perp$ as follows. First, we choose all the items in $I^\perp$ (i.e., we initially set as $x' = x^\perp$). Since solution $x^\perp$ has satisfied the indispensable bound constraint for every item type, it is left for us to regard the target weight constraint. We then add the remaining items one by one in an arbitrary order until the total weight of chosen items becomes at least the target weight $t$. Let $w$ be the weight of the last chosen item, then it holds $f(x') < t + w$, which implies the lemma statement. \hfill \Box

For notational convenience, we define the upper bound on the minimum total weight in the above lemma by

$$\lambda = t + w_{\text{max}} - 1,$$

which is going to be used in the following section where we combine $m$ partially feasible solutions of all item types. Note that it holds $O(\lambda) = O(t)$ (see Eq. (10)).
4. Combining partial solutions. Suppose that for an instance of problem MIXTURE\_LEXICO, the set \( X_i \) of \( s_i \) promising partial solutions of PARTIAL\_LEXICO (for items of the \( i \)-th type) in Eq. (27) has been obtained for every item type by applying procedure Partial\_DP (for items of the \( i \)-th type) at the first stage. The \( m \) times callings of procedure Partial\_DP require \( O(m) \times O(nt) = O(mnt) \) time. If the condition of Lemma 3.3 holds at the end of the first stage, we are no longer proceeding to the second stage since an optimal solution has been obtained.

We represent the set of all combinations of \( m \) partial solutions covering over all the types of items by \( \mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_m \). Each combination in \( \mathcal{X} \) corresponds to a solution \( x \) of problem MIXTURE\_LEXICO. We see that \( |\mathcal{X}| = O(t^m) \), and that the set \( \mathcal{X} \) contains an optimal solution \( x^* \) by Lemma 3.1.

In this section, we first explain a simple algorithm that enumerates all the possible \( O(t^m) \) combinations of promising partial solutions for constructing an optimal solution of problem MIXTURE\_LEXICO. Then, by utilizing the upper bound on the minimum total weight shown in Lemma 3.4, we design a dynamic programming procedure to find the best combination among the \( O(t^m) \) combinations in \( O(mt^2) \) time, which is the second stage of the proposed algorithm.

4.1. Enumeration of all combinations of promising partial solutions. It is obvious that for a combination \( x = (x_1; x_2; \cdots ; x_m) \) in \( \mathcal{X} \), the objective function values \( f(x) \) and \( g(x) \) 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 dynamic programming procedure Partial\_DP at the first stage for \( i = 1, 2, \ldots , m \). From the viewpoint of the lexicographic bi-criteria (see Eqs. (2) and (3)), finding the best solution \( x \) among the \( O(t^m) \) combinations in \( \mathcal{X} \) can be done in \( O(m) \times O(t^m) = O(mnt) \) time. By this consideration, Imahori and Karuno [2] showed that problem MIXTURE\_LEXICO for an arbitrary \( m \geq 2 \) can be solved in \( O(mnt + mt^m) \) time.

4.2. Dynamic programming procedure at the second stage. In this subsection, let \( x^* = (x_1^*; x_2^*; \cdots ; x_m^*) \) be an optimal solution again for an instance of problem MIXTURE\_LEXICO with \( f_i(x_i^*) = f_i^* \) for \( i = 1, 2, \ldots , m \). The optimal solution is feasible, and hence under the indispensable bound constraints, i.e., Eq. (5), together with Lemma 3.1, it satisfies

\[
\sum_{j=1}^{i} f_j^* \geq \sum_{j=1}^{i} f_j^+ \geq \sum_{j=1}^{i} b_j, \quad i = 1, 2, \ldots , m,
\]

and

\[
\sum_{j=i+1}^{m} f_j^* \geq \sum_{j=i+1}^{m} f_j^+ \geq \sum_{j=i+1}^{m} b_j, \quad i = 1, 2, \ldots , m.
\]

On the other hand, by Lemma 3.4 (i.e., \( f(x^*) = \sum_{i=1}^{m} f_i^* \leq \lambda \)), the optimal solution meets

\[
\sum_{j=1}^{i} f_j^* \leq \lambda - \sum_{j=i+1}^{m} f_j^* \leq \lambda - \sum_{j=i+1}^{m} f_j^+, \quad i = 1, 2, \ldots , m.
\]

Hence, we have

\[
\sum_{j=1}^{i} f_j^+ \leq \sum_{j=1}^{i} f_j^* \leq \lambda - \sum_{j=i+1}^{m} f_j^+ \quad (= O(t)), \quad i = 1, 2, \ldots , m. \quad (30)
\]
In consideration of the above lower and upper bounds on the i-th sum $\sum_{j=1}^{i} f_j^*$ of optimal sum weights for $i = 1, 2, \ldots, m$, we combine $m$ sets of promising partial solutions one by one from the first type to the $m$-th type in the following manner.

For $i = 1, 2, \ldots, m$ and for $q = \sum_{j=1}^{i} f_j^+ + \sum_{j=1}^{i} f_j^+ + 1, \ldots, \lambda - \sum_{j=i+1}^{m} f_j^+$ (see Eq. (30) for the range of parameter $q$), we define new 0-1 state variables $\alpha_i(q)$ as follows:

$$
\alpha_i(q) = \begin{cases} 
1 & \text{if there exists a partial list } (x_1; x_2; \cdots; x_i) \text{ of 0-1 vectors} \\
0 & \text{otherwise} 
\end{cases}, \quad (31)
$$

$$
\gamma_i(q) = \begin{cases} 
1 & \text{if } u_{1n}(q) = 1, \\
0 & \text{otherwise} 
\end{cases} \quad (32)
$$

and for $i = 2, \ldots, m$ and for $q = \sum_{j=1}^{i} f_j^+, \sum_{j=1}^{i} f_j^+ + 1, \ldots, \lambda - \sum_{j=i+1}^{m} f_j^+$,

$$
\alpha_i(q) = \begin{cases} 
1 & \text{if there exists an integer } r \text{ such that } f_i^+ \leq r \leq \lambda_i, \\
0 & \text{otherwise}, \\
\max \left\{ \gamma_{i-1}(q-r) + u_{in}(r) \mid f_i^+ \leq r \leq \lambda_i, \ u_{in}(r) = 1, \right\} & \text{if there exists an integer } r \text{ such that } f_i^+ \leq r \leq \lambda_i. 
\end{cases} \quad (33)
$$

$$
\gamma_i(q) = \begin{cases} 
1 & \text{if } u_{in}(r) = 1, \\
0 & \text{otherwise} 
\end{cases} \quad (34)
$$

We also define new priority recording variables $\gamma_i(q)$ for $i = 1, 2, \ldots, m$ and for $q = \sum_{j=1}^{i} f_j^+, \sum_{j=1}^{i} f_j^+ + 1, \ldots, \lambda - \sum_{j=i+1}^{m} f_j^+$ (see Eq. (30) again) to maintain the conditionally maximum of total priority $\sum_{j=1}^{i} g_j(x_j)$ of the partial list $(x_1; x_2; \cdots; x_i)$ of 0-1 vectors from the first type to the $i$-th type such that it meets $\sum_{j=1}^{i} f_j^+(x_j) = q$ and $f_j^+(x_j) \geq f_j^+$ for all $j = 1, 2, \ldots, i$.

The 0-1 state variables and priority recording variables are computed by the following dynamic programming recursives: For $q = f_1^+, f_2^+, \ldots, f_i^+, f_{i+1}^+, \ldots, f_m^+$, and for $q = \sum_{j=1}^{i} f_j^+, \sum_{j=1}^{i} f_j^+ + 1, \ldots, \lambda - \sum_{j=i+1}^{m} f_j^+$, we define new 0-1 state variables $\alpha_i(q)$ as follows:

$$
\alpha_i(q) = \begin{cases} 
1 & \text{if } u_{1n}(q) = 1, \\
0 & \text{otherwise} 
\end{cases} \quad (31)
$$

$$
\gamma_i(q) = \begin{cases} 
1 & \text{if } u_{1n}(q) = 1, \\
0 & \text{otherwise} 
\end{cases} \quad (32)
$$

and for $i = 2, \ldots, m$ and for $q = \sum_{j=1}^{i} f_j^+, \sum_{j=1}^{i} f_j^+ + 1, \ldots, \lambda - \sum_{j=i+1}^{m} f_j^+$,

$$
\alpha_i(q) = \begin{cases} 
1 & \text{if there exists an integer } r \text{ such that } f_i^+ \leq r \leq \lambda_i, \\
0 & \text{otherwise}, \\
\max \left\{ \gamma_{i-1}(q-r) + u_{in}(r) \mid f_i^+ \leq r \leq \lambda_i, \ u_{in}(r) = 1, \right\} & \text{if there exists an integer } r \text{ such that } f_i^+ \leq r \leq \lambda_i. 
\end{cases} \quad (33)
$$

$$
\gamma_i(q) = \begin{cases} 
1 & \text{if } u_{in}(r) = 1, \\
0 & \text{otherwise} 
\end{cases} \quad (34)
$$

We call this dynamic programming procedure CombineDP. Each computation of $\alpha_i(q)$ requires $O(\lambda_i) = O(t)$ time to seek for an appropriate integer $r$ in Eq. (33). In the seeking process, if more than one integer $r$ such that $f_i^+ \leq r \leq \lambda_i$, $u_{in}(r) = 1, q - r \geq \sum_{j=1}^{i} f_j^+$ and $a_{i-1}(q-r) = 1$ is found, the priority recording variable $\gamma_i(q)$ maintains the maximum of total priority with respect to such an integer $r$ by Eq. (34). Hence the time complexity of procedure CombineDP is evaluated by $\sum_{j=1}^{m} (O(\lambda) \times O(\lambda_i)) = O(mt^2)$.

Let $Q = \{ q \mid a_{m}(q) = 1, \ t \leq q \leq \lambda \}$ denote a set of promising possible values as the minimum total weight $f(x^*) = f^*$, and let $q_{\text{min}} = \min \{ q \in Q \}$. Then we see $q_{\text{min}} = f^*$. By an appropriate backtracking process, we can know the optimal sum weights $f_i^*$ for all item types $i = 1, 2, \ldots, m$ in additional $O(m)$ time. Further, by applying a backtracking procedure from the value $f_i^*$ to the computation process of PartialDP (for items of the $i$-th type) obtained at the first stage, we can construct
a partial solution \( x_i = x_i^* \) for each \( i = 1, 2, \ldots, m \) in additional \( O(n) \) time. This eventually delivers an optimal solution \( x^* = (x_1^*; x_2^*; \cdots; x_m^*) \).

The backtracking processes are dominated by the computation of recursives of procedures Partial\_DP and Combine\_DP in the sense of the time complexity. Therefore, we have the following result:

**Theorem 4.1.** Problem MIXTURE\_LEXICO for an arbitrary \( m \geq 3 \) can be solved in \( O(mnt + mt^2) \) time.

Obviously, the minimum total weight \( f^*(= f(x^*)) \) is also bounded by \( \sum_{i=1}^{m} \lambda_i = O(mt) \) from the above (see Lemma 3.1). By applying the framework of procedure Combine\_DP, this trivial upper bound can show that problem MIXTURE\_LEXICO for an arbitrary \( m \geq 3 \) is weakly NP-hard, while it makes the time complexity of the two-stage constitution algorithm \( O(mnt + m^2t^2) \), which is not linear in \( m \).

5. **Conclusions.** In this paper, we considered a lexicographic bi-criteria combinatorial optimization problem, which is a recent mathematical model of the combinatorial food mixture packing operation. For the case in which the number of item types is two (i.e., \( m = 2 \)), an \( O(nt) \) time dynamic programming algorithm had been designed, where \( n \) is the number of items of each type and \( t \) is the integral target weight of each package. This pseudo-polynomial time algorithm was constructed by combining an \( O(mnt + mt^m) \) time algorithm for an arbitrary \( m \geq 2 \) with a linear search technique. In this paper, we showed from the theoretical viewpoint of computation that the known time complexity \( O(mnt + mt^m) \) can be improved to \( O(mnt + mt^2) \) for an arbitrary \( m \geq 3 \) by a two-stage constitution algorithm with dynamic programming procedures, which is linear in both of \( m \) and \( n \). The proposed pseudo-polynomial time algorithm also figured out the weak NP-hardness of the food mixture packing problem for an arbitrary number \( m \geq 3 \) of item types.

We can obtain a faster implementation of the proposed two-stage constitution algorithm by modifying the dynamic programming procedure at the second stage (see Appendix A), although the theoretical time complexity still remains \( O(mnt + mt^2) \). For future research, it would be interesting to examine the empirical performance of the proposed algorithm.

**Appendix A. Critical index of item type.** In this extra section, we consider a modification of the \( O(mt^2) \) time procedure Combine\_DP presented in Section 4, although the time complexity remains the same. The key concept is a critical index of item type, which is going to be defined in the following.

Let \( x^* = (x_1^*; x_2^*; \cdots; x_m^*) \) be an optimal solution again for a given instance of problem MIXTURE\_LEXICO, and let \( f_i^* = f_i(x_i^*) \) for \( i = 1, 2, \ldots, m \). Further, let \( x_i^+ \) be a partially optimal solution of the \( i \)-th item type again for \( i = 1, 2, \ldots, m \), and let \( f_i^+ = f_i(x_i^+) \). The optimal solution \( x^* \) is feasible, and as in Section 4, it meets

\[
\sum_{j=1}^{i} f_j^* \geq \sum_{j=1}^{i} f_j^+ \geq \sum_{j=1}^{i} b_j, \quad i = 1, 2, \ldots, m,
\]

and

\[
\sum_{j=i+1}^{m} f_j^* \geq \sum_{j=i+1}^{m} f_j^+ \geq \sum_{j=i+1}^{m} b_j, \quad i = 1, 2, \ldots, m.
\]
Together with the assumption of Lemma 3.4 (i.e., $t > \sum_{j=1}^{m} f_j^+$), it also holds
\[
t - \sum_{j=i+1}^{m} b_j \geq t - \sum_{j=1}^{m} f_j^+ > \sum_{j=1}^{i} f_j^+ \geq \sum_{j=1}^{i} b_j, \quad i = 1, 2, \ldots, m.
\]

We now define the critical index of item type by
\[
i^* = \min\left\{ i \in \{1, 2, \ldots, m\} \mid \sum_{j=1}^{i} f_j^* \geq t - \sum_{j=i+1}^{m} f_j^\ast\right\}. \tag{35}
\]

The left term of the above inequality (i.e., $\sum_{j=1}^{i} f_j^*$) is a monotonic non-decreasing function with respect to index $i$, and the right term (i.e., $t - \sum_{j=i+1}^{m} f_j^\ast$) is also a monotonic non-decreasing function with respect to $i$. Further, it holds that $\sum_{j=1}^{m} f_j^\ast \geq t$ at the last index $m$ of item type, which denotes the target weight constraint (see Eq. (4)). Hence, the critical index $i^*$ of item type is well-defined, i.e., it surely satisfies $1 \leq i^* \leq m$. In addition, we remark that for each $i = i^*, i^*+1, \ldots, m$, it holds $\sum_{j=1}^{i} f_j^* \geq t - \sum_{j=i+1}^{m} f_j^\ast$.

**Lemma A.1.** For an instance of problem MIXTURE\_LEXICO, let $x^* = (x^*_1; x^*_2; \ldots; x^*_m)$ denote an optimal solution, and let $x_i^+$ denote a partially optimal solution of the $i$-th type for each $i = 1, 2, \ldots, m$. Let $i^*$ denote the critical index of item type, and we construct a solution $x' = (x'_1; x'_2; \ldots; x'_m)$ as
\[
x'_i = \begin{cases} x^*_i & \text{for } i = 1, 2, \ldots, i^*, \\ x^*_i & \text{for } i = i^*+1, i^*+2, \ldots, m. \end{cases} \tag{36}
\]

Then, the solution $x'$ is also an optimal solution of problem MIXTURE\_LEXICO.

**Proof.** By the definition of partially optimal solutions $x_i^+$, it holds $f_i(x_i^*) \geq f_i(x_i^+) \geq b_i$ for $i = 1, 2, \ldots, m$ (see Eq. (15)), and hence each partial solution $x'_i$ meets the indispensable bound constraint (see Eq. (5)). The total weight of solution $x'$ satisfies the target weight constraint (see Eq. (4)), i.e.,
\[
\sum_{i=1}^{m} f_i(x'_i) = \sum_{i=1}^{i^*} f_i(x_i^*) + \sum_{i=i^*+1}^{m} f_i(x_i^+)
\geq \left(t - \sum_{i=i^*+1}^{m} f_i^\ast\right) + \sum_{i=i^*+1}^{m} f_i^\ast
= t,
\]
which implies that the solution $x'$ is feasible for problem MIXTURE\_LEXICO. On the other hand, we have
\[
\sum_{i=1}^{m} f_i(x'_i) = \sum_{i=1}^{i^*} f_i(x_i^*) + \sum_{i=i^*+1}^{m} f_i(x_i^+)
\leq \sum_{i=1}^{i^*} f_i^* + \sum_{i=i^*+1}^{m} f_i^*
= f^*.
\]
Thus, the solution \( x' \) attains the minimum total weight \( f^* \). With the minimality of \( f_i^+ = f_i(x_i^+) \) for the \( i \)-th item type, these observations imply

\[
f_i(x_i^*) = f_i(x_i^+), \quad i = i^* + 1, i^* + 2, \ldots, m.
\]

By the definition of \( x_i^+ \) and the above equation, we obtain

\[
g_i(x_i^*) \leq g_i(x_i^+), \quad i = i^* + 1, i^* + 2, \ldots, m.
\]

Hence, it holds

\[
g^* = \sum_{i=1}^m g_i(x_i^*) \leq \sum_{i=1}^{i^*} g_i(x_i^*) + \sum_{i=i^*+1}^m g_i(x_i^+) = g(x').
\]

The solution \( x' \) attains the conditionally maximum total priority, and therefore it is also an optimal solution of problem MIXTURE_LEXICO. \(\square\)

We then define variables \( F_i \) and \( G_i \) for \( i = 1, 2, \ldots, m \) as follows:

- \( F_i \): the conditionally minimum of total weight \( \sum_{j=1}^i f_j(x_j) \) of the partial list \((x_1; x_2; \cdots; x_i)\) of 0-1 vectors from the first type to the \( i \)-th type such that \( \sum_{j=1}^i f_j(x_j) \geq t - \sum_{j=i+1}^m f_j^+ \) (see Eq. (35)) and \( f_j(x_j) \geq f_j^+ \) for all \( j = 1, 2, \ldots, i \).
- \( G_i \): the conditionally maximum of total priority \( \sum_{j=1}^i g_j(x_j) \) of the partial list \((x_1; x_2; \cdots; x_i)\) of 0-1 vectors from the first type to the \( i \)-th type such that \( \sum_{j=1}^i f_j(x_j) = F_i \) and \( f_j(x_j) \geq f_j^+ \) for all \( j = 1, 2, \ldots, i \).

With these variables, the minimum total weight \( f^* \) and the conditionally maximum total priority \( g^* \) of problem MIXTURE_LEXICO can be obtained as \( f^* = F_m \) and \( g^* = G_m \). By using the same dynamic programming recursives as Eqs. (31)–(34), we can obtain \( F_i \) and \( G_i \) for \( i = 1, 2, \ldots, m \).

If we know the critical index \( i^* \) of item type, \( F_m \) and \( G_m \) can be computed with a smaller computational effort as follows. For \( i = i^* + 1, i^* + 2, \ldots, m \), it holds \( F_i = F_{i-1} + f_i^+ \), where \( f_i^+ \) has been computed by the dynamic programming procedure at the first stage. By the definition of the critical index, we have

\[
\sum_{i=1}^{i^*-1} f_i^* < t - \sum_{i=i^*+1}^m f_i^+,
\]

\[
\sum_{i=1}^{i^*} f_i^* \geq t - \sum_{i=i^*+1}^m f_i^+.
\]

Thus, \( F_{i^*} \) can be computed with \( \alpha_{i^*-1}(q) \) for

\[
q = \sum_{i=1}^{i^*-1} f_i^+, \sum_{i=1}^{i^*-1} f_i^+ + 1, \ldots, t - \sum_{i=i^*+1}^m f_i^+,
\]

and \( u_{i^*,n}(r) \) for \( r = f_{i^*}^+, f_{i^*}^+ + 1, \ldots, \lambda_{i^*} \). Note that in Section 4, we used \( \alpha_{i^*-1}(q) \) for

\[
q = \sum_{i=1}^{i^*} f_i^+, \sum_{i=1}^{i^*} f_i^+ + 1, \ldots, \lambda - \sum_{i=i^*}^m f_i^+,
\]
where \( \lambda = t + w_{\text{max}} - 1 \) (see Eq. (29)), when computing \( \alpha_i(q) \) and \( \gamma_i(q) \) for \( i = 1, 2, \ldots, i^* - 1 \) and \( q = \sum_{j=1}^{i-1} f_j^+ + 1, \ldots, t - \sum_{j=i+1}^{m} f_j^+ \).

However, the critical index of item type cannot be known in advance. Due to this, for obtaining \( F_i \) and \( G_i \), we conduct the following computation with

1. \( \alpha_{i-1}(q) \) and \( \gamma_{i-1}(q) \) for \( q = \sum_{j=1}^{i-1} f_j^+ + 1, \ldots, t - \sum_{j=i+1}^{m} f_j^+ \), and
2. \( u_{in}(r) \) and \( v_{in}(r) \) for \( r = f_i^+, f_i^+ + 1, \ldots, \lambda_i \).

That is, let

\[
Q_1 = \left\{ r \mid r \geq t - \sum_{j=2}^{m} f_j^+ \left( > f_i^+ \right), \; u_{1n}(r) = 1 \right\}
\]

and if \( Q_1 \neq \emptyset \), let \( \omega_1 \) denote the minimum of value \( r \) in the non-empty set \( Q_1 \).

Then, compute

\[ F_1 = \begin{cases} \omega_1 & \text{if } Q_1 \neq \emptyset, \\ \infty & \text{otherwise}, \end{cases} \quad (37) \]

and

\[ G_1 = \begin{cases} v_{1n}(\omega_1) & \text{if } Q_1 \neq \emptyset, \\ -1 & \text{otherwise}. \end{cases} \quad (38) \]

For \( i = 2, 3, \ldots, m \), let

\[
Q_i = \left\{ q + r \mid q + r \geq t - \sum_{j=i+1}^{m} f_j^+ \left( > f_i^+ \right), \; \alpha_{i-1}(q) = 1 \right\}
\]

\[
\sum_{j=1}^{i-1} f_j^+ \leq q \leq t - \sum_{j=i}^{m} f_j^+, \; u_{in}(r) = 1 \right\}
\]

and if \( Q_i \neq \emptyset \), let \( \omega_i \) denote the minimum of value \( q + r \) in the non-empty set \( Q_i \), and let \( r_i \) denote the value of \( r \) taking the minimum \( \omega_i \) in set \( Q_i \). Then, compute

\[ F_i = \begin{cases} \min \left\{ F_{i-1} + f_i^+, \; \omega_i \right\} & \text{if } Q_i \neq \emptyset, \\ F_{i-1} + f_i^+ & \text{otherwise}, \end{cases} \quad (39) \]

and

\[
G_i = \begin{cases} G_{i-1} + v_{in}(f_i^+) & \text{if } Q_i \neq \emptyset \text{ and } F_{i-1} + f_i^+ < \omega_i, \\ \gamma_{i-1}(\omega_i - r_i) + v_{in}(r_i) & \text{if } Q_i \neq \emptyset \text{ and } F_{i-1} + f_i^+ > \omega_i, \\ \max \left\{ G_{i-1} + v_{in}(f_i^+) \right\} \quad & \text{if } Q_i \neq \emptyset \text{ and } F_{i-1} + f_i^+ = \omega_i, \\ G_{i-1} + v_{in}(f_i^+) & \text{if } Q_i = \emptyset \text{ and } F_i < \infty, \\ -1 & \text{otherwise}. \end{cases} \quad (40) \]

It takes \( O(t) \times O(\lambda_i) = O(t^2) \) time to obtain \( F_i \) and \( G_i \) for each \( i = 1, 2, \ldots, m \). As in procedure Combine_DP, by an appropriate backtracking, we can know the optimal sum weights \( f_i^+ \) for all item types \( i = 1, 2, \ldots, m \) in additional \( O(m) \) time, and the remaining computation process for eventually constructing an optimal solution is also the same as in the original two-stage constitution algorithm. Therefore,
problem MIXTURE_LEXICO for an arbitrary $m \geq 3$ can also be solved in $O(mnt + mt^2)$ time by utilizing the critical index of item type.

REFERENCES

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

[2] S. Imahori and Y. Karuno, Pseudo-polynomial time algorithms for food mixture packing by automatic combination weighers, in *Proceedings of International Symposium on Scheduling 2013 (ISS 2013)*, 2013, 59–64.

[3] S. Imahori, Y. Karuno, H. Nagamochi and X. Wang, Kansei engineering, humans and computers: Efficient dynamic programming algorithms for combinatorial food packing problems, *International Journal of Biometrics*, 3 (2011), 228–245.

[4] S. Imahori, Y. Karuno, R. Nishizaki and Y. Yoshimoto, Duplex and quasi-duplex operations in automated food packing systems, in *IEEE Xplore of the Fifth IEEE/SICE International Symposium on System Integration (SII 2012)*, 2012, 810–815.

[5] S. Imahori, Y. Karuno and K. Tateishi, Dynamic programming algorithms for producing food mixture packages by automatic combination weighers, *Journal of Advanced Mechanical Design, Systems, and Manufacturing*, 8 (2014), 1–11.

[6] Ishida Co., Ltd., *Products (Total System Solutions), Weighing and Packaging*, 2015. Available from: http://www.ishida.com/products/.

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

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

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

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

[11] Y. Karuno, K. Takashashi and A. Yamada, Dynamic programming algorithms with data rounding for combinatorial food packing problems, *Journal of Advanced Mechanical Design, Systems, and Manufacturing*, 7 (2013), 233–243.

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

[13] H. A. Wurdemann, V. Aminzadeh, J. S. Dai, J. Reed and G. Purnell, Category-based food ordering processes, *Trends in Food Science & Technology*, 22 (2011), 14–20.

[14] Yamato Scale Co., Ltd., *Category Search, Filling and Packaging*, 2015. Available from: http://www.yamato-scale.co.jp/en/products/index.

Received November 2014; revised April 2015.

E-mail address: imahori@ise.chuo-u.ac.jp
E-mail address: karuno@kit.ac.jp
E-mail address: m4623025@edu.kit.ac.jp