Dynamic Programming Algorithms for Duplex Food Packing Problems

Shinji Imahori*, Yoshiyuki Karuno† and Yui Yoshimoto†
* Department of Computational Science and Engineering
Nagoya University
Furo-cho, Chikusa, Nagoya 464-8603, Japan
† Department of Mechanical and System Engineering
Kyoto Institute of Technology
Matsugasaki, Sakyo, Kyoto 606-8585, Japan
karuno@kit.ac.jp

***Abstract*—In this paper, we discuss a lexicographic bi-criteria combinatorial optimization problem arising in automated food packing systems known as so-called automatic combination weighers. A typical food packing system possesses n weighing hoppers. Some amount of foods is thrown into each hopper, and it is called an item. We deal with a duplex packing operation such that the food packing system chooses two disjoint subsets I' and I'' from the set I of the current n items to produce two packages of foods. After choosing two subsets I' and I'', the resulting empty hoppers are supplied with next new items, and the set I is updated. By repeating the duplex packing operation, a large number of packages are produced two by two. The primary objective of lexicographic bi-criteria duplex food packing problem is to minimize the total weight of chosen items for two packages, making the total weight of each package no less than a specified target weight T. The second objective is to maximize the total priority of chosen items for two packages so that items with longer durations in hoppers are preferably chosen. The priority of an item is given as its duration in hopper. In this paper, we prove that the lexicographic bi-criteria duplex food packing problem can be solved in $O(nT^2)$ time by dynamic programming if all input data are integral.**

I. Introduction

We discuss a combinatorial optimization problem arising in automated food packing systems known as so-called *automatic combination weighers* (see Morinaka [8]). As depicted in Fig. 1, a typical food packing system possesses n weighing hoppers. Some amount of foods (such as a green pepper, a ham, a handful of potato chips, and so on) is thrown into each hopper, and it in the i-th hopper is called *item* i ($i = 1, 2, \ldots, n$). Given a set $I = \{i \mid i = 1, 2, \ldots, n\}$ of the current n items in hoppers, a *duplex food packing system* chooses two disjoint subsets I' and I'' from the set I, and puts the chosen items for the two subsets into individual packages. The resulting empty hoppers are supplied with next new items, and the set I is updated by taking the union of the remaining items and the newly supplied items. The duplex food packing system repeats the *duplex packing operation* to produce a large number of packages two by two. Note that it always chooses some current items in hoppers without knowing the weights of next new items.

In this paper, we first formulate the problem of choosing two disjoint subsets I' and I'' from the set I at each duplex packing operation as a lexicographic bi-criteria combinatorial optimization problem of off-line setting. From a viewpoint of the service conscience, the total weight of chosen items for each package must be no less than a specified target weight T. This is referred to as the *target weight constraint*, which is a hard constraint of automatic combination weighers (see Morinaka [8] again). The primary objective of lexicographic bi-criteria duplex food packing problem is to minimize the total weight of chosen items for two packages under the target weight constraint. In other words, this objective aims at minimizing the amount of *surplus* in two packages. In addition, items with longer durations in hoppers would like to be preferably chosen (see Kameoka and Nakatani [3]). For example, when the system handles some kind of fresh (or raw) food that would like to be vacuum-packed as soon as possible, it is undesirable that items with long durations in hoppers occur. For the purpose, we introduce a priority to each current item, following the previous papers (see Imahori et al. [2], Karuno, Nagamochi and Ohshima [5], Karuno, Nagamochi and Wang [6], [7]), and try to maximize the total priority of chosen items for two packages as the second objective.

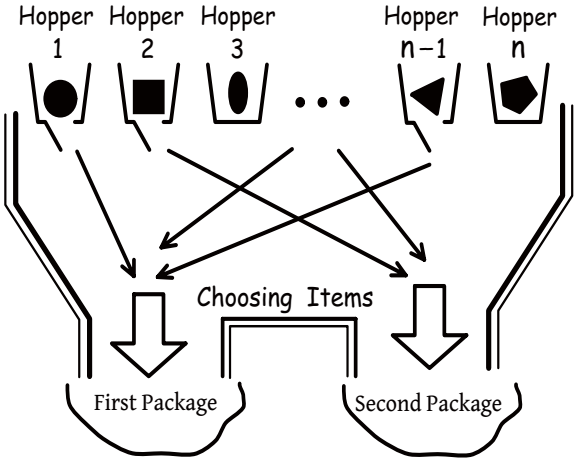

Fig. 1. A duplex food packing system.

Then, we propose $O(nT^2)$ time algorithms based on dynamic programming to the lexicographic bi-criteria duplex food packing problem, where all input data are assumed to be integral. The pseudo-polynomial time algorithms imply that the problem is not strongly NP-hard, but is weakly NP-hard [1].

A *singular food packing system* performs a packing operation such that it chooses a single subset I' from the set I of the current items. It repeats the *singular packing operation* to produce a large number of packages one by one. The singular food packing systems have been modeled, and $O(nT)$ time algorithms have recently been proposed to their lexicographic bi-criteria food packing problems (see Imahori et al. [2], Karuno, Nagamochi and Wang [7] again). There exist some automatic combination weighers which are able to produce about two hundred packages per minute at the maximum [13], i.e., approximately three hundred milliseconds per singular packing operation. They spend most time in measuring the weights of next new items accurately, and only a few milliseconds may be left for choosing a subset I' of items at each singular packing operation. Imahori et al. [2] have also showed numerical results that the $O(nT)$ time algorithm solved their test instances with up to $n = 40$ of the singular food packing problem within four milliseconds on a personal computer with Intel Pentium M CPU (1.20GHz) and 1GB memory. Several simulation studies on automatic combination weighers have also been reported (see Kameoka, Nakatani and Inui [4], Murakami et al. [9], [10], [11], [12]).

Note that a singular food packing system generally measures the weights of next new items at every singular packing operation, and hence it performs the measuring step twice in order to produce two packages. On the other hand, a duplex food packing system performs the measuring step once while it produces two packages. Thus, concerning the productivity, duplex food packing systems may be more advantageous than singular food packing systems, if a certain kind of food is handled such that it takes much time to measure the weights of next new items accurately.

II. Problem Description

We denote by n the number of weighing hoppers as mentioned in the previous section. Let N be the total number of packages to be produced, and let K_d be the total iteration number of duplex packing operations to produce N packages. Then, if the duplex food packing system always produces two packages at every packing operation (for example, a sufficient number of candidate items are given), it holds $K_d = \lceil N/2 \rceil$. On the other hand, let K_s be the total iteration number of singular packing operations. Then, if the singular food packing system always produces one package at every packing operation, it holds $K_s = N$.

We call the duplex food packing problem with respect to the lexicographic bi-criteria (i.e., the total weight of chosen items for two packages as the primary objective, and total priority as the second) LEXICO_DUPLEX. When we disregard the total priority (i.e., only with the primary objective of the total weight), we call the duplex food packing problem PRIMAL_DUPLEX.

On the other hand, we call the singular food packing problem with respect to the lexicographic bi-criteria LEXICO_SINGULAR, and the singular food packing problem only with the primary objective PRIMAL_SINGULAR.

The current item in the i-th hopper is referred to as item i (see Fig. 1 again). Let ℓ be the current iteration number of duplex packing operation ($1 \leq \ell \leq K_d$), and let ℓ_i be the iteration number at which the current item i has been thrown into the i-th hopper when it was empty. Then, we refer to $c_i = \ell - \ell_i + 1$ as the *duration* in hopper of item i. We are going to set the priority of item i by the duration.

An instance of problem LEXICO_DUPLEX at each duplex packing operation is composed of the following inputs.

- $I = \{i \mid i = 1, 2, \ldots, n\}$: set of the current n items.
- w_i: positive integer weight of item $i \in I$.
- γ_i ($:= c_i$): priority of item $i \in I$, which is assumed to be a positive integer.
- T: target weight for each package, which is also assumed to be a positive integer.

Besides them, the maximum of weights and total weight over all n items in I are denoted by

$$w_{max} = \max_{i \in I} \Big\{ w_i \Big\} \quad \text{and} \quad W = \sum_{i=1}^{n} w_i,$$

respectively.

Problem LEXICO_DUPLEX is formulated by using 0-1 vectors $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$, instead of using subsets I' and I'', where

$$x_i = \begin{cases} 1 & \text{if item } i \text{ is chosen for the first package,} \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

$$y_i = \begin{cases} 1 & \text{if item } i \text{ is chosen for the second package,} \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

LEXICO_DUPLEX

$$\text{minimize} \quad f(x, y) = \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} w_i y_i \tag{3}$$
as the primary objective,

$$\text{maximize} \quad g(x, y) = \sum_{i=1}^{n} \gamma_i x_i + \sum_{i=1}^{n} \gamma_i y_i \tag{4}$$
as the second objective,

$$\text{subject to} \quad \sum_{i=1}^{n} w_i x_i \geq T, \tag{5}$$

$$\sum_{i=1}^{n} w_i y_i \geq T, \tag{6}$$

$$x_i + y_i \leq 1, \ i = 1, 2, \ldots, n, \tag{7}$$

$$x_i, y_i \in \{0, 1\}, \ i = 1, 2, \ldots, n. \tag{8}$$

The target weight constraints for two packages are represented by Eqs. (5) and (6). The disjoint constraint for two packages is expressed by Eq. (7). The binary constraint of variables x_i and y_i is represented by Eq. (8). A solution (x, y) satisfying Eqs. (5)–(8) is referred to as a *feasible solution*. The primary objective of Eq. (3) aims at minimizing the amount of surplus in a feasible solution, together with Eqs. (5) and (6) (i.e., the target weight constraints). The second objective of Eq. (4) is introduced to expect that items with longer durations in hoppers are preferably chosen (recall that we do not solve the lexicographic bi-criteria duplex food packing problem of on-line setting, but of off-line setting at each duplex packing operation).

Suppose that by solving PRIMAL_SINGULAR (resp. LEXICO_SINGULAR), a subset I' has already been chosen from a given set I of current n items so that the surplus (i.e., $\sum_{i \in I'} w_i - T$) is minimized, and that someone asks to choose one more subset I'' from the set $I - I'$ of the remaining items. It is obvious that solving PRIMAL_DUPLEX (resp. LEXICO_DUPLEX) is different from such a repetition of solving PRIMAL_SINGULAR (resp. LEXICO_SINGULAR) twice (we refer to the repetition as *quasi-duplex packing operation*).

There may be many different ways to define the priority of an item, for example $\gamma_i := \max\{c_i - \alpha + 1, 1\}$, where the α is a positive integer threshold. However, we assume that the complexity of LEXICO_DUPLEX is independent of the definition of the priority.

For omitting some trivial cases, we also assume

$$w_i < T \quad \text{for any item } i \in I, \tag{9}$$

and

$$3T \leq W \ (\leq n \cdot w_{max}). \tag{10}$$

Under the above assumption, the duplex food packing system can always produce two packages at each duplex packing operation such that the total weight of each package is no less than the target weight T (we see that the inequality $2T \leq W$ is not a sufficient condition, but is only a necessary condition so that a given instance of problem LEXICO_DUPLEX has at least one feasible solution). The correctness can be derived from Lemma 1, which is going to be provided in the next section.

For a given instance of problem LEXICO_DUPLEX, we denote by Z^* the minimum of the total weight of chosen items for two packages in a feasible solution, and by $(x, y) = (\hat{x}, \hat{y})$ such a feasible solution that attains the minimum Z^* $(= f(\hat{x}, \hat{y}))$ of the total weight. An *optimal solution* $(x, y) = (x^*, y^*)$ is defined as a feasible solution such that it satisfies $f(x^*, y^*) = Z^*$ and maximizes the total priority among feasible solutions with the minimum total weight Z^*, i.e., it satisfies $g(x^*, y^*) \geq g(\hat{x}, \hat{y})$ for any feasible solution $(\hat{x}, \hat{y})$. We call the value $g(x^*, y^*)$ of the second objective *conditionally maximum total priority*, since it is the maximum of the total priority of a feasible solution under the condition that the primary objective is optimized. We denote by $G^* = g(x^*, y^*)$ the conditionally maximum total priority. Problem LEXICO_DUPLEX asks to find an optimal solution (x^*, y^*). If we are asked to find a feasible solution $(x, y) = (\hat{x}, \hat{y})$ with the minimum total weight $f(\hat{x}, \hat{y}) = Z^*$, the problem is PRIMAL_DUPLEX.

III. Dynamic Programming

In this section, we propose an $O(nT^2)$ time algorithm based on dynamic programming to problem LEXICO_DUPLEX. In § *A*, we first provide an upper bound on the total weight of each package yielded by an optimal solution of LEXICO_DUPLEX. In § *B*, we then define state variables and propose a dynamic programming algorithm. In § *C*, we finally discuss the correctness of the dynamic programming recursives.

A. Upper Bounds on the Total Weight of Each Package

Let $(x^*, y^*) = ((x_1^*, x_2^*, \ldots, x_n^*), (y_1^*, y_2^*, \ldots, y_n^*))$ be an optimal solution of a given instance of LEXICO_DUPLEX, and let

$$Z_1^* = \sum_{i=1}^{n} w_i x_i^* \quad \text{and} \quad Z_2^* = \sum_{i=1}^{n} w_i y_i^*$$

be the total weights of two individual packages yielded by the optimal solution. It holds $Z^* = Z_1^* + Z_2^*$ by definition. As a trivial upper bound, it holds $Z_s^* \leq W$ for each $s \in \{1, 2\}$. We obtain a better upper bound on the Z_s^* $(s \in \{1, 2\})$ by the following lemma.

Lemma 1 For an instance of problem LEXICO_DUPLEX, the total weights Z_1^* and Z_2^* of two packages yielded by an optimal solution (x^*, y^*) satisfy the following inequalities.

$$T \leq Z_s^* < T + w_{max} \quad \text{for each } s \in \{1, 2\}. \tag{11}$$

Proof. An optimal solution (x^*, y^*) satisfies the target weight constraints (see Eqs. (5) and (6)), and hence it is clear that the former of Eq. (11) holds.

On the other hand, by the optimality of (x^*, y^*), it holds $Z_1^* - w_k < T$ for any item k with $x_k^* = 1$, and it similarly holds $Z_2^* - w_{k'} < T$ for any item k' with $y_{k'}^* = 1$, which implies the latter of Eq. (11). ■

For notational convenience, we define

$$T' = T + w_{max} - 1. \tag{12}$$

Note that $T' < 2T = O(T)$ holds from Eqs. (9) and (12), while $W = \Omega(n)$ holds.

B. State Variables and Recursives

First, we define 0-1 state variables $u_k(p, q)$ $(k = 1, 2, \ldots, n,$ $p = 0, 1, \ldots, T',$ $q = 0, 1, \ldots, T'$ as follows.

$$u_k(p, q) = 1 \iff \begin{cases} \text{There exists a pair of (partial) 0-1 vectors} \\ (x_1, \ldots, x_k) \text{ and } (y_1, \ldots, y_k) \text{ such that} \\ x_i + y_i \leq 1 \text{ for any } i = 1, \ldots, k, \\ \sum_{i=1}^{k} w_i x_i = p, \text{ and } \sum_{i=1}^{k} w_i y_i = q. \end{cases}$$

$u_k(p,q) = 0$

$\iff$ Such a pair of 0-1 vectors does not exist.

The common interval $\{0,1,\ldots,T'\}$ of parameters p and q is determined by Lemma 1.

We also define variables $v_k(p,q)$ ($k = 1,2,\ldots,n$, $p = 0,1,\ldots,T'$, $q = 0,1,\ldots,T'$) to record the maximum of the total priority $\sum_{i=1}^{k}\gamma_i x_i + \sum_{i=1}^{k}\gamma_i y_i$ of a pair of (partial) 0-1 vectors $(x_1,x_2,\ldots,x_k)$ and $(y_1,y_2,\ldots,y_k)$ such that it satisfies $x_i + y_i \leq 1$ for any $i = 1,\ldots,k$, $\sum_{i=1}^{k} w_i x_i = p$, and $\sum_{i=1}^{k} w_i y_i = q$. Moreover, we introduce three kinds of additional variables $b_k^{(1)}(p,q)$, $b_k^{(2)}(p,q)$ and $b_k^{(3)}(p,q)$ ($k = 1,2,\ldots,n$, $p = 0,1,\ldots,T'$, $q = 0,1,\ldots,T'$) in order to maintain the (conditionally) maximum total priority in the $v_k(p,q)$, which correspond to the three possible cases for a couple of x_k and y_k, i.e., $\langle x_k, y_k\rangle \in \{\langle 0,0\rangle, \langle 0,1\rangle, \langle 1,0\rangle\}$ (see Eqs. (7) and (8)).

Then, we propose the following recursives of dynamic programming. For $p = 0,1,\ldots,T'$ and $q = 0,1,\ldots,T'$,

$$u_1(p,q) = \begin{cases} 1 & \text{if } (p=0 \text{ and } q=0), \text{ or} \\ & \quad \text{if } (p=0 \text{ and } q=w_1), \text{ or} \\ & \quad\quad \text{if } (p=w_1 \text{ and } q=0). \\ 0 & \text{otherwise,} \end{cases} \quad (13)$$

$$v_1(p,q) = \begin{cases} 0 & \text{if } (p=0 \text{ and } q=0), \\ \gamma_1 & \text{if } (p=0 \text{ and } q=w_1), \text{ or} \\ & \quad \text{if } (p=w_1 \text{ and } q=0). \\ -1 & \text{otherwise,} \end{cases} \quad (14)$$

and for $k = 2,3,\ldots,n$, $p = 0,1,\ldots,T'$ and $q = 0,1,\ldots,T'$,

$$u_k(p,q) = \begin{cases} 1 & \text{if } u_{k-1}(p,q) = 1, \text{ or} \\ & \quad \text{if } (q - w_k \geq 0 \text{ and} \\ & \quad\quad u_{k-1}(p, q-w_k) = 1), \text{ or} \\ & \quad \text{if } (p - w_k \geq 0 \text{ and} \\ & \quad\quad u_{k-1}(p-w_k, q) = 1), \\ 0 & \text{otherwise,} \end{cases} \quad (15)$$

$$b_k^{(1)}(p,q) = \begin{cases} v_{k-1}(p,q) & \text{if } u_{k-1}(p,q) = 1, \\ -1 & \text{otherwise,} \end{cases} \quad (16)$$

$$b_k^{(2)}(p,q) = \begin{cases} v_{k-1}(p, q-w_k) + \gamma_k & \\ \quad \text{if } (q - w_k \geq 0 \text{ and} & \\ \quad\quad u_{k-1}(p, q-w_k) = 1), & \\ -1 \quad \text{otherwise,} & \end{cases} \quad (17)$$

$$b_k^{(3)}(p,q) = \begin{cases} v_{k-1}(p-w_k, q) + \gamma_k & \\ \quad \text{if } (p - w_k \geq 0 \text{ and} & \\ \quad\quad u_{k-1}(p-w_k, q) = 1), & \\ -1 \quad \text{otherwise,} & \end{cases} \quad (18)$$

$$v_k(p,q) = \max\left\{b_k^{(1)}(p,q), b_k^{(2)}(p,q), b_k^{(3)}(p,q)\right\}. \quad (19)$$

The computation of all the $u_k(p,q)$, $b_k^{(1)}(p,q)$, $b_k^{(2)}(p,q)$, $b_k^{(3)}(p,q)$ and $v_k(p,q)$ requires $O(n \times T' \times T') = O(nT^2)$ time (see Eqs. (9) and (12)). After the computation, we find a minimum of $p+q$ such that it satisfies $u_n(p,q) = 1$, $p \geq T$ and $q \geq T$. Let $\hat{p} + \hat{q}$ be the minimum. Then, it satisfies $u_n(\hat{p},\hat{q}) = 1$, $\hat{p} \geq T$ and $\hat{q} \geq T$. We easily see that it also satisfies $\hat{p} + \hat{q} = Z^*$ (i.e., the minimum of the total weight of chosen items for two packages), where we regard $\hat{p} = Z_1^*$ and $\hat{q} = Z_2^*$ (see Lemma 1).

Let $S = \{(\hat{p},\hat{q}) \mid \hat{p} + \hat{q} = Z^*,\ u_n(\hat{p},\hat{q}) = 1,\ \hat{p} \in \{T, T+1,\ldots,T'\},\ \hat{q} \in \{T, T+1,\ldots,T'\}\}$ be the set of pairs of $\hat{p}$ and $\hat{q}$, and let $(p^*,q^*) \in S$ be a pair such that it satisfies $v_n(p^*,q^*) \geq v_n(\hat{p},\hat{q})$ for any pair $(\hat{p},\hat{q}) \in S$. Note that the set S satisfies $|S| = O(T'-T) \times O(T'-T) = O(T^2)$. We can find the minimum $p^* + q^*$ in $O(T^2)$ time by checking $O(T^2)$ variables of $u_n(p,q)$ and $v_n(p,q)$ with $T \leq p \leq T'$ and $T \leq q \leq T'$, which have already been computed.

In the next subsection, we are going to see that it holds $v_n(p^*,q^*) = G^*$ (i.e., the conditionally maximum total priority) from Eqs. (14), (16)–(19).

Thus, the proposed dynamic programming algorithm can compute the Z^* and G^* in $O(nT^2)$ time. By backtracking the computation process, we can construct an optimal solution (x^*,y^*) in $O(n)$ time additionally. We call the dynamic programming algorithm Lexico_Duplex_DP.

Theorem 1 For an instance of problem LEXICO_DUPLEX, Lexico_Duplex_DP can obtain an optimal solution (x^*,y^*) in $O(nT^2)$ time.

C. Correctness of Recursives

In order to complete the proof of Theorem 1, it is left to show that it holds $v_n(p^*,q^*) = G^*$.

Recall that there are three possible cases for a couple of x_k and y_k, i.e., $\langle x_k, y_k\rangle \in \{\langle 0,0\rangle, \langle 0,1\rangle, \langle 1,0\rangle\}$, $k = 1,2,\ldots,n$, due to the disjoint constraint for two packages and the binary constraint (see Eqs. (7) and (8)). For $k = 1$, we consider the following three cases: (i) the first item (i.e., item 1) is neither chosen for the first package nor for the second package, (ii) it is chosen for the second package, and (iii) it is chosen for the first package. In the case of (i), the priority γ_1 does not contribute to the $v_1(0,0)$ (i.e., $v_1(0,0) = 0$). In the cases of (ii) and (iii), the priority γ_1 contributes to the $v_1(0,w_1)$ and $v_1(w_1,0)$ (i.e., $v_1(0,w_1) = \gamma_1$ and $v_1(w_1,0) = \gamma_1$), respectively. Hence, the initialization of Eq. (14) is correct.

Similarly, for a general k ($2 \leq k \leq n$), we consider the following three cases: (i) item k is neither chosen for the first package nor for the second package, (ii) it is chosen for the second package, and (iii) it is chosen for the first package.

In the case of (i), if $u_{k-1}(p,q) = 1$ holds for a pair of p and q, then $u_k(p,q) = 1$ also holds. That is, $\langle x_k, y_k\rangle = \langle 0,0\rangle$ is a candidate setting in an optimal solution. The priority γ_k does not contribute to the total priority $v_k(p,q)$, and hence the $b_k^{(1)}(p,q)$ records the total priority $v_{k-1}(p,q)$ by Eq. (16). If the candidate setting $\langle x_k, y_k\rangle = \langle 0,0\rangle$ is actually taken up in the output solution of the dynamic programming algorithm, then item k is not chosen for any package.

In the case of (ii), if $q - w_k \geq 0$ and $u_{k-1}(p, q-w_k) = 1$ hold for a pair of p and q, $u_k(p,q) = 1$ also holds. That is, $\langle x_k, y_k\rangle = \langle 0,1\rangle$ is a candidate setting in an optimal solution. The priority γ_k may contribute to the total priority of the second package, and hence the $b_k^{(2)}(p,q)$ records the total

priority $v_{k-1}(p, q - w_k) + \gamma_k$ by Eq. (17). If the candidate setting $\langle x_k, y_k \rangle = \langle 0, 1 \rangle$ is actually taken up in the output solution, then item k is chosen for the second package.

In the case of (iii), if $p - w_k \geq 0$ and $u_{k-1}(p - w_k, q) = 1$ hold for a pair of p and q, $u_k(p, q) = 1$ also holds. That is, $\langle x_k, y_k \rangle = \langle 1, 0 \rangle$ is a candidate setting in an optimal solution. The priority γ_k may contribute to the total priority of the first package, and hence the $b_k^{(3)}(p, q)$ records the total priority $v_{k-1}(p - w_k, q) + \gamma_k$ by Eq. (18). If the candidate setting $\langle x_k, y_k \rangle = \langle 1, 0 \rangle$ is actually taken up in the output solution, then item k is chosen for the first package.

By Eq. (19), the $v_k(p, q)$ adopts the maximum of the total priority of a candidate setting (which breaks ties arbitrarily). Thus, the value $v_n(\hat{p}, \hat{q})$ maintains the maximum of the total priority of a pair of 0-1 vectors $(\hat{x}, \hat{y}) = ((\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n), (\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_n))$ such that it satisfies $\hat{x}_i + \hat{y}_i \leq 1$ for any $i = 1, \ldots, n$, $\sum_{i=1}^{n} w_i \hat{x}_i = \hat{p} \geq T$, and $\sum_{i=1}^{n} w_i \hat{y}_i = \hat{q} \geq T$. By definition, the pair $(p^*, q^*) \in S$ meets $v_n(p^*, q^*) \geq v_n(\hat{p}, \hat{q})$ for any pair $(\hat{p}, \hat{q}) \in S$.

Therefore, we have $v_n(p^*, q^*) = G^*$, which completes the proof of Theorem 1.

IV. Improvement For Implementation

We notice that for any $k \in \{1, 2, \ldots, n\}$, and for any pair of $p \in \{0, 1, \ldots, T'\}$ and $q \in \{0, 1, \ldots, T'\}$, the state variables and other variables for the conditionally maximum total priority computed in dynamic programming algorithm Lexico_Duplex_DP satisfy the following *symmetric property*.

$$u_k(p, q) = u_k(q, p) \quad \text{and} \quad v_k(p, q) = v_k(q, p). \tag{20}$$

Applying the symmetric property, we can reduce the amount of space used for maintaining the variables of $u_k(p, q)$, $b_k^{(1)}(p, q)$, $b_k^{(2)}(p, q)$, $b_k^{(3)}(p, q)$ and $v_k(p, q)$ in the dynamic programming algorithm by about half (although the time complexity remains $O(nT^2)$ time).

The modified dynamic programming recursives are presented as follows. For $p = 0, 1, \ldots, T'$ and $q = p, p + 1, \ldots, T'$,

$$u_1(p, q) = \begin{cases} 1 & \text{if } (p = 0 \text{ and } q = 0), \text{ or} \\ & \quad \text{if } (p = 0 \text{ and } q = w_1), \\ 0 & \text{otherwise,} \end{cases} \tag{21}$$

$$v_1(p, q) = \begin{cases} 0 & \text{if } (p = 0 \text{ and } q = 0), \\ \gamma_1 & \text{if } (p = 0 \text{ and } q = w_1), \\ -1 & \text{otherwise,} \end{cases} \tag{22}$$

and for $k = 2, 3, \ldots, n$, $p = 0, 1, \ldots, T'$ and $q = p, p + 1, \ldots, T'$,

$$u_k(p, q) = \begin{cases} 1 & \text{if } u_{k-1}(p, q) = 1, \text{ or} \\ & \quad \text{if } (q - w_k \geq p \text{ and} \\ & \qquad u_{k-1}(p, q - w_k) = 1), \text{ or} \\ & \quad \text{if } (0 \leq q - w_k < p \text{ and} \\ & \qquad u_{k-1}(q - w_k, p) = 1), \text{ or} \\ & \quad \text{if } (p - w_k \geq 0 \text{ and} \\ & \qquad u_{k-1}(p - w_k, q) = 1), \\ 0 & \text{otherwise,} \end{cases} \tag{23}$$

$$\begin{aligned} b_k^{(1)}(p, q) &= \begin{cases} v_{k-1}(p, q) & \text{if } u_{k-1}(p, q) = 1, \\ -1 & \text{otherwise,} \end{cases} \\ b_k^{(2)}(p, q) &= \begin{cases} v_{k-1}(p, q - w_k) + \gamma_k \\ \qquad \text{if } (q - w_k \geq p \text{ and} \\ \qquad\quad u_{k-1}(p, q - w_k) = 1), \\ v_{k-1}(q - w_k, p) + \gamma_k \\ \qquad \text{if } (0 \leq q - w_k < p \text{ and} \\ \qquad\quad u_{k-1}(q - w_k, p) = 1), \\ -1 \quad \text{otherwise,} \end{cases} \\ b_k^{(3)}(p, q) &= \begin{cases} v_{k-1}(p - w_k, q) + \gamma_k \\ \qquad \text{if } (p - w_k \geq 0 \text{ and} \\ \qquad\quad u_{k-1}(p - w_k, q) = 1), \\ -1 \quad \text{otherwise,} \end{cases} \\ v_k(p, q) &= \max\Big\{ b_k^{(1)}(p, q), b_k^{(2)}(p, q), b_k^{(3)}(p, q) \Big\}. \end{aligned} \tag{24}$$

The remaining procedure of the modified dynamic programming is similar to Lexico_Duplex_DP proposed in the previous section. That is, let $S = \{(\hat{p}, \hat{q}) \mid \hat{p} + \hat{q} = Z^*,\ u_n(\hat{p}, \hat{q}) = 1,\ T \leq \hat{p} \leq T',\ \hat{p} \leq \hat{q} \leq T'\}$ be the set of pairs of $\hat{p}$ and $\hat{q}$, and let $(p^*, q^*) \in S$ be a pair such that it satisfies $v_n(p^*, q^*) \geq v_n(\hat{p}, \hat{q})$ for any pair $(\hat{p}, \hat{q}) \in S$. Then, we can find the minimum $p^* + q^*$ in $O(T^2)$ time by checking $O(T^2)$ variables of $u_n(p, q)$ and $v_n(p, q)$ with $T \leq p \leq T'$ and $p \leq q \leq T'$, which have already been computed.

We refer to the modified dynamic programming algorithm as Lexico_Duplex_MDP. Even if we follow the modified recursives, the computation of all the $u_k(p, q)$, $b_k^{(1)}(p, q)$, $b_k^{(2)}(p, q)$, $b_k^{(3)}(p, q)$ and $v_k(p, q)$ requires $O(nT^2)$ time. Thus, the theoretical time complexity of Lexico_Duplex_MDP is the same as that of Lexico_Duplex_DP.

Theorem 2 For an instance of problem LEXICO_DUPLEX, Lexico_Duplex_MDP can obtain an optimal solution (x^*, y^*) in $O(nT^2)$ time.

However, by applying Eq. (20), Lexico_Duplex_MDP maintains the variables of $u_k(p, q)$, $b_k^{(1)}(p, q)$, $b_k^{(2)}(p, q)$, $b_k^{(3)}(p, q)$ and $v_k(p, q)$ only with respect to $p \leq q$. In other words, Lexico_Duplex_MDP is concerned with duplex packing operations such that the total weight of the second package is no less than that of the first package. This implies that the amount of space used for maintaining the variables in Lexico_Duplex_DP can be reduced by about half in the actual implementation.

V. Concluding Remarks

In this paper, we discussed a duplex food packing problem LEXICO_DUPLEX, which is a lexicographic bi-criteria combinatorial optimization problem. The primary objective is to minimize the total weight of chosen items for two packages, making the total weight of each package no less than a specified target weight. The second objective is to maximize the total priority of chosen items for two packages. We proposed two pseudo-polynomial time dynamic programming

algorithms, basic version Lexico_Duplex_DP and a modified version Lexico_Duplex_MDP, to the lexicographic bi-criteria duplex food packing problem, in which all input data are assumed to be integral.

The following directions for future research are open. It would be significant to compare the execution time of the modified version Lexico_Duplex_MDP with that of the basic version Lexico_Duplex_DP by conducting numerical experiments. It would be interesting to compare the execution time of Lexico_Duplex_MDP with that of a dynamic programming algorithm for problem LEXICO_SINGULAR to be required for producing the same number of packages. It would also be interesting to compare the amount of surplus yielded by solving the duplex food packing problem (i.e., solving LEXICO_DUPLEX) with that by solving the quasi-duplex food packing problem (i.e., solving LEXICO_SINGULAR for a given set I of items to produce a single package, and then solving LEXICO_SINGULAR once more for the remaining items to produce another package).

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).

REFERENCES

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

[2] Imahori, S., Karuno, Y., Nagamochi, H., and Wang, X. (2008). Efficient algorithms for combinatorial food packing problems. *Proceedings of 11th International Conference on Humans and Computers (HC 2008)*, November 2008, Nagaoka, Japan, pp. 317-322.

[3] Kameoka, K. and Nakatani, M. (2001). Feed control criterion for a combination weigher and its effects (in Japanese). *Transactions of the Society of Instrument and Control Engineers*, Vol. 37, No. 10, pp. 911–915.

[4] Kameoka, K., Nakatani, M., and Inui, N. (2000). Phenomena in probability and statistics found in a combinatorial weigher (in Japanese). *Transactions of the Society of Instrument and Control Engineers*, Vol. 36, No. 5, pp. 388–394.

[5] Karuno, Y., Nagamochi, H., and Ohshima, Y. (2006). A dynamic programming approach for a food packing problem (in Japanese). *Transactions of the Japan Society of Mechanical Engineers, Series C*, Vol. 72, No. 716, pp. 1390–1397.

[6] Karuno, Y., Nagamochi, H., and Wang, X. (2007). Bi-criteria food packing by dynamic programming. *Journal of the Operations Research Society of Japan*, Vol. 50, No. 4, pp. 376–389.

[7] Karuno, Y., Nagamochi, H., and Wang, X. (2009). Combinatorial optimization problems and algorithms in double-layered food packing equipments. *Proceedings of 4th International Symposium on Scheduling (ISS 2009)*, July 2009, Nagoya, Japan, pp. 12–17.

[8] Morinaka, H. (2000). Automatic combination weigher for product foods (in Japanese). *Journal of the Japan Society of Mechanical Engineers*, Vol. 103, No. 976, pp. 130–131.

[9] Murakami, Y., Kurata, J., Uchiyama, H., and Kawai, M. (2003). Raising the rate of feasible solutions in a bag-packing problem (by dividing input materials into groups of lighter items and heavier ones) (in Japanese). *Transactions of the Japan Society of Mechanical Engineers, Series C*, Vol. 69, No. 686, pp. 2830–2837.

[10] Murakami, Y., Kurata, J., Uchiyama, H. Taniya, K., and Kawai, M. (2003). Efficient algorithm for solving a bag-packing problem by excluding search space (in Japanese). *Transactions of the Japan Society of Mechanical Engineers, Series C*, Vol. 69, No. 688, pp. 3431–3438.

[11] Murakami, Y., Kurata, J., Uchiyama, H., and Ueno, T. (2002). Characterization of infeasible solutions in a bag-packing problem for achieving desired weight (in Japanese). *Transactions of the Society of Instrument and Control Engineers*, Vol. 38, No. 9, pp. 784–791.

[12] Murakami, Y., Uchiyama, H., Kurata, J., and Kotera, T. (2008). High efficiency of weighing machine for achieving desired weights by tuning input amount (in Japanese). *Transactions of the Japan Society of Mechanical Engineers, Series C*, Vol. 74, No. 743, pp. 1920–1925.

[13] ISHIDA CO., LTD. (2009). Weighing and Packaging (in Japanese). `http://www.ishida.co.jp`, [accessed Nov. 2009].