Journal of the Operations Research
Society of Japan
2007, Vol. 50, No. 4, 376-389

BI-CRITERIA FOOD PACKING BY DYNAMIC PROGRAMMING

Yoshiyuki Karuno
Kyoto Institute of Technology

Hiroshi Nagamochi
Kyoto University

Xiaoming Wang
Kyoto Institute of Technology

(Received October 31, 2006; Revised February 21, 2007)

Abstract In this paper, we deal with a certain type of automated food packing system with n hoppers. The system repeats throwing some amount of foods into empty hoppers to make all hoppers full of foods, and collecting the foods from several hoppers to pack them into a single package. We treat foods thrown into each hopper as an item i with an integer weight w_i. Given a set I of n items in the hoppers, the packing system chooses a subset I' ($\subseteq I$) of items, and puts them into a package so that the total weight of items is at least a specified target weight B. The packing system then throws new items into the empty hoppers, and the set I is updated to be the union of the remaining items in $I - I'$ and the new items. Repeating such packing operations, it produces a large number of packages one by one. In the packing system, an item may stay for a long time in some hopper until it is chosen for packing. To avoid such a situation, we introduce a priority γ_i for each item i, and formulate the problem of choosing a subset I' as a bi-criteria discrete optimization problem in which one objective is to minimize $\sum_{i\in I'} w_i$ such that $\sum_{i\in I'} w_i \geq B$ must be satisfied, and the other is to maximize $\sum_{i\in I'} \gamma_i$. In this paper, we propose an $O(n^2 w_{max})$ time algorithm based on dynamic programming to obtain a nondominated solution, where w_{max} is an upper bound on the weight of an item. We also report the results on computational experiments conducted to examine the performance of the proposed approach.

Keywords: Discrete optimization, automated food packing, 0-1 knapsack problem, dynamic programming

1. Introduction

In this paper, we consider a discrete optimization problem arising in a certain type of automated food packing system, so-called *automatic combination weigher* [5]. As depicted in Figure 1, the food packing system consists of n weighing hoppers. An item (such as a green pepper, a ham, a handful of potato chips, and so on) is thrown into each hopper, and it is weighed. Given a set $I = \{i \mid i = 1, 2, \ldots, n\}$ of n items in the hoppers, the packing system chooses a subset I' ($\subseteq I$) of items, and puts the items in I' into a package, where the total weight of items in a package is required to be no less than a specified target weight B. The packing system then throws new items into the empty hoppers, and the set I is updated to be the union of the remaining items in $I - I'$ and the new items. Repeating such packing operations, it produces a large number of packages one by one. Note that the system always has to choose some items in the current hoppers without knowing the weights of the next new items that will be thrown into the resulting empty hoppers.

We formulate the problem of choosing a subset I' from I at each packing operation as a 0-1 integer programming problem of off-line setting by using 0-1 variables x_i (e.g., the x_i is set to be zero if item i is chosen, otherwise it is set to be one). In order to avoid customer's complaint, the total weight of items put into each package must be at least B (i.e., the target weight). This is referred to as the *target weight constraint*, which is a hard constraint of the food packing problem [5]. In this paper, two objective functions are considered in the

problem of choosing a subset I'. Let w_i be the weight of item i, which is assumed to be an integer. Then, one objective is to minimize $\sum_{i\in I'} w_i$ under the target weight constraint (i.e., $\sum_{i\in I'} w_i \geq B$). This aims at attaining the total weight of each package as close to the target weight B as possible. If the problem of choosing a subset I' regards only this objective, it can be viewed as the 0-1 knapsack problem. It is known to be an NP-hard problem, but can be solved in $O(n^2 w_{max})$ time by applying dynamic programming if any w_i is bounded by a given integer w_{max} [4]. From the viewpoint of optimizing the total weight of each package, some numerical results for the food packing system have been reported by Kameoka, Nakatani and Inui [3], and by Murakami et al [6, 7]. However, the packing algorithms employed in their papers basically enumerate $O(2^n)$ feasible subsets I'.

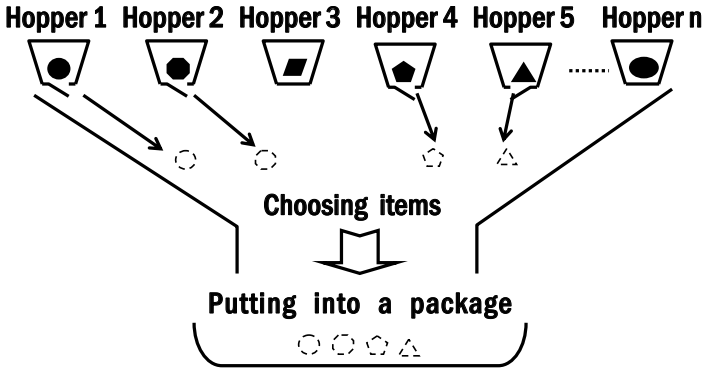

Figure 1: Automated food packing system

During a series of packing operations, an item may stay for a long time in some hopper until it is chosen to be packed [2]. It is undesirable when the packing system handles a kind of fresh (or raw) food. In this paper, a priority γ_i of item i is introduced into the off-line setting to prevent such a situation. The priority γ_i is given by a non-decreasing function of the duration in the hopper of item i. The other objective is to maximize the total priority $\sum_{i\in I'} \gamma_i$.

Karuno, Nagamochi and Ohshima [4] have formulated the problem of choosing a subset I' as a lexicographic bi-criteria optimization problem, in which the first objective is to minimize the total weight $\sum_{i\in I'} w_i$, and the second is to maximize the total priority $\sum_{i\in I'} \gamma_i$ under the target weight constraint. They have modified the $O(n^2 w_{max})$ time algorithm for the 0-1 knapsack problem such that it can solve the lexicographic bi-criteria problem in the same running time [4].

For any instance of the problem of choosing a subset I' at each packing operation, let $W = \sum_{i\in I} w_i$, and let $W_1, W_2, \ldots, W_A$ be total weights, each of which is obtained by some feasible solution I' ($\subseteq I$), such that

$$B \leq W_1 < W_2 < \cdots < W_{A-1} < W_A = W. \tag{1.1}$$

Notice that the positive integer A is bounded by nw_{max} (i.e., $A \leq nw_{max}$) since any w_i is an integer.

The *α-minimum weight solution* is defined as a solution I' (or the corresponding 0-1 vector $x = (x_1, x_2, \ldots, x_n)$) such that $\sum_{i\in I'} w_i = W_\alpha$, where $\alpha \in \{1, 2, \ldots, A\}$. We call the *$\alpha$ quality rank of total weight*. The *α-best solution* is defined as a solution I' that maximizes the total priority $\sum_{i\in I'} \gamma_i$ among α-minimum weight solutions. The *α-nondominated solution* is defined as a solution I' that maximizes the total priority $\sum_{i\in I'} \gamma_i$ among α solutions of

1-best, 2-best, $\cdots$, α-best. At this time, if there exist some solutions with the maximum of total priority, the solution with the least total weight among them is the representative, of course. In addition, if a given α is larger than A in Equation (1.1), it is replaced by the A. From the viewpoint of this definition, the modified $O(n^2 w_{max})$ time algorithm presented by [4] can obtain a 1-nondominated solution in pseudo-polynomial time. In this paper, we extend the modified $O(n^2 w_{max})$ time algorithm to the problem of finding an α-nondominated solution for a given α. The proposed algorithm also runs in $O(n^2 w_{max})$ time. By means of numerical experiments, we also examine the effect of α-nondominated solutions for reducing the maximum duration in the hoppers over all items thrown into the system. The problem at each packing operation with the two objective functions, the total weight $\sum_{i \in I'} w_i$ and the total priority $\sum_{i \in I'} \gamma_i$, is denoted by PACKING for short. In particular, the problem of finding an α-nondominated solution is denoted by PACKING(α). The problem of finding a 1-minimum weight solution is denoted by KNAPSACK. As mentioned above, problem PACKING(α) corresponds to the lexicographic bi-criteria problem when $\alpha = 1$.

The remainder of this paper is organized as follows. In Section 2, we provide the mathematical description of the problem PACKING(α). In Section 3, we propose an $O(n^2 w_{max})$ time algorithm based on dynamic programming to compute an α-nondominated solution. In Section 4, we examine the performance of the proposed approach for the food packing problem by means of numerical experiments, and report the results. In Section 5, we make some concluding remarks.

2. Problem Description

As in Section 1, the number of hoppers in the food packing system is denoted by n. Let N be the iteration number of packing operations (i.e., the number of packages to be produced).

The following inputs give an instance of the PACKING at each packing operation:

- $I = \{i \mid i = 1, 2, \ldots, n\}$: A set of n items. The item thrown into hopper i is referred to as item i (see again Figure 1).
- w_i: Positive integer weight of item $i \in I$, which is assumed to be bounded by a given positive integer w_{max}.
- γ_i: Priority of item $i \in I$, which is assumed to be a positive integer.
- B: Target weight for any package, which is also assumed to be a positive integer.
- $W = \sum_{i=1}^{n} w_i$: Total weight over all n items in I.
- $\Gamma = \sum_{i=1}^{n} \gamma_i$: Total priority over all n items in I.

For notational convenience, the weight vector $w = (w_1, w_2, \ldots, w_n)$ and priority vector $\gamma = (\gamma_1, \gamma_2, \ldots, \gamma_n)$ are used. Since any weight w_i is assumed to be a positive integer, the total weight W is also an integer, and it holds that

$$W \leq n \cdot w_{max}. \tag{2.1}$$

Let C_i be the current duration in the hopper of item i at the ℓ-th iteration of packing operation ($1 \leq \ell \leq N$), which is calculated as follows. The current duration C_i is set to be one if item i is newly thrown into an empty hopper. After the iteration of packing operation, the packing system updates the set I by taking the union of the remaining items in $I - I'$ and new items thrown into empty hoppers. At that time, the current durations of the remaining items are also updated as $C_i := C_i + 1$ (and the current durations of the

new items are set to be one). This implies that $1 \leq C_i \leq \ell$ holds for any $i \in I$ at the ℓ-th iteration of packing operation. In this paper, the priority γ_i is given by a non-decreasing function of the current duration C_i. The simplest way of defining the priority is $\gamma_i := C_i$ for any $i \in I$, which we adopt in this paper.

Let

$$C_{max}(\ell) = \max\{C_i \mid i \in I \text{ at the } \ell\text{-th iteration of packing operation}\}.$$

Then, the *maximum duration* over all items thrown into the system during a series of N packing operations is defined by

$$C_{max} = \max_{1 \leq \ell \leq N} C_{max}(\ell). \tag{2.2}$$

Let n_ℓ be the number of items chosen at the ℓ-th iteration of packing operation (i.e., $|I'|$ at the ℓ-th iteration), and let $D_i(\ell)$ be the eventual duration of such an item $i \in I'$. Let $D_{sum}(\ell) = \sum_{i \in I'} D_i(\ell)$ be the sum of eventual durations of items chosen at the ℓ-th iteration ($1 \leq \ell \leq N-1$), and $D_{sum}(N) = \sum_{i=1}^{n}\{C_i \mid C_i$ is the current duration at the N-th iteration$\}$. Then, we also define the *mean duration* over all items by

$$C_{mean} = \frac{\displaystyle\sum_{\ell=1}^{N} D_{sum}(\ell)}{\displaystyle n + \sum_{\ell=1}^{N-1} n_\ell}. \tag{2.3}$$

Hereafter, instead of a subset I' ($\subseteq I$), a solution of the PACKING (and also of the PACKING(α)) is represented by the 0-1 vector $x = (x_1, x_2, \ldots, x_n)$, where

$$x_i = \begin{cases} 0 & \text{if item } i \text{ is chosen,} \\ 1 & \text{otherwise.} \end{cases} \tag{2.4}$$

In the ordinary 0-1 knapsack problem, the x_i is set to be one if item i is chosen, otherwise it is set to be zero [1]. However, we exchange the roles of two values, zero and one, each other for the convenience of describing the proposed algorithm later.

By using the 0-1 vector x, the bi-criteria problem PACKING is formulated as follows:

PACKING

$$\text{objective 1} \qquad f(x) = \sum_{i=1}^{n} w_i(1 - x_i) \rightarrow \text{minimize} \tag{2.5}$$

$$\text{objective 2} \qquad g(x) = \sum_{i=1}^{n} \gamma_i(1 - x_i) \rightarrow \text{maximize} \tag{2.6}$$

$$\text{subject to} \qquad \sum_{i=1}^{n} w_i(1 - x_i) \geq B, \tag{2.7}$$

$$x_i \in \{0,\ 1\},\ i = 1, 2, \ldots, n. \tag{2.8}$$

The objective by Equation (2.5) aims at attaining the total weight of chosen items as close to the target weight B as possible, together with Equation (2.7) (i.e., the target

weight constraint). The other objective by Equation (2.6) is introduced to expect that the maximum duration C_{max} over all items thrown into the system during a series of packing operations is heuristically minimized (recall that we do not solve a packing problem of on-line setting, but of off-line setting at each packing operation). Equation (2.8) represents the binary constraint of variables.

In particular, for a given quality rank α of total weight, the PACKING(α) asks to find an α-nondominated solution $x = x^{[\alpha]}$ for the two criteria, Equations (2.5) and (2.6). That is, the solution $x = x^{[\alpha]}$ has the maximum total priority $g(x) = g(x^{[\alpha]}) = \Gamma^{[\alpha]}$ among feasible solutions with a total weight no greater than W_α (see again Equation (1.1)).

At each iteration of packing operation, the ordinary food packing system has basically solved the following KNAPSACK with a single objective:

KNAPSACK

$$\text{objective} \qquad f(x) = \sum_{i=1}^{n} w_i(1 - x_i) \to \text{minimize} \tag{2.9}$$

$$\text{subject to} \qquad \sum_{i=1}^{n} w_i(1 - x_i) \geq B, \tag{2.10}$$

$$x_i \in \{0,\ 1\},\ i = 1, 2, \ldots, n. \tag{2.11}$$

The optimal solution $x = \tilde{x}$ of KNAPSACK corresponds to a 1-minimum weight solution in the PACKING(α) (i.e., the total weight of solution $\tilde{x}$ is equal to W_1 in Equation (1.1)).

3. Dynamic Programming Approach

In this section, we propose a pseudo-polynomial time algorithm based on dynamic programming to the PACKING(α). Since the proposed algorithm is an extension of the algorithm for the KNAPSACK, we start with the algorithm to the problem of finding a 1-minimum weight solution. We call the algorithm Simple_DP. Then, we incorporate the priority conception in the Simple_DP. The extended algorithm is referred to as Nondominated_DP(α) for a given α.

This section is organized as follows. In 3.1, we describe the Simple_DP to the KNAPSACK. In 3.2, we extend the Simple_DP to the Nondominated_DP(α). In 3.3, we provide a problem instance of PACKING to illustrate the behavior of Nondominated_DP(α).

3.1. Simple_DP

First, we define 0-1 variables $y_k(p)$ $(k = 1, 2, \ldots, n,\ p = W, W-1, W-2, \ldots, B)$, each of which means that:

$$y_k(p) = 1 \iff \text{There exists a 0-1 vector } (x_1, x_2, \ldots, x_k) \text{ such that } W - \textstyle\sum_{i=1}^{k} w_i x_i = p.$$

$$y_k(p) = 0 \iff \text{There exists no such a 0-1 vector.}$$

The 0-1 variables $y_k(p)$ satisfy the following recursive equations of dynamic programming [1]: For $k = 1, 2, \ldots, n$ and $p = W, W-1, W-2, \ldots, B$,

$$y_1(p) = \begin{cases} 1 & \text{if } p = W \text{ or} \\ & \quad \text{if } p = W - w_1, \\ 0 & \text{otherwise;} \end{cases} \tag{3.1}$$

$$y_k(p) = \begin{cases} 1 & \text{if } y_{k-1}(p) = 1 \text{ or} \\ & \quad \text{if } (p + w_k \le W \text{ and } y_{k-1}(p + w_k) = 1), \\ 0 & \text{otherwise.} \end{cases} \tag{3.2}$$

After computing all $y_k(p)$ $(k = 1, 2, \ldots, n,\ p = W, W-1, W-2, \ldots, B)$, we find a minimum $p = p_{min}$ such that it satisfies $y_n(p) = 1$ and $p \ge B$. It is easy to see that the p_{min} is equal to W_1. Thus, the minimum total weight W_1 can be obtained in $O(nW) = O(n^2 w_{max})$ time (see Equation (2.1)). By backtracking the computation process, we construct in $O(n^2 w_{max})$ time a 1-minimum weight solution $x = \tilde{x}$ (i.e., an optimal solution to the KNAPSACK).

The above result has been known as the following lemma [4].

Lemma 3.1 For an instance of the KNAPSACK, the Simple_DP can obtain an optimal solution $x = \tilde{x}$ in $O(n^2 w_{max})$ time.

3.2. Nondominated_DP(α)

For $k = 1, 2, \ldots, n$, and $p = W, W-1, W-2, \ldots, B$, we incorporate new variables $z_k(p)$ in the Simple_DP to record the total priority $\Gamma - \sum_{i=1}^{k} \gamma_i x_i$ of the 0-1 vector $(x_1, x_2, \ldots, x_k)$ with $W - \sum_{i=1}^{k} w_i x_i = p$. Two kinds of additional variables $u_k(p)$ and $v_k(p)$ are also introduced. If $y_k(p) = 1$ holds due to $y_{k-1}(p) = 1$, the $u_k(p)$ records the total priority $z_{k-1}(p)$, which means that the x_k is set to be zero. On the other hand, if $y_k(p) = 1$ holds due to $p + w_k \le W$ and $y_{k-1}(p + w_k) = 1$, the $v_k(p)$ records the total priority $z_{k-1}(p + w_k) - \gamma_k$, which means that the x_k is set to be one. Let $z_k(p) = \max\{u_k(p), v_k(p)\}$. Then the corresponding 0-1 vector $(x_1, x_2, \ldots, x_k)$ to the $z_k(p)$ has the maximum of total priority among 0-1 vectors with $W - \sum_{i=1}^{k} w_i x_i = p$.

The recursive equations of dynamic programming are provided as follows: For $k = 1, 2, \ldots, n$ and $p = W, W-1, W-2, \ldots, B$,

$$y_1(p) = \begin{cases} 1 & \text{if } p = W \text{ or} \\ & \quad \text{if } p = W - w_1, \\ 0 & \text{otherwise;} \end{cases}$$

$$z_1(p) = \begin{cases} \Gamma & \text{if } p = W, \\ \Gamma - \gamma_1 & \text{if } p = W - w_1, \\ - & \text{otherwise, undefined;} \end{cases} \tag{3.3}$$

$$y_k(p) = \begin{cases} 1 & \text{if } y_{k-1}(p) = 1 \text{ or} \\ & \quad \text{if } (p + w_k \le W \text{ and } y_{k-1}(p + w_k) = 1), \\ 0 & \text{otherwise;} \end{cases}$$

$$u_k(p) = \begin{cases} z_{k-1}(p) & \text{if } (y_k(p) = 1 \text{ and } y_{k-1}(p) = 1), \\ 0 & \text{otherwise;} \end{cases} \tag{3.4}$$

$$v_k(p) = \begin{cases} z_{k-1}(p + w_k) - \gamma_k & \text{if } (y_k(p) = 1 \text{ and} \\ & \quad (p + w_k \le W \text{ and } y_{k-1}(p + w_k) = 1)), \\ 0 & \text{otherwise;} \end{cases} \tag{3.5}$$

$$z_k(p) = \begin{cases} \max\{u_k(p), v_k(p)\} & \text{if } y_k(p) = 1, \\ - & \text{otherwise, undefined.} \end{cases} \tag{3.6}$$

After computing all $y_k(p)$ and all $z_k(p)$, we find a minimum $p = p^{[\alpha]}$ such that it satisfies

$$z_n(p^{[\alpha]}) = \max\{z_n(p) \mid y_n(p) = 1,\ B \le p \le W_\alpha\}.$$

This definition implies that $p^{[1]} = p_{min}$. If all $y_k(p)$ and all $z_k(p)$ have already been computed, the $p^{[\alpha]}$ can be obtained in $O(W) = O(nw_{max})$ time. For the $p^{[\alpha]}$, it is clear that $p^{[\alpha]} \leq W_\alpha$ and $z_n(p^{[\alpha]}) = \Gamma^{[\alpha]}$ $(= g(x^{[\alpha]}))$ hold, where the $x^{[\alpha]}$ denotes an α-nondominated solution. Thus, as in the Simple_DP, the $\Gamma^{[\alpha]}$ can be obtained in $O(nW) = O(n^2 w_{max})$ time. By backtracking the computation process, we construct the solution $x^{[\alpha]}$ also in $O(n^2 w_{max})$ time. The following lemma has been known [4].

Lemma 3.2 For an instance of the PACKING(α) with $\alpha = 1$, the Nondominated_DP(1) can obtain a 1-nondominated solution $x = x^{[1]}$ in $O(n^2 w_{max})$ time.

From the above discussion, we can extend the result to the following theorem:

Theorem 3.1 For an instance of the PACKING(α) with any quality rank α of total weight, the Nondominated_DP(α) can obtain an α-nondominated solution $x = x^{[\alpha]}$ in $O(n^2 w_{max})$ time.

3.3. An example

In this subsection, we provide a problem instance of PACKING to illustrate the behavior of Nondominated_DP(α). The instance consists of five items. Let $I = \{1, 2, 3, 4, 5\}$ be the set of five items. For the I, the weights are given by $w = (3, 7, 5, 8, 2)$, and priorities by $\gamma = (5, 5, 1, 1, 3)$. The total weight of all items is $W = 25$, and total priority of all items is $\Gamma = 15$. The target weight is given by $B = 14$.

First, according to Equations (3.1)$\sim$(3.6), every entry of variables $y_k(p)$ and $z_k(p)$ in Table 1 is filled in. As mentioned in the previous subsection, each entry of the x_k is set to be zero if $y_k(p) = 1$ holds due to $y_{k-1}(p) = 1$, while it is set to be one if $y_k(p) = 1$ holds due to $p + w_k \leq W$ and $y_{k-1}(p + w_k) = 1$. If both of the conditions hold, the x_k is determined based on the choice of $z_k(p)$ (i.e., it is set to be zero if $z_k(p) := u_k(p)$, while it is set to be one if $z_k(p) := v_k(p)$).

The computation process results in Table 1. From this table, we see that the minimum $p = p^{[1]} = p_{min}$ with $y_5(p) = 1$ and $p \geq B$ is 14. Hence, we have $W_1 = 14$ and $\Gamma^{[1]} = 9$ $(= z_5(14))$. The asterisks attached to the values of $y_k(p)$ indicate the backtracking process for the case of $\alpha = 1$. By this indication, we obtain the 1-nondominated solution $x^{[1]} = (1, 0, 0, 1, 0)$.

From Table 1, we also see that

$$W_2 = 15,\ W_3 = 16,\ W_4 = 17,\ W_5 = 18,\ W_6 = 20,\ W_7 = 22,\ W_8 = 23,\ W_9 = W = 25.$$

For $\alpha = 2$ and 3, we have $p^{[2]} = p^{[3]} = 15$ since $\Gamma^{[2]} = \max\{z_5(14), z_5(15)\} = \Gamma^{[3]} = \max\{z_5(14), z_5(15), z_5(16)\} = z_5(15) = 11$. Hence, we obtain the 2- and 3-nondominated solutions $x^{[2]} = x^{[3]} = (0, 0, 0, 1, 1)$. By the similar way, we obtain the 4-, 5-, 6-, 7-, and 8-nondominated solutions $x^{[4]} = x^{[5]} = x^{[6]} = x^{[7]} = x^{[8]} = (0, 0, 0, 1, 0)$, and the 9-nondominated solution $x^{[9]} = (0, 0, 0, 0, 0)$.

4. Numerical Results

In this section, we examine the effect of α-nondominated solutions obtained by the proposed Nondominated_DP(α) for reducing the maximum duration C_{max} over all items thrown into the system during a series of packing operations.

The problem instances to be tested are randomly generated as follows:

Table 1: Behavior of Nondominated_DP(α)

p	$y_1(p)$	$z_1(p)$	x_1	$y_2(p)$	$z_2(p)$	x_2	$y_3(p)$	$z_3(p)$	x_3	$y_4(p)$	$z_4(p)$	x_4	$y_5(p)$	$z_5(p)$	x_5
25	1	15	0	1	15	0	1	15	0	1	15	0	1	15	0
24	0	-	-	0	-	-	0	-	-	0	-	-	0	-	-
23	0	-	-	0	-	-	0	-	-	0	-	-	1	12	1
22	1*	10	1	1*	10	0	1*	10	0	1	10	0	1	10	0
21	0	-	-	0	-	-	0	-	-	0	-	-	0	-	-
20	0	-	-	0	-	-	1	14	1	1	14	0	1	14	0
19	0	-	-	0	-	-	0	-	-	0	-	-	0	-	-
18	0	-	-	1	10	1	1	10	0	1	10	0	1	11	1
17	0	-	-	0	-	-	1	9	1	1	14	1	1	14	0
16	0	-	-	0	-	-	0	-	-	0	-	-	1	7	1
15	0	-	-	1	5	1	1	5	0	1	5	0	1	11	1
14	0	-	-	0	-	-	0	-	-	1*	9	1	1*	9	0

$n = 5$, $w = (3, 7, 5, 8, 2)$, $\gamma = (5, 5, 1, 1, 3)$, $B = 14$

- The number of hoppers: $n \in \{10, 15, 20, 25, 30, 35\}$.
- Integer weights: w_i's are uniformly random integers in each of three types of intervals $[40, 50]$, $[35, 55]$ and $[30, 60]$. In all the intervals, the (expected) mean value of w_i's is $w_{mean} = 45$.
- Target weight: $B \in \{180, 200\}$.
- The iteration number of packing operations: $N = 10000$.
- The quality rank of total weight: $\alpha \in \{1, 2, 3, 10\}$.

The total weight of a typical package with green peppers sold in supermarkets in Japan is around 150 [g]. The way of defining the priority that we adopt in this paper is $\gamma_i := C_i$ for any $i \in I$, where the C_i denotes the current duration in the hopper of item i at the ℓ-th iteration of packing operation ($1 \leq \ell \leq N$). We also give the results of Simple_DP, which can not regard the priorities γ_i ($= C_i$) of items. As an implementation of the Simple_DP, we employ the Nondominated_DP(1), but give a constant priority to each item i, which is independent of the duration C_i of the item.

The program is written in C. It is compiled by Microsoft Visual C++, and run on a personal computer with Intel Pentium M CPU (1.20 GHz) and 1GB memory (Panasonic CF-W4HWSAXC). In all of Table 2∼Table 5, each of the data indicates the mean value for ten series of N packaging operations (and hence 10000×10 packages are produced to obtain each data). The notations used in the tables have the following meanings:

- C_{max}: The maximum duration over all items thrown into the system during a series of N packing operations (see Equation (2.2)).
- C_{mean}: The mean duration for all items thrown into the system during a series of N packing operations (see Equation (2.3)).
- R_B: The accomplished rate of packages with total weight equal to the target weight B produced during a series of N packing operations, i.e.,

$$R_B = \frac{\text{The number of packages with total weight equal to } B}{\text{The iteration number of packing operations, } N} \times 100 \ [\%]$$

Table 2: Performance of the proposed DP [1]

The Number of				Simple_DP		
Hoppers, n	C_{max}	C_{mean}	R_B [%]	W_{mean}	W_{max}	CPU [msec]
10	88	2.5	87.6	180.6	210	0.3
15	810	3.7	96.7	180.2	205	0.7
20	2265	4.9	98.3	180.1	200	1.4
25	5674	6.2	98.7	180.1	202	2.2
30	7144	7.4	99.1	180.0	197	3.3
35	8677	8.6	99.1	180.0	199	4.6
				Nondominated_DP(1)		
n	C_{max}	C_{mean}	R_B [%]	W_{mean}	W_{max}	CPU [msec]
10	20.2	2.5	87.9	180.6	213	0.3
15	24.8	3.7	97.4	180.1	209	0.7
20	25.6	4.9	99.1	180.0	203	1.4
25	25.7	6.2	99.7	180.0	198	2.4
30	27.8	7.4	99.9	180.0	196	3.5
35	27.1	8.6	99.9	180.0	192	5.0
				Nondominated_DP(2)		
n	C_{max}	C_{mean}	R_B [%]	W_{mean}	W_{max}	CPU [msec]
10	13.8	2.5	57.5	181.0	213	0.3
15	17.7	3.7	64.2	180.5	202	0.7
20	20.4	4.9	64.2	180.4	201	1.4
25	20.7	6.2	63.5	180.4	196	2.4
30	21.7	7.4	62.5	180.4	192	3.6
35	21.6	8.5	61.4	180.4	189	4.9
				Nondominated_DP(3)		
n	C_{max}	C_{mean}	R_B [%]	W_{mean}	W_{max}	CPU [msec]
10	10.7	2.5	46.3	181.3	212	0.3
15	14.4	3.7	51.2	180.8	201	0.7
20	16.5	4.9	50.1	180.7	199	1.5
25	17.1	6.2	48.7	180.7	194	2.4
30	17.6	7.4	46.8	180.8	190	3.6
35	18.2	8.5	45.1	180.8	182	4.9
				Nondominated_DP(10)		
n	C_{max}	C_{mean}	R_B [%]	W_{mean}	W_{max}	CPU [msec]
10	5.4	2.4	24.9	183.9	227	0.3
15	7.5	3.6	24.0	183.4	207	0.7
20	8.5	4.8	20.6	183.8	199	1.5
25	10.1	6.0	17.3	184.1	191	2.4
30	10.9	7.2	14.4	184.4	189	3.5
35	12.2	8.4	11.8	184.8	189	4.9

$w_i \in [35, 55]$, $B = 180$

Table 3: Performance of the proposed DP [2]

Weights w_i				Simple_DP		
of Items in:	C_{max}	C_{mean}	R_B [%]	W_{mean}	W_{max}	CPU [msec]
$[40, 50]$	3642	4.8	90.2	182.2	215	1.2
$[35, 55]$	2265	4.9	98.3	180.1	200	1.4
$[30, 60]$	7855	4.9	86.3	180.9	227	1.7
				Nondominated_DP(1)		
w_i in:	C_{max}	C_{mean}	R_B [%]	W_{mean}	W_{max}	CPU [msec]
$[40, 50]$	21.8	4.8	90.9	181.9	210	1.2
$[35, 55]$	25.6	4.9	99.1	180.0	203	1.4
$[30, 60]$	38.9	4.9	97.3	180.2	216	1.6
				Nondominated_DP(2)		
w_i in:	C_{max}	C_{mean}	R_B [%]	W_{mean}	W_{max}	CPU [msec]
$[40, 50]$	19.6	4.8	82.1	182.2	214	1.2
$[35, 55]$	20.4	4.9	64.2	180.4	201	1.4
$[30, 60]$	28.1	4.9	56.7	180.6	216	1.6
				Nondominated_DP(3)		
w_i in:	C_{max}	C_{mean}	R_B [%]	W_{mean}	W_{max}	CPU [msec]
$[40, 50]$	18.0	4.8	80.9	182.3	213	1.2
$[35, 55]$	16.5	4.9	50.1	180.7	199	1.5
$[30, 60]$	24.1	4.9	40.2	180.9	214	1.6
				Nondominated_DP(10)		
w_i in:	C_{max}	C_{mean}	R_B [%]	W_{mean}	W_{max}	CPU [msec]
$[40, 50]$	11.5	4.8	68.8	183.1	217	1.3
$[35, 55]$	8.5	4.8	20.6	183.8	199	1.5
$[30, 60]$	11.1	4.8	8.8	184.8	212	1.6

$n = 20$, $B = 180$

- W_{mean}: The mean value of total weights for N packages.
- W_{max}: The maximum of total weight over all N packages.
- CPU: The computation time of CPU required to obtain an α-nondominated solution at each packing operation.

Recall that the framework of Nondominated_DP(1) is employed in the implementation of Simple_DP. Therefore, the CPU times of Simple_DP in Table 2∼Table 5 are just referential data.

Table 2 shows the results of Simple_DP and Nondominated_DP(α) when the target weight is given by $B = 180$ and the weights w_i of items are generated from $[35, 55]$. In this setting, $B \bmod w_{mean} \equiv 0$ holds. The maximum durations C_{max} by Simple_DP are greater than 800 for instances with $n \geq 15$. On the other hand, the Nondominated_DP(α) has the maximum duration C_{max} less than 30 for any tested instance with up to $n = 35$ and any $\alpha \in \{1, 2, 3, 10\}$. In particular, the Nondominated_DP(1) attains not only the smaller maximum durations C_{max}, but also does accomplished rates R_B greater than 99 [%] for instances with $n \geq 20$. The deviation of the mean value W_{mean} of total weights from the

Table 4: Performance of the proposed DP [3]

The Number of				Simple_DP		
Hoppers, n	C_{max}	C_{mean}	R_B [%]	W_{mean}	W_{max}	CPU [msec]
10	59	2.2	67.2	202.1	240	0.2
15	530	3.3	80.5	201.4	233	0.7
20	5631	4.4	81.9	201.3	234	1.4
25	8714	5.5	82.3	201.3	235	2.4
30	9534	6.6	82.4	201.3	235	3.4
35	9836	7.7	82.6	201.3	234	4.8
				Nondominated_DP(1)		
n	C_{max}	C_{mean}	R_B [%]	W_{mean}	W_{max}	CPU [msec]
10	13.8	2.2	68.3	201.9	238	0.2
15	22.5	3.3	84.8	201.0	233	0.7
20	28.7	4.4	89.1	200.7	231	1.4
25	35.7	5.5	91.4	200.5	231	2.3
30	40.4	6.6	92.8	200.4	230	3.4
35	46.9	7.8	93.9	200.4	226	4.8
				Nondominated_DP(2)		
n	C_{max}	C_{mean}	R_B [%]	W_{mean}	W_{max}	CPU [msec]
10	10.0	2.2	44.7	202.4	237	0.2
15	16.1	3.3	54.3	201.3	238	0.7
20	22.1	4.4	56.1	201.0	231	1.4
25	26.7	5.5	56.1	200.8	229	2.3
30	31.9	6.6	56.0	200.7	228	3.4
35	36.9	7.7	55.3	200.6	229	4.8
				Nondominated_DP(3)		
n	C_{max}	C_{mean}	R_B [%]	W_{mean}	W_{max}	CPU [msec]
10	8.5	2.2	35.7	202.9	238	0.2
15	13.2	3.3	43.0	201.6	233	0.7
20	18.1	4.4	43.6	201.3	231	1.4
25	21.8	5.5	43.7	201.1	228	2.3
30	25.3	6.6	43.4	201.0	226	3.5
35	29.8	7.7	42.7	201.0	223	4.8
				Nondominated_DP(10)		
n	C_{max}	C_{mean}	R_B [%]	W_{mean}	W_{max}	CPU [msec]
10	5.9	2.2	15.5	207.0	241	0.2
15	7.4	3.3	19.6	204.5	220	0.7
20	9.1	4.4	18.9	204.5	209	1.4
25	11.0	5.4	16.8	204.8	209	2.4
30	12.2	6.5	15.3	205.0	209	3.5
35	13.5	7.6	13.7	205.2	209	4.9

$w_i \in [35, 55]$, $B = 200$

Table 5: Performance of the proposed DP [4]

Weights w_i				Simple_DP		
of Items in:	C_{max}	C_{mean}	R_B [%]	W_{mean}	W_{max}	CPU [msec]
$[40, 50]$	10000	4.0	0.0	222.5	244	1.3
$[35, 55]$	5631	4.4	81.9	201.3	234	1.4
$[30, 60]$	8035	4.4	99.9	200.0	213	1.6
				Nondominated_DP(1)		
w_i in:	C_{max}	C_{mean}	R_B [%]	W_{mean}	W_{max}	CPU [msec]
$[40, 50]$	63.4	4.0	0.0	222.5	244	1.2
$[35, 55]$	28.7	4.4	89.1	200.7	231	1.4
$[30, 60]$	9.1	4.4	100	200.0	200	1.5
				Nondominated_DP(2)		
w_i in:	C_{max}	C_{mean}	R_B [%]	W_{mean}	W_{max}	CPU [msec]
$[40, 50]$	54.7	4.0	0.0	222.5	242	1.3
$[35, 55]$	22.1	4.4	56.1	201.0	231	1.4
$[30, 60]$	8.7	4.4	64.0	200.4	201	1.5
				Nondominated_DP(3)		
w_i in:	C_{max}	C_{mean}	R_B [%]	W_{mean}	W_{max}	CPU [msec]
$[40, 50]$	39.9	4.0	0.0	222.5	241	1.3
$[35, 55]$	18.1	4.4	43.6	201.3	231	1.4
$[30, 60]$	8.4	4.4	47.6	200.7	202	1.5
				Nondominated_DP(10)		
w_i in:	C_{max}	C_{mean}	R_B [%]	W_{mean}	W_{max}	CPU [msec]
$[40, 50]$	12.2	4.0	0.0	222.5	240	1.3
$[35, 55]$	9.1	4.4	18.9	204.5	209	1.4
$[30, 60]$	7.8	4.4	12.1	204.3	209	1.5

$n = 20$, $B = 200$

target weight B is less than 1 by Simple_DP and by Nondominated_DP(1) as well. For a larger α, the Nondominated_DP(α) obtains a smaller C_{max}, but its R_B and W_{mean} are getting worse. A remarkable difference between Simple_DP and Nondominated_DP(α) is not observed with respect to the mean duration C_{mean}.

An existent food packing system with fourteen hoppers (i.e., $n = 14$) can mechanically weigh the items for about 140 packages per minute [5]. The CPU time of Nondominated_DP(α) is less than 5.0 [msec] even for instances with $n = 35$, and therefore it can be good for practical use.

Table 3 provides a referential result, in which the intervals generating weights w_i of items are alternated. The number of hoppers is fixed to $n = 20$, and the target weight is given by $B = 180$.

Table 4 shows the results of Simple_DP and Nondominated_DP(α) when the target weight is given by $B = 200$ and the weights w_i of items are generated in $[35, 55]$. It does not hold that $B \bmod w_{mean} \equiv 0$. Each of the accomplished rate R_B of Simple_DP and Nondominated_DP(1) is worse than that in Table 2 where $B = 180$. However, the Nondominated_DP(α) shows better performance than Simple_DP with respect to the maximum

duration C_{max}, as observed in Table 2.

Table 5 also provides a referential result, in which the intervals generating weights w_i of items are alternated. The number of hoppers is fixed to $n = 20$, and the target weight is given by $B = 200$. The accomplished rate R_B seems to be affected not only by the relation between the target weight B and mean weight w_{mean}, but also by the range of weights w_i.

Anyway, from these results, we can conclude that the proposed approach has a significant effect for reducing the maximum duration C_{max} over all items thrown into the system during a series of packing operations.

5. Concluding Remarks

In this paper, we considered a discrete optimization problem arising in a certain type of automated food packing system with n weighing hoppers, so-called automatic combination weigher. We formulated it as a bi-criteria optimization problem, denoted by PACKING(α), and proposed an $O(n^2 w_{max})$ time algorithm based on dynamic programming, where the w_{max} denotes the upper bound of the weight of each item. The proposed approach aimed at minimizing the maximum duration in the system of items heuristically, while attaining the total weight of each package as close to the target weight as possible. By numerical experiments, we observed that the proposed approach has a significant effect for reducing the maximum duration.

It is left for the future research to examine the effect of some different way of defining the priority of each item. In the numerical experiments, we solved only instances with $w_{max} < n^2$. Since the precision of weighers installed in the hoppers may be highly improved, the CPU time of the proposed algorithm should also be reported for instances with $w_{max} \geq n^2$.

Acknowledgement

We would like to express our gratitude to the anonymous referees whose constructive suggestions contributed to improving the written style of the problem description. This research was partially supported by a Scientific Grant in Aid from the Ministry of Education, Culture, Sports, Science and Technology of Japan.

References

[1] T. Ibaraki: *Algorithms and Data Structure* (in Japanese) (Shokodo Co., Ltd., Tokyo, 1989).

[2] 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.

[3] 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.

[4] Y. Karuno, H. Nagamochi, and Y. Ohshima: A dynamic programming approach for a food packing problem (in Japanese). *Transactions of the Japan Society of Mechanical Engineers, Series C*, **72** (2006), 1390–1397.

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

[6] Y. Murakami, J. Kurata, H. Uchiyama, K. Taniya, and M. Kawai: Efficient algorithm for solving a bag-packing problem by excluding search space (in Japanese). *Transactions of the Japan Society of Mechanical Engineers, Series C*, **69** (2003), 3431–3438.

[7] Y. Murakami, J. Kurata, H. Uchiyama, and T. Ueno: Characterization of infeasible solutions in a bag-packing problem for achieving desired weight (in Japanese). *Transactions of the Society of Instrument and Control Engineers*, **38** (2002), 784–791.

Yoshiyuki Karuno
Department of Mechanical and
System Engineering
Graduate School of Science and Technology
Kyoto Institute of Technology
Matsugasaki, Sakyo
Kyoto 606-8585, Japan
E-mail: karuno@kit.ac.jp