Iterative improvement approaches for collecting weighted items in directed bipartite graphs

Yoshiyuki KARUNO* and Seiya TANAKA**
* Faculty of Mechanical Engineering, Kyoto Institute of Technology
Goshokaido-cho, Matsugasaki, Sakyo-ku, Kyoto-shi, Kyoto 606-8585, Japan
E-mail: karuno@kit.ac.jp
** Graduate School of Science and Technology, Kyoto Institute of Technology
Goshokaido-cho, Matsugasaki, Sakyo-ku, Kyoto-shi, Kyoto 606-8585, Japan

Received: 7 November 2017; Revised: 15 April 2018; Accepted: 13 May 2018

Abstract
In this paper, an iterative improvement heuristic based on the simulated annealing is designed for a weighted item collecting problem in directed bipartite graphs. The weighted item collecting problem is a generalization of an integrated circuit design problem, and it is also a variant of 0-1 knapsack problems in graphs. Recently, a greedy heuristic algorithm has been presented for the weighted item collecting problem. The greedy heuristic algorithm obtains an optimal solution of a known test instance of the original integrated circuit design problem in a short execution time, while for a randomly generated instance, there may be some room for improvement of the heuristic quality. In order to search for a better heuristic solution, some neighborhood structures for an incumbent solution are defined, and each of them is embedded in a framework of the simulated annealing. Numerical experiments are conducted to examine the iterative improvement performance, and the results are reported.

Keywords: Engineering optimization, 0-1 knapsack problems, Directed bipartite graphs, Arc reversing strategy, Simulated annealing

1. Introduction

The weighted item collecting problem in directed bipartite structures to be discussed in this paper is described as follows. Let \( I = \{ i \mid i = 1, 2, \ldots, m \} \) denote a set of \( m \) items, and let \( J = \{ j \mid j = 1, 2, \ldots, n \} \) denote a set of \( n \) players. Two non-negative weights \( w_i \) and \( w_{i+1} \) such that \( w_i + w_{i+1} > 0 \) are associated with each item \( i \in I \). We regard each of the two weights as a profit of the item. A positive cost \( c_j \) is associated with each player \( j \in J \). On the \( n \) players as a team, a positive budget \( b \) is imposed.

There are two kinds of signals, by which the items and the players are connected. For convenience, we assign a color, either blue or red, to each kind of the connection signals. For each player \( j \in J \), let \( B_j \subseteq I \) (resp., \( R_j \subseteq I \)) denote the set of items, each of which is initially connected with the player \( j \in J \) by a blue signal (resp., by a red signal). It must be satisfied that \( B_j \cap R_j = \emptyset \) for any player \( j \in J \). On the other hand, without loss of generality, it is assumed that

\[
B_1 \cup R_1 \cup B_2 \cup R_2 \cup \cdots \cup B_n \cup R_n = I,
\]

and each player \( j \in J \) is assumed to meet

\[
B_j \cup R_j \neq \emptyset.
\]

We say that a player \( j \in J \) is reversed when changing every color of the initial connection signals associated with the player \( j \in J \) to the opposite one (i.e., either from blue to red, or from red to blue). As a solution of the weighted item collecting problem, a reversing vector \( x = (x_1, x_2, \ldots, x_n) \) is defined, where for each \( j = 1, 2, \ldots, n \),

\[
x_j = \begin{cases} 1 & \text{if player } j \text{ is reversed,} \\ 0 & \text{otherwise.} \end{cases}
\]
That is, for a reversed player \( j \in J \) in a reversing vector \( x \) (i.e., for a player \( j \in J \) with \( x_j = 1 \)), an item \( i \in B_j \) (resp., an item \( i \in R_j \)) is connected with the player by a red signal (resp., by a blue signal). A signal color (i.e., either blue or red) between an item and an associated player with the item can be represented by a direction of an arc when regarding the set of \( m \) items, the set of \( n \) players, and the connection between them as a directed bipartite graph. We are going to illustrate an example of such a directed bipartite graph in the following section.

When the \( n \) players take a reversing vector \( x \) as their arc reversing strategy, they pay the reversing cost

\[
c(x) = \sum_{j=1}^{n} c_j x_j.
\]

A reversing vector \( x \) is referred to as feasible if it satisfies the budget constraint, i.e., \( c(x) \leq b \). Notice that the zero reversing vector \( x = (x_1, x_2, \ldots, x_n) \) with \( x_j = 0 \) for all players \( j \in J \) is feasible, since the \( x = 0 \) meets \( c(0) = 0 \leq b \).

For a feasible reversing vector \( x \), let \( I_B(x) \subseteq I \) (resp., \( I_R(x) \subseteq I \)) denote the set of items, each of which has only blue connection signals (resp., only red connection signals). We regard an item \( i \in I_B(x) \cup I_R(x) \) with common color connection signals as a successful result of the cooperation of players associated with the item, and the \( n \) players as a team can get the profit \( w_i \) (resp., \( w_{m+1} \)) when \( i \in I_B(x) \) holds (resp., when \( i \in I_R(x) \) holds). The total profit as the objective function to be maximized in this paper is represented by

\[
f(x) = \sum_{i \in I_B(x)} w_i + \sum_{i \in I_R(x)} w_{m+1}.
\]

The weighted item collecting problem asks to find a feasible reversing vector \( x = x^* \) which maximizes the total profit \( f(x) \) defined in Eq. (3). We may call the \( x^* \) an optimal solution in the following.

The weighted item collecting problem is viewed as a generalization of the minimal switching graph problem (MSG for short) treated by Tang (2005). Problem MSG has been introduced by Tang et al. (1999) to model a constrained via minimization in the context of integrated circuit design automation. That is, the set of items in the weighted item collection problem corresponds to a set of given via candidates in a double-sided circuit board, and the set of players expresses a set of given wiring clusters. A reversed player represents a wiring cluster to be moved from its initial side of the circuit board to the opposite side. The total profit of Eq. (3) indicates the weighted number of eliminative via candidates. The weighted item collecting problem involves the budget constraint of Eq. (2), while problem MSG is an enough budget case (i.e., \( b \geq \sum_{j=1}^{n} c_j \)) with the unit profit for each item (i.e., \( w_i = w_{m+1} = 1 \) for each \( i \in I \)). Hence, the weighted item collecting problem is a more general model than problem MSG to make a compromise in a given conflictive state among the players.

An application of the genetic algorithm to problem MSG has been reported in Tang (2005), and a greedy heuristic algorithm has recently been presented by the authors. Both of the genetic and the greedy heuristic algorithms have empirically obtained an optimal solution of the instance with \( m = n = 48 \) provided as a benchmark of problem MSG in Tang (2005). In particular, the greedy heuristic algorithm finds the optimal solution in a short execution time with less than one millisecond on an ordinary laptop computer (see Karuno and Tanaka, 2017a). Also, it has been designed to handle heuristically the budget constraint of an arc reversing strategy. On the other hand, for a randomly generated instance of the weighted item collecting problem, there may be some room for improvement of the heuristic quality (see again Karuno and Tanaka, 2017a). Unfortunately, no randomly generated instance was tested for the genetic algorithm in Tang (2005).

In this paper, an iterative improvement heuristic based on the simulated annealing (e.g., see Aarts and Korst, 1989) is designed in order to search for a better heuristic solution starting with a greedy heuristic one. Especially, some neighborhood structures for an incumbent solution are defined, and each of them is embedded in a framework of the simulated annealing. Instead of the direct expression of a reversing vector, we also utilize a sequence of \( n \)-bit strings, called basic semi-solutions, as in a greedy heuristic algorithm (see Karuno and Tanaka, 2017a). Numerical experiments are conducted to examine the iterative improvement performance, and the results are reported, e.g., a multi-insertion type of neighborhood structure delivers a better improvement from the greedy heuristic solution.

2. Underlying Properties

2.1. Bipartite Representation of a State of Connection between Items and Players

Let \( G(x) = (I \cup J, A_B(x) \cup A_R(x)) \) denote a reversing graph with respect to a reversing vector \( x \) as the directed bipartite structure, where \( A_B(x) \) and \( A_R(x) \) are disjoint arc sets such that \( A_B(x) \subseteq I \times J \) and \( A_R(x) \subseteq J \times I \). More precisely, an arc \((i, j) \in A_B(x)\) (resp., an arc \((j, i) \in A_R(x)\)) indicates a blue (resp., a red) signal between item \( i \in I \) and player \( j \in J \).
In particular, let $G(0) = (I \cup J, A_B(0) \cup A_R(0))$ denote the initial reversing graph with respect to the zero reversing vector $x = 0$.

In Fig. 1, we provide an illustration of the initial reversing graph $G(0)$ and another reversing graph $G(x)$ with respect to $x = (1, 0, 0, 1, 0, 1, 0)$, where the number of items is $m = 7$ and the number of players is $n = 7$. In (a) of Fig.1, for example, we see that the fourth player in the initial reversing graph $G(0)$ has the connection with items represented by $B_4 = [3, 6]$ and $R_4 = [1, 4]$. In the $G(0)$, each item $i \in I$ has both colors of connection signals, and there is no item with common color signals, i.e., $I_R(0) = I_B(0) = 0$. Hence, we see the total profit $f(0) = 0$ of the zero reversing vector $x = 0$.

In (b) of Fig.1, three players $j = 1, 4$ and $6$ are reversed, and we notice $I_B(x) = [1, 2, 4]$ as the cooperation in the sense of blue color and $I_R(x) = [3, 6, 7]$ as the cooperation in the sense of red color. Henceforth, we may call an item in $I_B(x)$ (resp., in $I_R(x)$) a completely blue-signaled (resp., a completely red-signaled) one. The players pay the reversing cost $c(x) = c_1 + c_4 + c_6$. If the budget constraint is met, i.e., $b \geq c_1 + c_4 + c_6$, then the reversing vector $x = (1, 0, 0, 1, 0, 1, 0)$ is feasible, and it brings the total profit $f(x) = w_1 + w_2 + w_4 + w_{m+3} + w_{m+6} + w_{m+7}$.

![Fig. 1](image)

**2.2. Basic Semi-solutions**

The greedy heuristic algorithm presented by the authors has utilized an important property of basic semi-solutions, each of which can get a profit of at least one item (see again Karuno and Tanaka, 2017a; Tang, 2005) when $c_j \leq b$ holds for each player $j \in J$. In this paper, we also utilize the basic semi-solutions in a framework of the simulated annealing. In this section, we provide an overview of them, following Karuno and Tanaka (2017a).

For a reversing vector $x = (x_1, x_2, \ldots, x_n)$, the counterpart vector (or simply, counterpart) of the $x$ is defined by $\overline{x} = (\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n)$, where for each player $j = 1, 2, \ldots, n$, it holds

$$\overline{x}_j = 1 - x_j. \quad (4)$$

The definition implies that the counterpart of the $\overline{x}$ is the original reversing vector $x$. Further, $I_B(x) = I_R(\overline{x})$ and $I_R(x) = I_B(\overline{x})$ hold for a reversing vector $x$ and the counterpart $\overline{x}$. For example, consider a reversing vector $x = (1, 0, 0, 1, 0, 1, 0)$ and the counterpart $\overline{x} = (0, 1, 1, 0, 1, 0, 1)$ in (b) of Fig. 1. Then, we see $I_B(x) = I_R(\overline{x}) = \{1, 2, 4\}$ and $I_R(x) = I_B(\overline{x}) = \{3, 6, 7\}$.

A reversing vector $x = (x_1, x_2, \ldots, x_n)$ can be regarded as a string of $n$ binary bits. Introducing the ordinary symbol $*$ for representing “no care” (e.g., see Tang, 2005), a semi-solution is defined by $s = [s_1, s_2, \ldots, s_n]$, where for each player $j = 1, 2, \ldots, n$,

$$s_j = \begin{cases} 1 & \text{if player } j \text{ is reversed,} \\ 0 & \text{if player } j \text{ is not reversed,} \\ * & \text{if player } j \text{ is not cared.} \end{cases} \quad (5)$$

The counterpart semi-solution of the $s$ is also defined by $\overline{s} = [\overline{s}_1, \overline{s}_2, \ldots, \overline{s}_n]$, where for each $j = 1, 2, \ldots, n$,

$$\overline{s}_j = \begin{cases} 1 - s_j & \text{if } s_j \in \{0, 1\}, \\ * & \text{otherwise (i.e., if } s_j = *\text{).} \end{cases} \quad (6)$$

A blue basic (resp., a red basic) semi-solution $\overline{s}^{(b)}$ (resp., $\overline{s}^{(m+b)}$) for each item $i \in I$ is defined to summarize all the solutions taking the profit $w_i$ (resp., the profit $w_{i+m}$) of the item as a completely blue-signaled one (resp., as a completely
red-signaled one). Notice that the counterpart semi-solution of a blue basic semi-solution is the corresponding red basic semi-solution, i.e., for each item \(i \in I\), it holds that \(s^{(m-n)} = \bar{s}^{(i)}\). For the problem instance provided in Fig. 1, there are the following \(m = 7\) blue basic semi-solutions:

\[
\begin{align*}
    s^{(1)} &= [1, 0, *, 1, *, *, *], \\
    s^{(2)} &= [*, 0, 0, *, *, 1, *], \\
    s^{(3)} &= [*, 1, *, 0, 1, *, *], \\
    s^{(4)} &= [*, *, *, 1, 0, *, *], \\
    s^{(5)} &= [*, 1, *, *, 0, 0, *], \\
    s^{(6)} &= [*, *, *, 0, 1, 0, *], \\
    s^{(7)} &= [*, *, *, *, *, *, 0, 1],
\end{align*}
\]

and the following \(m = 7\) red basic semi-solutions:

\[
\begin{align*}
    s^{(8)} &= \bar{s}^{(1)} = [0, 1, *, 0, *, *, *], \\
    s^{(9)} &= \bar{s}^{(2)} = [*, 1, 1, *, *, 0, *], \\
    s^{(10)} &= \bar{s}^{(3)} = [*, 0, *, 1, 0, *, *], \\
    s^{(11)} &= \bar{s}^{(4)} = [*, *, *, 0, 1, *, *], \\
    s^{(12)} &= \bar{s}^{(5)} = [*, 0, *, *, 1, 1, *], \\
    s^{(13)} &= \bar{s}^{(6)} = [*, *, *, 1, 0, 1, *], \\
    s^{(14)} &= \bar{s}^{(7)} = [*, *, *, *, *, 1, 0].
\end{align*}
\]

A reversing vector \(x = (x_1, x_2, \ldots, x_n)\) is said to contain a semi-solution \(s = [s_1, s_2, \ldots, s_n]\) if \(x_j = s_j\) holds for any player \(j \in J\) with \(s_j \in \{0, 1\}\), and also in this paper, it is expressed by \(x \supseteq s\). For a semi-solution \(s = [s_1, s_2, \ldots, s_n]\) and a reversing vector \(x = (x_1, x_2, \ldots, x_n)\) with \(x \supseteq s\), if \(s_j = *\) implies \(x_j = 0\), then the \(x\) is the minimal solution for the \(s\). Obviously, from a semi-solution, the minimal solution can be obtained in \(O(n)\) time (Karuno and Tanaka, 2017a). Further, the minimal cost of a semi-solution \(s = [s_1, s_2, \ldots, s_n]\) is defined by

\[
c_{\min}(s) = \sum \{c_j \mid j \in J \text{ with } s_j = 1\},
\]

which is clearly equal to the reversing cost of the minimal solution for the \(s\).

2.3. Compoundable Semi-solutions

We say that semi-solutions \(s = [s_1, s_2, \ldots, s_n]\) and \(s' = [s'_1, s'_2, \ldots, s'_n]\) are inconsistent if there exists some player \(j \in J\) who meets all the following three conditions:

(i) \(s_j \neq s'_j\),

(ii) \(s_j \neq *\),

(iii) \(s'_j \neq *\).

Otherwise, they are consistent (see Karuno and Tanaka, 2017a). The definition implies that for any item \(i \in I\), the blue basic and red basic semi-solutions \(s^{(i)}\) and \(\bar{s}^{(i)}\) are inconsistent. For example, in the problem instance provided in Fig. 1, two semi-solutions \(s^{(1)}\) and \(\bar{s}^{(3)}\) are consistent. For two distinct semi-solutions, their consistency can obviously be checked in \(O(n)\) time by Eq. (8).

For two distinct semi-solutions \(s\) and \(s'\), suppose that they are consistent. Then, we obtain a semi-solution \(\hat{s}\) from the two semi-solutions \(s\) and \(s'\) such that for each player \(j = 1, 2, \ldots, n\), it holds

\[
\hat{s}_j = \begin{cases} 
    s_j & \text{if } s_j = s'_j \in \{0, 1\}, \\
    s_j & \text{if } s_j \in \{0, 1\} \text{ and } s'_j = *, \\
    s'_j & \text{if } s_j = * \text{ and } s'_j \in \{0, 1\}, \\
    * & \text{if } s_j = s'_j = *,
\end{cases}
\]

since no player \(j \in J\) meets Eq. (8). For the semi-solution \(\hat{s}\), let \(\hat{x}\) temporarily denote the minimal solution. Then, both of \(\hat{x} \supseteq s\) and \(\hat{x} \supseteq s'\) hold (see Karuno and Tanaka, 2017a). The \(\hat{s}\) is called a compound semi-solution of the consistent \(s\) and
For example, as seen in the above, two semi-solutions $s^{(1)}$ and $s^{(2)}$ in Fig. 1 are consistent, which implies that there is a reversing vector such that if it is feasible, it can get the total profit no less than $w_j + w_{m+1}$.

We are going to utilize the following procedure for finding a compound semi-solution in the proposed heuristic algorithm based on the simulated annealing:

**Procedure** COMPOUNDING ($s, s', \delta$)

Input: Two distinct semi-solutions $s = [s_1, s_2, \ldots, s_n]$ with $c_{\text{min}}(s) \leq b$ and $s' = [s'_1, s'_2, \ldots, s'_n]$.

Output: A semi-solution $\delta = [\delta_1, \delta_2, \ldots, \delta_n]$ (such that it is either a compound semi-solution of the $s$ and $s'$ or $\delta = s$).

Step 1. If the two semi-solutions $s$ and $s'$ are consistent, then go to Step 2; otherwise (i.e., if they are inconsistent), go to Step 4.

Step 2. Compute a compound semi-solution $\delta$ of the given two semi-solutions $s$ and $s'$ by Eq. (9).

Step 3. If $c_{\text{min}}(\delta) \leq b$ holds, then go to Step 5; otherwise (i.e., if $c_{\text{min}}(\delta) > b$ holds), go to Step 4.

Step 4. Let $\delta := s$ (when either the $s$ and $s'$ are inconsistent, or the minimal solution $\delta$ for the compound $\delta$ is infeasible).

Step 5. Return the semi-solution $\delta$.

We remark that the minimal solution for the first input semi-solution $s$ is feasible since $c_{\text{min}}(s) \leq b$ must be required, and the minimal solution for the output semi-solution $\delta$ is also feasible. The time complexity of procedure COMPOUNDING is $O(n)$, since each step of the procedure runs in $O(n)$ time. A greedy heuristic algorithm presented by the authors has applied the procedure to the $2m$ basic semi-solutions one by one in a non-increasing order of their profits (see Karuno and Tanaka, 2017a). Our iterative improvement heuristic based on the simulated annealing is also going to apply the procedure to an incumbent sequence of the $2m$ basic semi-solutions. The proposed heuristic algorithm in this paper is obviously a different approach from the existing genetic algorithm which maintains a set of reversing vectors $x = (x_1, x_2, \ldots, x_n)$ in the searching process (see Tang, 2005).

3. An Application of the Simulated Annealing

The simulated annealing is an algorithm based on the local search technique, and it is an example of generally applicable heuristics (e.g., see Aarts and Korst, 1989; Ibaraki, 1989; Skiena, 2008). As shown in the name, the searching process is often introduced by following an analogy between the solutions of a combinatorial optimization problem and the physical states of a many-particle system in a thermal process. That is, the simulated annealing involves parameters such as a temperature, a cooling rate, and so on. The local search and the simulated annealing both accept a neighboring solution of an incumbent one whenever it is superior to the incumbent, i.e., with probability one. A symbolic behavior of the simulated annealing as a generalization of the local search is to accept an inferior solution to an incumbent solution with a certain probability in order to escape a poor local optimum, while an ordinary local search algorithm never accepts an inferior solution.

When we attempt to apply the simulated annealing to a combinatorial optimization problem, we define a representation manner of the solutions and a neighborhood structure of a solution. As mentioned in the previous section, we treat a sequence of the $2m$ basic semi-solutions in the searching process of the simulated annealing as the representation of a reversing vector $x = (x_1, x_2, \ldots, x_n)$, and define a neighborhood structure also for a sequence of the $2m$ basic semi-solutions. The framework of the simulated annealing to be employed in this paper is shown in Algorithm 1, where the $2m$ basic semi-solutions $s^{(0)}$ and $s^{(mn+m)} = \pi^{(0)}$ have been obtained by an $O(mn)$ time preprocessing (see Tang, 2005). The acceptance probability for an inferior solution is described by Line 25 in Algorithm 1. More details of the proposed heuristic algorithm are explained as follows.

3.1. A Transformation from a Compounding Sequence to a Reversing Vector

For an instance of the weighted item collecting problem, let $X$ denote the set of all the reversing vectors $x$, which satisfies $|X| = 2^n$, and let $F \subseteq X$ denote the feasible region of the $X$, i.e., any reversing vector $x \in F$ meets the budget constraint. Also, let $\sigma = [\sigma[1], \sigma[2], \ldots, \sigma[2m]]$ denote a permutation on $[1, 2, \ldots, 2m]$, which we call a compounding sequence of the $2m$ basic semi-solutions. For notational convenience, we represent by $s = s'$ the semi-solution $s = [s_1, s_2, \ldots, s_n]$ with $s_j = \pi$ for all $j \in J$. The minimal solution for the $s'$ is the zero reversing vector $x = 0$. For a compounding sequence $\sigma$, by calling procedure COMPOUNDING $O(m)$ times, starting with the semi-solution $s'$, we can obtain a feasible reversing vector $x \in F$, since $c_{\text{min}}(s') = c(0) \leq b$. (Recall that procedure COMPOUNDING returns a semi-solution whose minimal solution is feasible whenever the minimal solution of the first input semi-solution is feasible.)
Algorithm 1 SIMULATED ANNEALING

Input: A set \( I \) of \( m \) items, a set \( J \) of \( n \) players, non-negative weights \( w_i \) and \( u_{\text{inc}} \) such that \( w_i + u_{\text{inc}} > 0 \) for each \( i \in I \), a positive cost \( c_j \) for each player \( j \in J \), a positive budget \( b \), blue and red basic semi-solutions for each item \( i \in I \), \( s_i^{(j)} \) and \( s_i^{(m)} (= s_i^{(j)}) \), respectively.

Parameter settings: The initial temperature \( T_{\text{init}} \), the final temperature \( T_{\text{max}} (= T_{\text{init}}) \), a cooling rate \( \alpha \) with \( \alpha < 1 \), the number \( \ell_{\text{max}} \) of iterations in a temperature, a positive upper limit \( u_{\text{max}} \) on the number of iterations with no improvement of the best solution in the searching process, a normalization factor \( \kappa \) in the acceptance probability for an inferior solution.

Output: A feasible heuristic solution \( x' = (x'_1, x'_2, \ldots, x'_n) \).

1: Initialize the compounding sequence \( \sigma' = \sigma[1], \sigma[2], \ldots, \sigma[2m] \) of the \( 2m \) basic semi-solutions such that
   \[ w_{\sigma[1]} \geq w_{\sigma[2]} \geq \cdots \geq w_{\sigma[2m]} ; \]
2: Compute a greedy heuristic solution \( x_{\text{inc}} \) as the initial solution: /* The \( x_{\text{inc}} \) stores an incumbent solution during the searching process. */
3: \( f_{\text{inc}} := f(x_{\text{inc}}); x' := x_{\text{inc}}; f' := f(x') */ The \( x' \) stores the best solution during the searching process. */
4: if \( (f' < f(0)) \) then
5: \( x' := 0; f' := f(x')\)
6: end if
7: \( t := t_{\text{max}}; u := 0; \) /* The \( u \) indicates the cumulative number of iterations after the last updating of the best solution. */
8: while \( (t > t_{\text{min}} \text{ and } u < u_{\text{max}}) \) do /* This is the termination test of the searching process. */
9: for \( \ell = 1 \) to \( \ell_{\text{max}} \) do
10: Generate a neighboring sequence \( \sigma_N = (\sigma_{N[1]}, \sigma_{N[2]}, \ldots, \sigma_{N[2m]}) \in \mathcal{N}(\sigma) \) of the current compounding sequence \( \sigma; \)
11: \( s := x' \)
12: for \( k = 1 \) to \( 2m \) do
13: \( s' := s^\text{SWAP}; \) Call procedure \( \text{COMPOUNDING}(s, s'; \delta); s := \delta; \) /* The counterpart of \( x \) is not checked at this time in the primitive version. */
14: end for
15: Obtain the minimal solution \( x \) for the semi-solution \( s \) constructed from the neighboring sequence \( \sigma_N; \)
16: /* The \( x \) is the number of iterations with no improvement of the best solution in the searching process. */
17: \( u := u + 1; \)
18: if \( (f(x) \geq f_{\text{inc}}) \) then
19: \( x_{\text{inc}} := x; f_{\text{inc}} := f(x_{\text{inc}}); \sigma := \sigma_N; \) /* A superior \( x \) to the incumbent \( x_{\text{inc}} \) is always accepted as the next incumbent. */
20: if \( (f(x) > f') \) then
21: \( x' := x; f' := f(x')\); \( u := 0; \) /* This is the updating of the best solution. */
22: end if
23: else
24: \( \Delta := f_{\text{inc}} - f(x); \)
25: if \( \left( \exp(-\Delta/(s_k \times t)) \geq \text{random}(0, 1) \right) \) then
26: \( x_{\text{inc}} := x; f_{\text{inc}} := f(x_{\text{inc}}); \sigma := \sigma_N; \) /* An inferior \( x \) is accepted as the next incumbent with the above probability. */
27: end if
28: end if
29: end for
30: \( t := t + t; \) /* This is the cooling step. */
31: end while
32: return \( x' \).

The proposed heuristic algorithm first sets the initial compounding sequence \( \sigma = (\sigma[1], \sigma[2], \ldots, \sigma[2m]) \) to be a non-increasing order of the \( 2m \) basic semi-solutions with their profits, i.e., \( w_{\sigma[1]} \geq w_{\sigma[2]} \geq \cdots \geq w_{\sigma[2m]} \), and the initial incumbent solution to be a greedy heuristic solution (see Lines 1 and 2 in Algorithm 1). The greedy heuristic algorithm to compute the initial incumbent solution is a version of greedy manners presented by the authors (see Karuno and Tanaka, 2017a), and the detail is provided in Appendix.

The transformation from a compound sequence \( \sigma \) of the \( 2m \) basic semi-solution to a feasible reversing vector \( x \) is performed during Lines 10–15 in Algorithm 1. For Line 10, some neighborhood structures \( \mathcal{N}(\sigma) \) of a current compounding sequence \( \sigma \) are defined in the following section.

3.2. Neighborhood Structures of a Compounding Sequence

Let \( \mathcal{S} \) denote the set of all compounding sequences \( \sigma \) of the \( 2m \) basic semi-solutions, where \( |\mathcal{S}| = (2m)! \) holds. For a compounding sequence \( \sigma = (\sigma[1], \sigma[2], \ldots, \sigma[2m]) \in \mathcal{S} \), one of typical neighborhood structures may be the swapping, i.e.,

\[
\mathcal{N}_{\text{SWAP}}(\sigma) = \{ \sigma_N \in \mathcal{S} \mid \sigma_N = (\sigma[1], \ldots, \sigma[k - 1], \sigma[k+1], \ldots, \sigma[\ell - 1], \sigma[k], \sigma[\ell +1], \ldots, \sigma[2m]), \]
\[
1 \leq k < \ell \leq 2m, \]

for which we understand that the size is \( |\mathcal{N}_{\text{SWAP}}(\sigma)| = O(m^2) \).

Another typical neighborhood structure may be the insertion. Since the commutative property holds for the compounding of two consistent semi-solutions (see Eq. (9)), we use two particular neighborhood structures of the insertion,
the so-called move-to-front (MtF for short) and an opposite version, move-to-rear (MtR for short), i.e.,
\[ N_{\text{MtF}}(s) = \{ \sigma_N \in S | \sigma_N = (\sigma[k], \sigma[1], \ldots, \sigma[k-1], \sigma[k+1], \ldots, \sigma[2m]), 1 < k \leq 2m \}, \]
\[ N_{\text{MtR}}(s) = \{ \sigma_N \in S | \sigma_N = (\sigma[1], \ldots, \sigma[k-1], \sigma[k+1], \ldots, \sigma[2m], \sigma[k]), 1 \leq k < 2m \}, \]
for which we see \(|N_{\text{MtF}}(s)| = |N_{\text{MtR}}(s)| = O(m)\).

We also define a restricted version of the MtF, and that of the MtR peculiarly for the weighted item collecting problem. As seen in Section 3.1, for a compounding sequence \( s^* \), we can obtain a feasible reversing vector \( x \in F \). Let \( s = s(\sigma) \) denote the semi-solution constructed from a compounding sequence \( \sigma \) by the \( O(m) \) callings of procedure COMPONDING, and let \( x = x(\sigma) \) denote the minimal solution for the \( s(\sigma) \). We have seen that \( s(\sigma) \in F \). We further define the following two subsets of basic semi-solutions:
\[ C(\sigma) = \{ s^{(i)} | x(\sigma) \cong s^{(i)} , i = 1, 2, \ldots, 2m \}, \]
\[ \overline{C}(\sigma) = \{ s^{(i)} | i = 1, 2, \ldots, 2m \} \setminus C(\sigma). \]

Notice that the minimal solution \( s(\sigma) \) gets the following total profit:
\[ f(s(\sigma)) = \sum_{i=1,2,\ldots,2m} s^{(i)}w_i. \]
Since the associative property also holds for the compounding of more than two consistent semi-solutions (see again Eq. (9)), we would attempt to obtain more quickly another set of consistent basic semi-solutions from the current \( C(\sigma) \) by utilizing the following neighborhood structures:
\[ N_{N_{\text{MtF}}} = \{ \sigma_N \in S | \sigma_N = (\sigma[k], \sigma[1], \ldots, \sigma[k-1], \sigma[k+1], \ldots, \sigma[2m]), \sigma[k] \in \overline{C}(\sigma), \]
\[ 1 < k \leq 2m \}, \]
\[ N_{N_{\text{MtR}}} = \{ \sigma_N \in S | \sigma_N = (\sigma[1], \ldots, \sigma[k-1], \sigma[k+1], \ldots, \sigma[2m], \sigma[k]), \sigma[k] \in C(\sigma), \]
\[ 1 \leq k < 2m \}. \]
It is easy to see that the difference between neighborhood structures MtF and rMtF (resp., MtR and rMtR) is the condition \( \sigma[k] \in \overline{C}(\sigma) \) (resp., \( \sigma[k] \in C(\sigma) \)) for a basic semi-solution to be moved. However, the rMtR may tend to make the simulated annealing a more negative search since it may regardlessly move a more hopeful item out of the set \( C(\sigma) \).

Further, we consider a neighborhood structure such that the two kinds of movements of rMtF and rMtR are simultaneously applied to a compounding sequence, i.e.,
\[ N_{N_{\text{F&}}} = \{ \sigma_N \in S | \sigma_N = (\sigma[k], \sigma[1], \ldots, \sigma[k-1], \sigma[k+1], \ldots, \sigma[2m], \sigma[\ell]), \]
\[ \sigma[k] \in \overline{C}(\sigma), 1 < k \leq 2m, \sigma[\ell] \in C(\sigma), 1 \leq \ell (\neq k) < 2m \} \]
That is, we may expect a slightly more accelerated search of another set of consistent basic semi-solutions than the sole exploitation of each neighborhood structure of the rMtF and rMtR. For the neighborhood structure F&R, we see that \(|N_{N_{\text{F&}}}| = O(m) + O(m) = O(m)\).

In the following numerical section, each of these six neighborhood structures is embedded in Line 10 of Algorithm 1, and the heuristic behavior of the simulated annealing algorithm is empirically examined.

3.3. Settings of Annealing Parameters
In this paper, we basically follow a framework of the simulated annealing described in a textbook of algorithm design (see Skiena, 2008), and apply it to the weighted item collecting problem in directed bipartite graphs. As shown before in Algorithm 1, we use three temperature management parameters, \( t_{\text{max}}, t_{\text{min}}, \) and \( \alpha \). Generally, the initial temperature and the cooling rate are set to be \( t_{\text{max}} := 1 \) and \( 0.8 \leq \alpha \leq 0.99 \), respectively. In addition to the two typical parameters, we introduce the final temperature \( t_{\text{max}} \) for a forcible termination of the searching process. The temperature \( t \) is initially set to be \( t := t_{\text{max}} \) (see Line 7 in Algorithm 1), and after a constant number \( \ell_{\text{max}} \) of iterations of generating a neighboring solution from an incumbent solution, the \( t \) is lowered by \( t := \alpha \times t \) (see Line 30 in Algorithm 1). When the condition of \( t \leq t_{\text{min}} \) is met (that is, the sufficient cooling has finished), the searching process is terminated (see Line 8 in Algorithm 1). The number \( \ell_{\text{max}} \) of iterations of generating a neighboring solution from an incumbent solution is usually set to be \( 10^2 \leq \ell_{\text{max}} \leq 10^3 \).
The temperature $t$ determines the acceptance probability for an inferior solution to an incumbent solution, together with a normalization factor $k_B$. As described in Line 25 of Algorithm 1, we utilize the well-known definition of an ordinary acceptance probability, i.e., $\exp(-\Delta/(k_B \times t))$ for an inferior solution $x$ to the incumbent $x_{inc}$ with the deterioration $\Delta = f(x_{inc}) - f(x) > 0$. For a larger deterioration $\Delta$ at a temperature $t$, the acceptance probability takes a smaller value, and for a lower temperature $t$, it also takes a smaller value for an inferior solution with the same deterioration $\Delta$. For the normalization factor $k_B$, the mean value of the profits of the items may be used as a reference. For an enough budget instance with the unit profits (i.e., an instance of problem MSG), we set it to be $k_B := 1$, and we basically use the same value for a different type of instance in this paper.

Further, we maintain by the $u$ in the searching process the cumulative number of iterations with no improvement of the best solution $x'$. When the cumulative number $u$ of iterations without improvement of the best solution reaches a prescribed upper limit $u_{\text{max}}$, we regard that there is no hope for a more improvement in the searching process, and we terminate the computation (i.e., when $u = u_{\text{max}}$ holds in Line 8 of Algorithm 1). It may be hard to know an appropriate value of the upper limit $u_{\text{max}}$ in advance, and may be necessary to tune it for an instance of the weighted item collecting problem. In the following numerical section, we examine empirically a certain set of some values of the upper limit $u_{\text{max}}$ for a tuning of the simulated annealing, as well as the neighborhood structures.

4. Numerical Results

The program for numerical experiments is written in C language, and is run on a laptop computer with Windows 10 Pro (64bit), Intel Core i7 6500U CPU (2.50 GHz) and 8GB memory. Instances of the weighted item collecting problem to be tested are randomly generated. In order to know the maximum of the total profit of a reversing vector, i.e., $f^* = f(x')$, when we make a larger instance with an enough budget, we duplicate an instance with $m = n = 20$ as a composing unit. For such a size of problem instances, we can compute the maximum $f^*$ of the total profit of a reversing vector in a practical execution time on the laptop computer by a complete enumeration manner of Gray code representation (e.g., see Skiena, 2008). Such a manner of generating larger instances is motivated from the way taken in the benchmark construction of $m = n = 48$ with sixteen copies of a unit of $m = n = 3$ by Tang (2005).

We first generate an instance as a composing unit in the following settings:

- The number of items: $m = 20$, and the number of players: $n = 20$.
- Profits of each item $i \in I$: either the unit profits $w_i = w_{\text{uni}} = 1$, or uniformly random integers $w_{\text{uni}} \in \{1,2,\ldots,10\}$,
- Cost of each player $j \in J$: either the unit cost $c_j = 1$, or a uniformly random integer $c_j \in \{1,2,\ldots,10\}$,
- Budget: either an enough budget $b = \sum_{j \in J} c_j$, or a limited budget case of $b = \lceil \sum_{j \in J} c_j/4 \rceil$, i.e., $b/\sum_{j \in J} c_j = 0.25$.
- The number of signaling players of each item $i \in I$ (i.e., the degree of the item): $d_i = || \{j \in J \mid i \in B_j \cup R_j \} || \in \{2,3,4\}$ with probabilities $0.35, 0.35$ and $0.30$, respectively.
- The total number of arcs between the sets $I$ and $J$: $|A_{B}(0) \cup A_{f}(0)| = \sum_{i \in I} d_i$. The direction of each arc is chosen between the two alternatives with the same probability. It is checked whether each player is associated with at least one item.

By the same manner, we prepare one hundred instances with the size of $m = n = 20$ for every collection of the above settings. Then, from each composing unit, we construct three larger instances with $m = n = 20 \lambda$ for $\lambda \in \{2,3,4\}$, i.e., $m = n = 40, m = n = 60$ and $m = n = 80$, in which the profits of items and the costs of players are unchanged, while the budget is changed to be $b := \lambda b$. Of course, we are also interested in a random combination of some different composing units as a larger instance generator. However, in this paper we would like to examine the duplication as the first step of generating larger instances, and we would like to investigate another instance generator based on the random combination of composing units in future research.

The annealing parameters are set as follows:

- The initial temperature: $t_{\text{max}} = 1.0$, the final temperature: $t_{\text{min}} = 0.1$, and the cooling rate: $\alpha = 0.9$.
- The number of iterations at each temperature: $\ell_{\text{max}} = 1000$.
- The upper limit on the number of iterations with no improvement of the best solution: $u_{\text{max}} \in \{100,1000,10000\}$.
- The normalization factor in the acceptance probability: $k_B = 1.0$.  

[DOI: 10.1299/jamdsm.2018jamdsm0051] © 2018 The Japan Society of Mechanical Engineers
In all the following tables, each of the data indicates the mean value over one hundred test instances. For any instance to be tested in this section, it is satisfied that \( c_j \leq b \) for some player \( j \in J \), and we see that for an optimal solution \( x^* \), \( f(x^*) > 0 \) holds. Hence, we here define the relative error of the heuristic total profit \( f(x^*) \) obtained by the simulated annealing algorithm from the maximum \( f(x^*) \) to be
\[
E_R = \frac{f(x^*) - f(x^*)}{f(x^*)} \times 100 \%.
\]
Also we represent the number of instances for which the simulated annealing algorithm attains an optimal solution among one hundred test instances by \#OPT. The definition implies that the average relative error \( E_R \) over one hundred test instances is zero when \#OPT=100.

Table 1 shows the heuristic performance on test instances with \( m = n = 20 \) and \( b = \sum_{j \in J} c_j \). For the case of the unit profits and unit costs, the greedy heuristic algorithm returns a heuristic solution with 19.1 \%\) relative error from the optimal value, while the simulated annealing algorithm with each neighborhood structure, except for the MR with \( u_{\text{max}} \in \{100, 1000\} \), finds an optimal solution for each of one hundred test instances. The execution time of the simulated annealing is shorter than that of the enumeration in Gray code representation of a reversing vector. For the case of random profits and random costs, the average relative error of the simulated annealing is no more than 0.7 \%\) and the \#OPT is no less than 82, which also implies well improvement from the greedy heuristic solution.

Table 2 provides the results on test instances with \( m = n = 40 \) and \( b = \sum_{j \in J} c_j \). For the case of the unit profits and unit costs, the greedy heuristic algorithm also returns a heuristic solution with 19.1 \%\) relative error from the optimal value, while the simulated annealing algorithm employing each neighborhood structure with \( u_{\text{max}} \in 10000 \) attains an optimal solution for each of one hundred test instances. For the case of random profits and random costs, the average relative error of the simulated annealing is no more than 1.3 \%\), although any neighborhood structure indicates \#OPT\(<100 \). In this table, we also observe well improvement of the simulated annealing algorithm from the greedy heuristic solution.

Table 3 illustrates the heuristic performance on test instances with \( m = n = 60 \) and \( b = \sum_{j \in J} c_j \). In this table, we notice a clearer difference on heuristic performance among employed neighborhood structures in the simulated annealing algorithm. For the case of the unit profits and unit costs, when \( u_{\text{max}} = 100 \), only the rMtF and F&R indicate \#OPT\(>80 \). For the case of random profits and random costs, when \( u_{\text{max}} = 1000 \), only the F&R indicates \#OPT\(>80 \). In both cases, the SWAP obtains good values of \#OPT\(\geq87 \) when \( u_{\text{max}} = 10000 \), while it does \#OPT\(\leq48 \) when \( u_{\text{max}} = 100 \). Notice

Table 1 Heuristic performance on test instances with \( m = n = 20 \) and \( b = \sum_{j \in J} c_j \) \( (\text{\(t_{\text{max}} = 1.0, t_{\text{max}} = 0.1, \alpha = 0.9, u_{\text{max}} = 1000 \text{ and } k_{\text{R}} = 1.0\))}

| Neighborhood | \( u_{\text{max}} \) | \( f(\lambda) \) | \#OPT | Time \( \times 10^{-2} \) [sec] | \( f(\lambda) \) | \#OPT | Time \( \times 10^{-2} \) [sec] |
|--------------|----------------|--------|-------|-------------------------------|----------------|--------|-------------------------------|
| Greedy       | -              | 11.6   | 19.1  | 5                             | 0.1            | 81.9   | 10.9                          |
| SWAP         | 100            | 14.4   | 0     | 100                           | 1.1            | 91.6   | 0.5                          |
|              | 14.4           | 0.0    | 100   | 2.2                           | 91.8           | 0.3    | 93                           |
|              | 10000          | 14.4   | 0.0   | 100                           | 11.9           | 92.1   | 0.0                          |
| MtF          | 100            | 14.4   | 0.0   | 100                           | 1.2            | 91.7   | 0.5                          |
|              | 10000          | 14.4   | 0.0   | 100                           | 12.7           | 91.7   | 0.4                          |
| MtR          | 100            | 14.4   | 0.3   | 84                            | 1.1            | 91.5   | 0.7                          |
|              | 10000          | 14.4   | 0.1   | 99                            | 2.1            | 91.8   | 0.4                          |
| rMtF         | 100            | 14.4   | 0.0   | 100                           | 1.2            | 92.0   | 0.1                          |
|              | 10000          | 14.4   | 0.0   | 100                           | 13.0           | 92.0   | 0.1                          |
| rMtR         | 100            | 14.4   | 0.0   | 100                           | 1.2            | 91.5   | 0.7                          |
|              | 10000          | 14.4   | 0.0   | 100                           | 12.3           | 92.0   | 0.2                          |
| F&R          | 100            | 14.4   | 0.0   | 100                           | 1.3            | 92.1   | 0.1                          |
|              | 10000          | 14.4   | 0.0   | 100                           | 13.4           | 92.1   | 0.0                          |
| Gray         | -              | 14.4   | -     | -                            | 142.0          | 92.1   | -                            |
Table 2  Heuristic performance on test instances with \( m = n = 40 \) and \( b = \sum_{j \in I} c_j \) \( (t_{\text{max}} = 1.0, \ t_{\text{max}} = 0.1, \ \alpha = 0.9, \ t_{\text{max}} = 1000 \text{ and } \kappa_B = 1.0) \)

| Neighborhood | \( u_{\text{max}} \) | \( f(x) \) | \( E_{\text{R}} \) [%] | \#OPT | Time [sec] | \( f(x) \) | \( E_{\text{R}} \) [%] | \#OPT | Time [sec] |
|--------------|----------------|----------|----------------|-------|----------|----------------|----------|-------|----------|
| Greedy       | -              | 23.3     | 19.1           | 5     | \(<=0.1\) | 182.0          | 1.3      | 64    | 0.1      |
| SWAP         | 100            | 28.6     | 0.7            | 82    | 0.1      | 183.1          | 0.7      | 73    | 0.1      |
|              | 1000           | 28.8     | 0.3            | 92    | 0.1      | 184.1          | 0.1      | 94    | 0.5      |
| MtF          | 100            | 28.8     | 0.1            | 99    | 0.1      | 183.3          | 0.5      | 81    | 0.1      |
|              | 1000           | 28.8     | 0.0            | 100   | 0.1      | 183.4          | 0.5      | 83    | 0.1      |
| MtR          | 100            | 28.6     | 0.8            | 79    | 0.1      | 182.6          | 0.9      | 67    | 0.1      |
|              | 1000           | 28.7     | 0.4            | 90    | 0.1      | 183.0          | 0.7      | 74    | 0.1      |
| rMtF         | 100            | 28.8     | 0.0            | 100   | 0.5      | 183.6          | 0.4      | 84    | 0.5      |
| rMtR         | 100            | 28.7     | 0.6            | 85    | 0.1      | 182.2          | 1.1      | 59    | 0.1      |
|              | 1000           | 28.8     | 0.2            | 96    | 0.1      | 182.6          | 0.9      | 65    | 0.1      |
| F&R          | 100            | 28.8     | 0.1            | 99    | 0.1      | 183.7          | 0.4      | 81    | 0.1      |
|              | 1000           | 28.8     | 0.0            | 100   | 0.5      | 183.6          | 0.4      | 84    | 0.5      |
| Gray         | -              | 28.8     | -              | -     | -        | 184.3          | -        | -     | -        |

Table 3  Heuristic performance on test instances with \( m = n = 60 \) and \( b = \sum_{j \in I} c_j \) \( (t_{\text{max}} = 1.0, \ t_{\text{max}} = 0.1, \ \alpha = 0.9, \ t_{\text{max}} = 1000 \text{ and } \kappa_B = 1.0) \)

| Neighborhood | \( u_{\text{max}} \) | \( f(x) \) | \( E_{\text{R}} \) [%] | \#OPT | Time [sec] | \( f(x) \) | \( E_{\text{R}} \) [%] | \#OPT | Time [sec] |
|--------------|----------------|----------|----------------|-------|----------|----------------|----------|-------|----------|
| Greedy       | -              | 35.0     | 19.1           | 5     | \(<=0.1\) | 246.5          | 10.7     | 3     | \(<=0.1\) |
| SWAP         | 100            | 42.5     | 1.9            | 48    | 0.1      | 272.0          | 1.6      | 39    | 0.1      |
|              | 1000           | 42.8     | 1.1            | 67    | 0.2      | 274.6          | 0.7      | 62    | 0.2      |
| MtF          | 100            | 43.0     | 0.8            | 73    | 0.1      | 274.9          | 0.6      | 67    | 0.1      |
|              | 1000           | 43.2     | 0.3            | 91    | 0.2      | 275.1          | 0.5      | 70    | 0.2      |
| MtR          | 100            | 42.4     | 2.1            | 41    | 0.1      | 275.4          | 0.4      | 77    | 0.9      |
|              | 1000           | 42.7     | 1.3            | 59    | 0.2      | 276.4          | 0.4      | 79    | 0.9      |
| rMtF         | 100            | 42.3     | 0.1            | 97    | 1.0      | 271.8          | 1.7      | 40    | 0.1      |
| rMtR         | 100            | 43.1     | 0.4            | 83    | 0.1      | 275.3          | 0.4      | 72    | 0.1      |
|              | 1000           | 43.2     | 0.2            | 93    | 0.2      | 275.5          | 0.4      | 75    | 0.2      |
| F&R          | 100            | 42.6     | 1.5            | 53    | 0.1      | 273.1          | 1.2      | 45    | 0.1      |
|              | 1000           | 42.9     | 0.9            | 67    | 0.2      | 273.6          | 1.0      | 49    | 0.2      |
| Gray         | -              | 43.3     | 0.4            | 84    | 0.1      | 275.1          | 0.5      | 70    | 0.1      |

that a smaller value of the \( u_{\text{max}} \) tends toward a shorter execution time of the simulated annealing algorithm than a larger \( u_{\text{max}} \). The SWAP does not utilize a special feature of the weighted item collecting problem such as the commutative and associative properties for the compounding and Eq. (13). Hence, for a larger instance, it may be necessary for the SWAP to have a sufficient number of iterations (and much execution time) as in a general implementation of the simulated annealing. On the other hand, the solo exploitation, i.e., each of the MtF, MtR, rMtF and rMtR, is restricted for the search space in some sense, and it sometimes shows a smaller difference on the heuristic performance between \( u_{\text{max}} = 100 \) and \( u_{\text{max}} = 10000 \) than that of the SWAP. In this table, we see that the execution time of the simulated annealing algorithm is
still less than 1.1 seconds.

Table 4 shows the results on test instances with \( m = n = 80 \) and \( b = \sum_{j \neq i} c_{j} \). For the case of the unit profits and unit costs, any neighborhood structure with \( u_{\max} \leq 1000 \) unfortunately indicates \#OPT<80, although it with \( u_{\max} = 10000 \) attains \#OPT\geq95 within two seconds of the execution time. Also for the case of random profits and random costs, any neighborhood structure with \( u_{\max} \leq 1000 \) indicates \#OPT<80. Even the F&R with \( u_{\max} = 1000 \) does \#OPT<90. The relative error is still less than 1 [%] when \( u_{\max} = 10000 \), but the worse results may suggest a more careful tuning of the annealing parameters, e.g., a larger normalization factor \( \kappa \) and a quicker cooling \( \alpha \), or another effective neighborhood structure for solving a further large instance.

### Table 4: Heuristic performance on test instances with \( m = n = 80 \) and \( b = \sum_{j \neq i} c_{j} \) \((t_{\max} = 1.0, \ t_{\min} = 0.1, \ \alpha = 0.9, \ t_{\max} = 1000 \) and \( \kappa_{B} = 1.0)\)

| Neighborhood | \( u_{\max} \) | \( f(x) \) | \( E_{R} (%) \) | \#OPT | Time [sec] | \( f(x) \) | \( E_{R} (%) \) | \#OPT | Time [sec] |
|--------------|----------------|--------------|----------------|--------|------------|--------------|--------------|--------|------------|
| Greedy       | -              | 46.7         | 19.1           | 5      | \( <0.1 \) | 328.5        | 10.8         | 1      | \( <0.1 \) |
| SWAP         | 100            | 56.2         | 2.7            | 29     | 0.2        | 360.8        | 2.1          | 23     | 0.1        |
|              | 1000           | 57.0         | 1.3            | 55     | 0.4        | 364.9        | 1.1          | 45     | 0.4        |
| MtF          | 100            | 57.1         | 1.0            | 58     | 0.2        | 366.1        | 0.7          | 55     | 0.1        |
|              | 1000           | 57.4         | 0.6            | 71     | 0.3        | 366.5        | 0.6          | 61     | 0.3        |
| MtR          | 100            | 57.7         | 0.1            | 97     | 1.8        | 367.8        | 0.3          | 84     | 1.8        |
|              | 1000           | 57.7         | 0.1            | 100    | 1.6        | 367.5        | 0.5          | 67     | 1.5        |
| rMtF         | 100            | 57.3         | 0.8            | 64     | 0.2        | 367.0        | 0.4          | 55     | 0.2        |
|              | 1000           | 57.4         | 0.6            | 72     | 0.3        | 367.6        | 0.3          | 68     | 0.3        |
| rMtR         | 100            | 56.6         | 2.2            | 40     | 0.2        | 361.8        | 1.9          | 26     | 0.2        |
|              | 1000           | 56.9         | 1.4            | 55     | 0.3        | 363.2        | 1.5          | 32     | 0.3        |
| F&R          | 100            | 57.3         | 0.8            | 63     | 0.2        | 367.0        | 0.5          | 59     | 0.2        |
|              | 1000           | 57.5         | 0.5            | 75     | 0.3        | 367.8        | 0.2          | 76     | 0.3        |
| Gray         | -              | 57.7         | -              | -      | -          | 368.5        | -            | -      | -          |

### Table 5: Heuristic performance on test instances with \( m = n = 20 \) and \( b/\sum_{j \neq i} c_{j} = 0.25 \) \((t_{\max} = 1.0, \ t_{\min} = 0.1, \ \alpha = 0.9, \ t_{\max} = 1000 \) and \( \kappa_{B} = 1.0)\)

| Neighborhood | \( u_{\max} \) | \( f(x) \) | \( E_{R} (%) \) | \#OPT | Time [sec] | \( f(x) \) | \( E_{R} (%) \) | \#OPT | Time [sec] |
|--------------|----------------|--------------|----------------|--------|------------|--------------|--------------|--------|------------|
| Greedy       | -              | 8.6          | 33.0           | 2      | 0.1        | 63.4         | 19.8         | 8      | 0.1        |
| SWAP         | 100            | 12.9         | 0.5            | 94     | 1.2        | 78.4         | 0.9          | 80     | 1.1        |
|              | 1000           | 12.9         | 0.1            | 99     | 2.3        | 78.8         | 0.4          | 87     | 2.3        |
| MtF          | 100            | 12.9         | 0.0            | 100    | 12.3       | 79.0         | 0.2          | 95     | 12.6       |
|              | 1000           | 12.9         | 0.0            | 100    | 12.3       | 79.0         | 0.1          | 97     | 12.6       |
| MtR          | 100            | 12.8         | 1.4            | 85     | 1.2        | 77.9         | 1.6          | 69     | 1.2        |
|              | 1000           | 12.9         | 0.3            | 97     | 2.4        | 78.5         | 0.9          | 81     | 2.3        |
| rMtF         | 100            | 12.9         | 0.0            | 100    | 12.6       | 78.9         | 0.2          | 95     | 13.0       |
|              | 1000           | 12.9         | 0.0            | 100    | 13.8       | 79.0         | 0.1          | 99     | 13.0       |
| rMtR         | 100            | 12.8         | 0.8            | 90     | 1.4        | 77.4         | 2.1          | 66     | 1.5        |
|              | 1000           | 12.9         | 0.3            | 97     | 2.7        | 77.7         | 1.7          | 71     | 2.7        |
| F&R          | 100            | 12.9         | 0.0            | 100    | 2.6        | 79.0         | 0.1          | 99     | 2.6        |
|              | 1000           | 12.9         | 0.0            | 100    | 14.6       | 79.1         | 0.0          | 100    | 14.4       |
| Gray         | -              | 12.9         | -              | -      | -          | 79.1         | -            | -      | -          |

[DOI: 10.1299/jamdsm.2018jamdsm0051] © 2018 The Japan Society of Mechanical Engineers
Table 5 provides the results on test instances with \( m = n = 20 \) and \( b / \sum_{j \in I} c_j = 0.25 \), i.e., a case of the limited budget. The limited budget makes an instance closer to a variant of 0-1 knapsack problems in graphs (e.g., see Karuno et al., 2010; Pferschy and Schauer, 2009; Samphaiboon and Yamada, 2000; Wilbaut et al., 2008). The greedy heuristic algorithm used in this paper is the simplest one, and it does not involve a tie-breaking among basic semi-solutions with the same item weight by the minimal costs of Eq. (7) (see Karuno and Tanaka, 2017b). For the case of the unit profits and unit costs, the average relative error of the greedy heuristic algorithm is 33 [%], while that of the simulated annealing algorithm is at most 1.4 [%] (see the entry of the MtR with \( u_{\text{max}} = 100 \)). The simulated annealing algorithms with the MtF, rMtF and F&R attain \#OPT = 100 for any \( u_{\text{max}} \in \{100, 1000, 10000\} \). For the case of random profits and random costs, the average relative error of the simulated annealing algorithm is at most 2.1 [%]. The simulated annealing algorithms with the MtF, rMtF and F&R indicate \#OPT \( \geq 97 \) for any \( u_{\text{max}} \in \{100, 1000, 10000\} \). We remark that due to the limited budget, a larger instance by the duplication of a composing unit with factor \( \lambda \in \{2, 3, 4\} \) no longer has the optimal value of the maximum \( f^* \) for the composing unit multiplied by \( \lambda \), i.e., the optimal value is not always \( \lambda f^* \).

Through the five tables, except for three entries of the rMtF with \( u_{\text{max}} = 1000 \) and the unit profits in Table 2, with \( u_{\text{max}} = 100 \) and random profits in Table 3, and with \( u_{\text{max}} = 100 \) and the unit profits in Table 4, \#OPT values with the F&R are no less than the corresponding values with the other neighborhood structures.

5. Concluding Remarks

In this paper, we considered a weighted item collecting problem in directed bipartite graphs, and designed an iterative improvement heuristic based on the simulated annealing. A greedy heuristic algorithm recently presented by the authors obtained an optimal solution of a known test instance of the original integrated circuit design model of Tang (2005) in a short execution time, while for a randomly generated instance, there was some room for improvement of the heuristic quality. In this paper, utilizing a compounding sequence of basic semi-solutions, we defined some neighborhood structures for an incumbent solution, and embedded each of them in a framework of the simulated annealing. We also conducted numerical experiments mainly on enough budget test instances with up to eighty items and eighty players to examine the iterative improvement performance. The results showed that the simulated annealing algorithm improved well the heuristic quality from the greedy one with respect to the relative error, and it often attained an optimal solution for each of one hundred test instances with up to forty items and forty players. Within our numerical experiments, a multi-insertion type of neighborhood structure, named as the F&R, made the simulated annealing algorithm perform more effective search than the swapping, and some single insertion types of neighborhood structures.

For future research, it would be interesting to tune the annealing parameters more carefully. In addition, it would be necessary to develop another appropriate generator of larger size test instances. It also would be significant to investigate a more effective neighborhood structure of an incumbent solution for addressing an instance with a limited budget. Further, it would be interesting to examine an objective function with a penalty term of the reversing cost (see Karuno and Tanaka, 2017b). Such a change in the objective function may bring an instance with the optimal value of zero, and we should treat the heuristic performance by an appropriate manner (e.g., see Vazirani, 2001).

Appendix: A Greedy Heuristic Algorithm

In the simulated annealing algorithm, we utilize the greedy heuristic algorithm shown in Algorithm 2 to obtain the initial feasible solution. This is the simplest version presented in Karuno and Tanaka (2017a) with a modification which addresses two individual profits for each item. As in Algorithm 1, the \( 2m \) basic semi-solutions \( s^{(0)} \) and \( 3^{(0)} \) for \( m \) items \( i \in I \) have been obtained from the given directed bipartite graph by an \( O(mn) \) time preprocessing (see again Tang, 2005).

In Algorithm 2, the \( 2m \) basic semi-solutions are initially sorted in the non-increasing order of their profits, which requires \( O(m \log m) \) time. Since the time complexity of procedure COMPOUNDING\((x, s^\#: 3)\) is \( O(n) \), it takes \( O(mn) \) time to perform the for-loop of \( k = 1, 2, \ldots, 2m \) during Lines 4–6 in Algorithm 2. Hence, the greedy heuristic algorithm runs in \( O(m \log m + mn) \) time.

Acknowledgment

The authors are grateful to an anonymous referee for his/her careful reading. This research was partially supported by JSPS KAKENHI Grant JP16K01241.
Algorithm 2 GREEDY

Input: A set $I$ of $m$ items, a set $J$ of $n$ players, non-negative weights $w_i$ and $w_{m+i}$ such that $w_i + w_{m+i} > 0$ for each item $i \in I$, a positive cost $c_j$ for each player $j \in J$, a positive budget $b$, blue and red basic semi-solutions for each item $i \in I$, $s^{(i)}$ and $s^{(m+i)} (= \bar{s}^{(i)})$, respectively.

Output: A feasible heuristic solution $x' = (x'_1, x'_2, \ldots, x'_n)$.

1: $x' := 0; f' := f(0)$;
2: Initialize the compounding sequence $s = (s[1], s[2], \ldots, s[2m])$ of the $2m$ basic semi-solutions so that $w[s[1]] \geq w[s[2]] \geq \ldots \geq w[s[2m]]$;
3: $s := s'$;
4: for $k = 1$ to $2m$ do
5: $s' := \hat{s}[k]$; Call procedure COMPOUNDING($s, s', \hat{s}$); $s := \hat{s}$;
6: end for
7: Obtain the minimal solution $x$ for the semi-solution $s$;
8: if ($f(x) \geq f'$) then
9: $x' := x; f' := f(x')$;
10: end if
11: return $x'$.

References

Aarts, E. and Korst, J., Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing, (1989), Wiley.

Ibaraki, T., Algorithms and Data Structures (in Japanese), (1989), Shoko-do, Tokyo.

Karuno, Y., Nagamochi, H., and Wang, X., Optimization problems and algorithms in double-layered food packing systems, Journal of Advanced Mechanical Design, Systems, and Manufacturing, Vol. 4, No. 3, (2010), pp. 605–615.

Karuno, Y. and Tanaka, S., Cooperative item collecting problems in directed bipartite structures, Journal of Advanced Mechanical Design, Systems, and Manufacturing, Vol. 11, No. 2, (2017a), [DOI: 10.1299/jamdsm.2017jamdsm0025], 16 pages.

Karuno, Y. and Tanaka, S., Weighted item collecting in directed bipartite structure with a reversing cost penalty, Proceedings of the Fifth IIAE International Conference on Industrial Application Engineering, (2017b), pp. 62–68.

Pferschy, U. and Schauer, J., The knapsack problem with conflict graphs, Journal of Graph Algorithms and Applications, Vol. 13, No. 2, (2009), pp. 233–249.

Samphaiboon, N. and Yamada, T., Heuristic and exact algorithms for the precedence-constrained knapsack problem, Journal of Optimization Theory and Applications, Vol. 105, No. 3, (2000), pp. 659–676.

Skiena, S.S., The Algorithm Design Manual, (2008), Springer, London.

Tang, M., An adaptive genetic algorithm for the minimal switching graph problem, Lecture Notes in Computer Science, Vol. 3448, Raidl, G.R. and Gottlieb, J. (Eds.): EvoCOP 2005, Springer, (2005), pp. 224–233.

Tang, M., Eshraghian, K., and Cheung, H.N., An efficient approach to constrained via minimization for two-layer VLSI routing, Proceedings of IEEE Asia and South Pacific Design Automation Conference, (1999), pp. 149–152.

Vazirani, V.V., Approximation Algorithms, 2nd Ed., (2001), Springer, Berlin.

Wilbaut, C., Hanafi, S., and Salhi, S., A survey of effective heuristics and their application to a variety of knapsack problems, IMA Journal of Management Mathematics, Vol. 19, (2008), pp. 227–244.