IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 27 May 2024, accepted 7 June 2024, date of publication 12 June 2024, date of current version 20 June 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3413051

RESEARCH ARTICLE

Problem Definition and Optimization Method for Bipartite Graph Scheduling

HIROSHI IKEDA AND TATSUYA TAKANAGA
Fujitsu Ltd., Nakahara-ku, Kawasaki, Kanagawa 211-8588, Japan
Corresponding author: Hiroshi Ikeda (ikeike@fujitsu.com)

ABSTRACT Bipartite graphs can describe various systems in real world. In this study, we define a new problem class for optimizing the cost or profit associated with state changes in systems represented by bipartite graphs and propose a heuristic approach based on a fast greedy algorithm. We present the problem setting of the network circuit and device removal problem as a real-world problem and demonstrate the superiority of the proposed method for large-scale applications. This study contributes to the understanding and solving of various optimization problems in industrial fields, providing a comprehensive approach to cost (profit) optimization problems associated with state changes in systems represented by bipartite graphs.

INDEX TERMS Bipartite graph, combinatorial problem, scheduling optimization, sequence problem.

I. INTRODUCTION

In graph theory, a bipartite graph is a type of graph in which nodes can be divided into two nonadjacent groups. Each node in the graph is connected exclusively to nodes from the other group. Bipartite graphs are useful for representing system relationships in various real-world optimization problems. For instance, in facility location problems, the relationships between facility locations, such as hospitals, schools, and parks, and demand points, such as residential areas, commercial districts, and transportation centers, can be expressed using a bipartite graph [1], [2]. Specifically, facility-location candidates are defined as nodes in one group, while demand points are nodes in the other group. When the demand of a demand point is satisfied by placing a facility at a certain candidate location, a link is established between the candidate location and the demand point node [3], [4], [5]. Therefore, a bipartite graph represents the relationship between the placement candidate positions and demand points.

Furthermore, the relationship between the circuits and devices in network systems can also be represented using a bipartite graph. A bipartite graph can be created considering circuits and devices as nodes and connecting nodes when a circuit uses a device. Consequently, the problem of removing outdated network circuits and devices during the transition to a new network can be expressed as a system optimization problem within the bipartite graph. The network circuit and device removal problem is an optimization problem in which the circuits are sequentially removed to minimize running costs of during all circuits and devices have been eliminated. This problem can be interpreted as an optimization problem involving the cost (or profit) associated with the state change from the operating state to the removed state in a system with a bipartite graph representing the relationship between circuits and devices. This problem is also a part of the optical network modernization issue [6].

Other cost (profit) optimization problems associated with state change in systems represented by bipartite graphs include the network circuit and device removal problem. For example, there is a problem of sequentially arranging facilities at the facility placement positions obtained as a solution to the facility-location problem, which has been described earlier [7]. The time at which the demand at each demand point begins to meet varies depending on the facility arrangement order. Although the time from the placement of the first facility until all facilities are located is consistent, the total amount of the product of satisfied demand and time varies depending on the order of facility placement. As the total amount of the product satisfying demand and time correlates with profit, the problem of determining the arrangement order of facilities is also an example of

The associate editor coordinating the review of this manuscript and approving it for publication was Zhiwu Li.

optimizing cost (profit) associated with the state change in systems represented by bipartite graphs.

The suspension of production equipment owing to factory shutdown and periodic maintenance and the operation of production equipment upon business resumption also constitute optimization problems [8], [9]. This is because a bipartite graph can represent the relationship between products and manufacturing equipment and the use of manufacturing equipment in the production of products. When a group of manufacturing equipment shuts down, and if the necessary equipment for manufacturing the product is halted, the product cannot be manufactured. Consequently, when manufacturing equipment is stopped one by one, the time when each product cannot be manufactured varies depending on the order of stopping, and the total amount of the product's profit and time differs. Additionally, when the manufacturing equipment is operated one by one upon resumption of operations, the time when the respective products can be manufactured differs depending on the operation order, and the total amount of the product's profit and time also varies. Therefore, these optimization problems can be considered as cost (profit) optimization problems associated with state changes in systems represented by bipartite graphs. Numerous optimization problems exist in industrial fields [10].

In this study, we propose a novel problem class called Bipartite Graph Scheduling (BGS), which is an optimization problem concerning cost (profit) resulting from state changes in systems represented by a bipartite graph. In Section II, we present the definition of BGS and propose heuristics based on a greedy algorithm to effectively solve BGS. Furthermore, Section III describes the specific problem settings that can be addressed using the proposed method, and Section IV presents the results obtained. Finally, the conclusion is provided in Section V.

II. METHOD

A. DEFINITION OF BGS

This section defines the system described by BGS. The system includes two kinds of objects; one whose order of state change can be directly controlled, and another whose state change indirectly depends on the state change of other objects. Each object in the system corresponds to the node of the graph, and the dependency relation of the state change between the objects corresponds to an edge in the graph. Consequently, the graph is a bipartite graph as edges exist only between objects that can be directly controlled and objects that change state indirectly. Hereafter, the set of objects that can be directly controlled is referred to as A-group, and the set of objects whose state changes only indirectly is referred as B-group. Each object included in the A-group is called an A object, and each object included in the B-group is called a B object. In addition, the state change of each object is referred to as "firing," analogous to transitions. In this system, all objects fire only once during the period from the initial state to the final state. In other words, in the initial state, all objects have not fired, and in the final state, all objects have fired. This period is called a state change period, and since only one A object fires per unit time, the length of the state change period is determined by multiplying the unit time by the total number of A objects.

In this system, each object has a cost or profit per unit time, which changes only by firing the object itself. The cost or profit of each unit time of each object is aggregated to define the total cost or profit of the system for each unit time during the state change period. B object always depends on one or more A objects, and this dependency can be either an AND or OR relationship. In the case of an AND relationship, B objects exhibit the property that they fire only when all A objects with dependencies fire. Furthermore, in the case of OR relationship, the B object fires when any of the dependent A objects fires.

In summary, the following is the definition of the system described by BGS:

- The system consists of two kinds of objects, A and B objects, and their firing dependencies.
- If an object is represented by a graph node and a dependency is represented by a graph edge, the result is a bipartite graph.
- An object fires only once during a state change period.
- An object changes its cost or benefit per unit time due to firing.
- Only one of the A objects fires per unit time.
- The cost or profit of a system is the sum of the costs or profits of each object at each unit time.
- Dependencies are either AND or OR relationships.

BGS is defined as an optimization problem class that minimizes the objective function in the case of cost and maximizes the objective function in the case of profit. In a BGS system, the firing of a B object depends on the firing of an A object. Therefore, when the firing order of the A object is determined, the firing time of the B object is uniquely determined. Consequently, the cost or profit of the system is uniquely determined. Therefore, the control variable of the BGS is the firing order of the A object, which we decide to call the schedule. Considering each object and its firing in the BGS as a job and job execution, the BGS can be seen as a variant of the single-machine scheduling problem. Appendix describes the relationship with the single-machine scheduling problem.

The system handled by the BGS and its state change are described in Fig. 1 referring to the network circuit and device removal problem. In the network circuit and device removal problem, a circuit corresponds to an A object and a device corresponds to a B object. First, all circuits and devices are operational, and at the end of the state change period, all circuits and devices are turned off. A circuit or device outage corresponds to an object firing. Fig. 1(a) shows a bipartite graph representing the system of the network circuit and device removal problem. On the left side of Fig. 1(a), there are A objects, constituting the A-group,

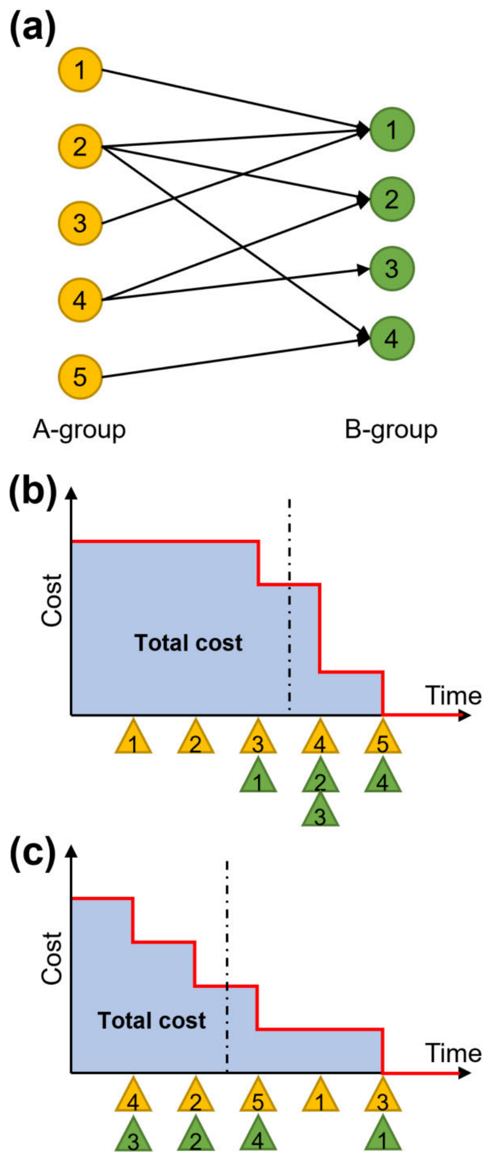

FIGURE 1. **Schematic diagram of BGS.**

that can directly control firing, and on the right side, there are B objects (B-group) whose fire is indirectly determined. The edges between objects A and B represent dependencies. All dependencies in Fig. 1 are AND relationships. Each object fires only once during a state change period, changing its cost per unit time.

In the system shown in Fig. 1(a), the B object incurs a cost of 1 per unit time before firing and 0 after firing. Conversely, the A object has a cost of 0 per unit time both before and after firing. Fig. 1(b) is a graph showing the cost per unit time when the A object is fired in the order of 1, 2, 3, 4, and 5. The state change period begins at time $t = 0$ and the cost per unit time of the system at time t is $s(t)$. If $t = 0$, then $s(0) = 4$ because all B objects remain unfired. Since all dependencies are AND relationships, the first time a B object fires is when A object 3 has been fired, occurring at $t = 3$, the B object 1 fires. Therefore, at $t < 3$, $s(t) = 4$, and $s(3) = 3$. Next, the A object 4 is fired at $t = 4$, the B object 2 and 3 are simultaneously fired, resulting in $s(4) = 1$. At $t = 5$, all A and B objects have been fired, and $s(5) = 0$. The cost of the system can be obtained by integrating $s(t)$ shown in Fig. 1(b) over the state change period. When the schedule is 1, 2, 3, 4, and 5, the cost becomes 16. The goal of the BGS is to find the schedule that minimizes the cost of the system. The optimal schedule for the system in Fig. 1(a) is 4, 2, 5, 1, and 3, as shown in Fig. 1(c), and the minimum cost is 11.

Here, we consider the relationship between the optimal schedules of the two systems in which only the costs of each object before and after firing differ by a certain amount. In system #1, the cost before firing A object a_i is $C(a_i)$ and the cost after firing is 0; the cost before firing B object b_j is $D(b_j)$, and the cost after firing is 0. However, in system #2 the cost before firing A object a_i is $C(a_i) + U(a_i)$, and the cost after firing is $U(a_i)$; the cost before firing B object b_j is $D(b_j)+V(b_j)$, and the cost after firing B object b_j is $V(b_j)$. The cost of system #2 is continuously required as an offset value even after firing. Let $\boldsymbol{x}$ be the schedule. Since both systems are the same except for the cost before and after the firing, the firing time of the corresponding object is the same for both systems if $\boldsymbol{x}$ is the same. The state change period begins at time t$t = 0$. If the number of A objects in the system is $|A|$, the end of the state change period is $t = |A|$. Let $t_{a_i}(\boldsymbol{x})$ be the firing time of A object ai at $\boldsymbol{x}$. The firing time of the B object b_j is defined as $t_{b_j}(\boldsymbol{x})$. If the cost of system #1 is $S_1(\boldsymbol{x})$ and the cost of system #2 is $S_2(\boldsymbol{x})$, $S_1(\boldsymbol{x})$ and $S_2(\boldsymbol{x})$ are expressed by the following equations respectively,

$$S_1(\boldsymbol{x}) = \sum_i t_{a_i}(\boldsymbol{x})C(a_i) + \sum_j t_{b_j}(\boldsymbol{x})D(b_j), \tag{1}$$

$$S_2(\boldsymbol{x}) = \sum_i t_{a_i}(\boldsymbol{x})C(a_i) + |A| \sum_i U(a_i) + \sum_j t_{b_j}(\boldsymbol{x})D(b_j) + |A| \sum_j V(b_j). \tag{2}$$

Therefore, the relationship between the two systems is given by:

$$S_2(\boldsymbol{x}) = S_1(\boldsymbol{x}) + |A| \sum_i U(a_i) + |A| \sum_j V(b_j), \tag{3}$$

The second and third terms on the right side of (3) are constants independent of $\boldsymbol{x}$; hence, $S_2(\boldsymbol{x})$ is minimized at $\boldsymbol{x}$ where $S_1(\boldsymbol{x})$ is minimized, and $S_1(\boldsymbol{x})$ is minimized at $\boldsymbol{x}$ where $S_2(\boldsymbol{x})$ is minimized. Therefore, determining $\boldsymbol{x}$ that minimizes the cost of system #1 also results in finding the $\boldsymbol{x}$ that minimizes the cost of system #2. In other words, in the optimization of systems like system #2, where the cost after firing an object is not necessarily zero, we can transform the system into one similar to system #1, where the cost after firing an object is zero, and thereafter find the optimal schedule.

Next, consider the optimal BGS schedule that maximizes the system profit. Suppose that there is a system #3, which is represented by the same bipartite graph as system #1. In system #3, the profit before firing A object a_i is $-C(a_i) - U(a_i)$, the profit after firing is $U(a_i)$, the profit before firing the B object b_j is $-D(b_j) - V(b_j)$, and the profit after firing is $-V(b_j)$. If $S_3(\boldsymbol{x})$ is the profit of system #3 in $\boldsymbol{x}$, then the following equation is obtained,

$$S_3(\boldsymbol{x}) = -\sum_i t_{a_i}(\boldsymbol{x})C(a_i) - |A|\sum_i U(a_i) - \sum_j t_{b_j}(\boldsymbol{x})D(b_j) - |A|\sum_j V(b_j). \tag{4}$$

Rewriting the above equation using (1),

$$S_3(\boldsymbol{x}) = -S_1(\boldsymbol{x}) - |A|\sum_i U(a_i) - |A|\sum_j V(b_j), \tag{5}$$

As the second and third terms of the right side of (5) are constants and independent of $\boldsymbol{x}$, $S_3(\boldsymbol{x})$ becomes maximum at $\boldsymbol{x}$ where $S_1(\boldsymbol{x})$ becomes minimum, and $S_1(\boldsymbol{x})$ becomes minimum at $\boldsymbol{x}$ where $S_3(\boldsymbol{x})$ becomes maximum. Therefore, if $\boldsymbol{x}$ is determined to minimize the cost of the system #1, $\boldsymbol{x}$ is determined to maximize the profit of the system #3. Specifically, while the optimization of systems like system #3, focuses on the profit of objects rather than their cost, we can transform the system into one like system #1, which focuses on the cost of objects, and then find the optimal schedule.

From the relationship among the three systems represented by (3) and (5), the following description is limited to a system such as system #1 in which the cost after firing of an object is zero and the cost is minimized, unless otherwise stated. Table 1 summarizes the variables used in the following sections. In Table 1, $G_A(O_A) = \{a|a \in O_A\}$ and $G_B(O_B) = \{b|b \in O_B\}$.

TABLE 1. Variable symbols and explanation.

Variable symbols	Explanation
$a \in A$	Object a in set A
A	Set of objects in A-group
$b_i \in B$	Object b in set B
B	Set of objects in B-group
$t(i)$	Firing time of object i
$C(a)$	Cost per unit time of object a in set A before firing
$D(b)$	Cost per unit time of object b in set B before firing
A_b	Set of objects in A that object b depends on
B_a	Set of objects in B that are dependent on object a in A
$\|A\|$	Number of objects in set A
$\|B\|$	Number of objects in set B
O_A	Firing order of objects in set A
O_B	Firing order of objects in set B
$G_A(O_A)$	Set of objects in O_A
$G_B(O_B)$	Set of objects in O_B
Z	Object selection indicator
Q	System cost

If the dependency is an AND relationship, the firing time is given by

$$t(b) = \max_{a \in A_b} t(a). \tag{6}$$

Here, $t(b)$ represents the firing time of the object b. Therefore, in the case of an AND relationship, the firing time of the B object coincides with the maximum value of the firing time of the A object in the dependency relationship. However, if the dependency is an OR relationship, the firing time can be expressed by the following equation,

$$t(b) = \min_{a \in A_b} t(a). \tag{7}$$

In the OR relationship, the firing time of the B object matches the minimum firing time of the dependent A object.

B. HEURISTIC FOR BGS

The solution method for the BGS developed in this study is the heuristics based on the greedy algorithm. The development method uses different algorithms owing to the dependencies between objects in the system. While BGS covers systems with mixed AND and OR dependencies between objects when viewed as a whole system, the development method only deals with AND relationships or OR relationships. Section II-B1 describes algorithms for AND relationships and Section II-B2 describes algorithms for OR relationships.

1) ALGORITHM OF AND RELATIONSHIP

The basic idea behind the development method is to sequentially select the objects of B-group that have a significant decrease in the system cost per unit time when fired and can be fired by a small number of firings of A-group. The algorithm that concretely implements this idea is shown in Algorithm 1. The theoretical time complexity of Algorithm 1 is $O(|B|^2|A|)$.

In step 14 of Algorithm 1, the function calcIndexOrderA, calculates Z and $addO_A$ for the B-group object b that has not yet fired. Here, $addO_A$ is a permutation of objects in A-group that need to be fired to fire b. Z is an index that takes a large value for objects in B-group that can be fired with a significant decrease in the cost of the system per unit time and a small number of firings from A-group. Algorithm 2 shows the method for calculating Z. In this function, the following equation is specifically calculated:

$$T(b) = \left|A_b \cap \overline{G_A}\right|, \tag{8}$$

$$K(b) = D(b) + \sum_{a \in (A_b \cap \overline{G_A})} C(a), \tag{9}$$

$$Z(b) = \begin{cases} BigZ, & \text{if } T(b) = 0, \\ \dfrac{K(b)}{T(b)}, & \text{otherwise.} \end{cases} \tag{10}$$

Here, the $BigZ$ is set to a value sufficiently larger than any $K(b)/T(b)$ when $T(b) \neq 0$. Moreover, G_A is a set of fired A-group objects. $\bar{G}$ is a complement of the set G. || indicates

Algorithm 1 BGS Heuristics for AND Relations

```
Input: $S_A, S_B, E$
Output: $O_A$
1: Init empty dict $dictO_A$ (key $G_B$, value $O_A$)
2: Init empty dict $dictQ$ (key $G_B$, value $Q$)
3: Init empty stacks $S_1, S_2$
4: $O_B \Leftarrow \varnothing$, $dictO_A[G_B(O_B)] \Leftarrow \varnothing$, $dictQ[G_B(O_B)] \Leftarrow 0$.
5: $S_1$.push($O_B$)
6: for $i = 1$ to $|B|$ do
7:     Set $maxZ$ as small value.
8:     while $S_1 \neq \varnothing$ do
9:         $O_B \Leftarrow S_1$.pop()
10:        for each $b$ in $B$ do
11:            if $b$ in $O_B$ then
                   continue
12:            end if
13:            // Calculate $Z$ and order of A objects
14:            $Z, addO_A \Leftarrow$ calcIndexOrderA($O_B, b$)
15:            if $Z < maxZ$ then
                   continue
16:            end if
17:            if $Z > maxZ$ then
18:                $S_2$.clear()
19:                $maxZ \Leftarrow Z$
20:            end if
21:            $curO_B \Leftarrow O_B$
22:            // Calculate $Q$ from $O_B, b, addO_A$
23:            $Q \Leftarrow$ calcCostOfAND($O_B, b, addO_A$)
24:            $curO_B$.append($b$)
25:            $curG_B \Leftarrow G_B(curO_B)$
26:            if $curG_B$ in $dictQ$ then
27:                if $Q \geq dictQ[curG_B]$ then
                       continue
28:                end if
29:                // Remove elements from $S_2$ based on $curO_B$
30:                removeOrderBsFromStack($S_2, curO_B$)
31:            end if
32:            $dictQ[curG_B] \Leftarrow Q$
33:            // Concatenate $addO_A$ to $dictO_A[G_B(O_B)]$
34:            $curO_A \Leftarrow dictO_A[G_B(O_B)], addO_A$
35:            $dictO_A[curG_B] \Leftarrow curO_A$
36:            $S_2$.push($curO_B$)
37:        end for
38:    end while
39:    if $S_2$.size() $> STACK\_LIMIT$ then
40:        reduceStackRandomly($S_2$)
41:    end if
42:    // Swap stacks
43:    $S_1 \Leftarrow S_2$; $S_2$.clear()
44: end for
45: $O_B \Leftarrow S_1$.pop()
46: $O_A \Leftarrow dictO_A[G_B(O_B)]$
47: return $O_A$
```

Algorithm 2 calcIndexOrderA:

```
Require: $O_B, b$
Ensure: Index $Z$, Order of A objects to fire in addition $addO_A$
1: Init $addO_A \Leftarrow \varnothing$, $K \leftarrow D(b)$
2: Init set of A objects $F_A \Leftarrow \varnothing$
3: Init set of A object costs $V_A \Leftarrow \varnothing$
4: for each $a$ in $A_b$ do
5:     if $a$ in $dictO_A[G_B(O_B)]$ then
           continue
6:     end if
7:     $F_A$.append($a$)
8:     $F_A$.append($C(a)$)
9:     $K \Leftarrow K + C(a)$
10: end for
11: $T \Leftarrow$ len($F_A$)
12: if $T = 0$ then
13:     $Z \Leftarrow BigZ$
14:     return $Z, addO_A$
15: end if
16: $Z \Leftarrow K/T$
17: // Create list to store pairs of $F_A$ and $V_A$
18: $pairs = []$
19: for $i = 1$ to $T$ do
20:     $pairs$.append(($F_A[i], V_A[i]$))
21: end for
22: Sort pairs in descending order based on $V_A[i]$
23: // Update $addO_A$ with sorted pairs
24: for $i = 1$ to $T$ do
25:     $addO_A$.append($pairs[i-1][0]$)
26: end for
27: return $Z, addO_A$
```

the number of elements in the set. Namely, $T(b)$ indicates the number of unfired objects among the objects of the A-group on which the object b depends on. $K(b)$ represents a decrease in the cost of the system per unit time when object b is fired.

In step 34 of Algorithm 1, the object with the maximum Z value from the B-group is added to the end of the firing order. In step 36, it is stored in the stack S_2. When there are multiple firing orders with maximum Z values, all of them are stored in S_2. This is essential for selecting the firing order based on the indicator Z in the next firing. However, if the number of firing orders in S_2 exceeds a predetermined number, the function reduceStackRandomly in step 40 of Algorithm 1 is used to randomly remove the stored firing orders in S_2, thereby reducing it to the predetermined number. In the function calcCostOfAND in step 23 of Algorithm 1, the cost of firing b is calculated. The algorithm for this cost calculation is shown in Algorithm 3. In a function removeOrderBsFromStack in step 30 of Algorithm 1, if there is a firing order in S_2 with the same elements as the firing

Algorithm 3 calcCostOfAND

```
Require: $O_B$, $b$, Order of A objects to fire in addition $addO_A$
Ensure: $Q$
 1: Init $Q \Leftarrow dictQ[G_B(O_B)]$
 2: $T \Leftarrow$ len($addO_A$)
 3: for $i = 1$ to $T$ do
 4:     $a \Leftarrow addO_A[i]$
 5:     $Q \Leftarrow Q + i \times C(a)$
 6: end for
 7: for each $b$ in $B$ do
 8:     if $b$ in $O_B$ then
            continue
 9:     end if
10:     $Q \Leftarrow Q + T \times D(b)$
11: end for
12: for each $a$ in $A$ do
13:     if $a$ in $dictO_A[G_B(O_B)]$ then
            continue
14:     end if
15:     if $a$ in $addO_A$ then
            continue
16:     end if
17:     $Q \Leftarrow Q + T \times C(a)$
18: end for
19: return $Q$
```

order $curO_B$ with the maximum Z value, and its cost is greater than the cost Q of $curO_B$, that firing order is removed from S_2. All objects are fired when the block from step 6 to step 44 of Algorithm 1 is completed. As the firing order of all the objects in the B-group is stored in stack S_1, the firing order of the A-group corresponding to it is taken out in step 46 resulting in the output of the algorithm.

2) ALGORITHM OF OR RELATIONSHIP

The basic idea for handling dependency of OR relationship is to sequentially select the objects in the A-group that, when fired, will result in the largest decrease in the system cost per unit time. The algorithm that implements this idea concretely is shown in Algorithm 4. The theoretical time complexity of Algorithm 4 is $O(|A|^2|B|)$.

In step 14 of Algorithm 4, the function calcIndexAddB computes Z and $addB$ for the unfired object in A-group. Z is an indicator that takes a larger value when the decrease in the cost of the system per unit time upon firing is greater. Algorithm 5 shows the method for calculating Z. In this function, the following equation is specifically calculated:

$$Z(a) = C(a) + \sum_{b \in (B_a \cap \overline{G_B})} D(b). \tag{11}$$

Here, G_B is a set of objects in B-group that have already been fired. Also, the $addB$ is a set of objects in B-group that will be fired when object a is fired.

In step 24 of Algorithm 4, the object with the maximum Z value from A-group is added to the end of the schedule, and in step 36, it is stored in stack S_2. If there are multiple objects with the maximum Z value, all of them are stored in S_2. Similar to the case with the AND relationship,

Algorithm 4 BGS Heuristics for OR Relations

```
Require: $S_A$, $S_B$, E
Ensure: $O_A$
 1: Init empty dict $dictO_B$ (key $G_A$, value $O_B$)
 2: Init empty dict $dictQ$ (key $G_A$, value $Q$)
 3: Init empty stacks $S_1$, $S_2$
 4: $O_A \Leftarrow \varnothing$, $dictO_B[G_A(O_A)] \Leftarrow \varnothing$, $dictQ[G_A(O_A)] \Leftarrow 0$
 5: $S_1$.push($O_A$)
 6: for $i = 1$ to $|A|$ do
 7:     Set $maxZ$ as small value
 8:     while $S_1 \neq \varnothing$ do
 9:         $O_A \Leftarrow S_1$.pop()
10:         // Calculate $Q$ from $O_A$
11:         $Q \Leftarrow$ calcCostOfOR($O_A$)
12:         for each $a$ in $A$ do
13:             if $a$ in $O_A$ then
                    continue.
14:             end if
15:             // Calculate $Z$ and set of B objects
16:             $Z$, $addB \Leftarrow$ calcIndexAddB($O_A$, $a$)
17:             if $Z < maxZ$ then
                    continue
18:             end if
19:             if $Z > maxZ$ then
20:                 $S_2$.clear()
21:                 $maxZ \Leftarrow Z$
22:             end if
23:             $curO_A \Leftarrow O_A$
24:             $curO_A$.append($a$)
25:             $curG_A \Leftarrow G_A(curO_A)$
26:             if $curG_A$ in $dictQ$ then
27:                 if $Q \geq dictQ[curG_A]$ then
                        continue
28:                 end if
29:                 // Remove elements from $S_2$ based on $curO_A$
30:                 removeOrderAsFromStack($S_2$, $curO_A$)
31:             end if
32:             $dictQ[curG_A] \Leftarrow Q$
33:             // Concatenate $addB$ to $dictO_B[G_A(O_A)]$
34:             $curO_B \Leftarrow dictO_B[G_A(O_A)]$, $addB$
35:             $dictO_B[curG_A] \Leftarrow curO_B$
36:             $S_2$.push($curO_A$)
37:         end for
38:     end while
39:     if $S_2$.size() > $STACK\_LIMIT$ then
40:         reduceStackRandomly($S_2$)
41:     end if
42:     // Swap stacks
43:     $S_1 \Leftarrow S_2$; $S_2$.clear()
44: end for
45: $O_A \Leftarrow S_1$.pop()
46: return $O_A$
```

this ensures that the schedule is selected based on indicator Z in the next firing. If the number of schedules in S_2 is too large, it will take a long time to process. Therefore, if the number of firing orders in S_2 exceeds a predetermined number, the function reduceStackRandomly in Step 40 of Algorithm 4 randomly removes the stored firing orders from S_2, reducing it to the predetermined number. This is similar to the case of the AND relationship. In step 11 of Algorithm 4, the function calcCostOfOR is responsible for calculating the cost of firing the next object a. As the cost of firing of the next object a and the accompanying firing of object b are included, the cost calculation looks like Algorithm 6. When the for block from step 6 to step 44 of Algorithm 4 is completed, all objects are fired. The firing order of all the objects in the A-group is stored in stack S_1 and the schedule is retrieved in step 45, which is considered as the output of the algorithm.

Algorithm 5 calcIndexAddB

```
Require: $O_A$, $a$
Ensure: Index $Z$, Set of B objects to fire in addition $addB$
1:  Init $addB \Leftarrow \emptyset, Z \leftarrow C(a)$
2:  for each $b$ in $B_a$ do
3:      if $b$ in $dictO_B[G_A(O_A)]$ then
            continue
4:      end if
5:      $addB$.append($b$)
6:      $Z \Leftarrow Z + D(b)$
7:  end for
8:  return $Z, addB$
```

Algorithm 6 calcCostOfOR

```
Require: $O_A$
Ensure: $Q$
1:  $Q \Leftarrow dictQ[G_A(O_A)]$
2:  for each $b$ in $B$ do
3:      if $b$ in $dictO_B[G_A(O_A)]$ then
            continue
4:      end if
5:      $Q \Leftarrow Q + D(b)$
6:  end for
7:  for each $a$ in A do
8:      if $a$ in $O_A$ then
            continue
9:      end if
10:     $Q \Leftarrow Q + C(a)$
11: end for
12: return $Q$
```

III. PROBLEM SETTING

In this study, we use the network circuit and device removal problem as a specific instance of the BGS problem class, and compare the approaches from previous research with the proposed heuristics. The network circuit and device removal problem is described in [6]. This problem minimizes the running cost of devices when removing aged devices in a large-scale network. Specifically, to remove aged devices, all circuits using that device need to be migrated to a new network. During the circuit migration process, there is a running cost for maintaining the device. Once the circuit migration is completed, the device can be removed, which reduces the running cost to zero. The objective is to control the schedule of circuit migration to minimize the total running cost until all devices are removed.

We used the same instances of the network line removal problem as in [6]. The heuristics described in Section II-A are applied to these instances. In the network circuit and device removal problem, the circuit is an object of the A-group and the device is an object of the B-group, and the dependency relationships are AND relationships. Additionally, the costs before firing of all objects in the A-group are set to 0, while the costs before firing of each object in the B-group are set to 1. Since the costs before firing of each object in BGS can be different, the network circuit and device removal problem can be considered as a BGS with conditions on the costs before firing.

Sugimura et al. [6] previously developed and evaluated an integer programing and higher order binary optimization formulation, both solvable by Gurobi, one of the commercial MIP solvers. They also developed a binary quadratic problem formulation using a digital annealer (DA) [11]. As a result, they achieved up to 35% improvement in solution quality using the DA. Additionally, they achieved more than 20 times faster execution speed. DA ran for 1800 s. The evaluation of the execution speed is based on the time it takes for Gurobi's solution method and DA to obtain equivalent solutions.

Furthermore, owing to the formulation difference between [6] and our study, the following relationship arises between the costs of the two methods, even for the same circuit migration order:

$$S(O_A) = S'(O_A) + |B|. \tag{12}$$

Here, O_A is an arbitrary permutation of objects in the A-group, which is called a schedule in BGS. $S'(O_A)$ is the cost in [6] and $S(O_A)$ is the cost in this study. $|B|$ is the number of objects in the A-group, i.e., the number of devices. For the evaluation in Section IV, we first converted the costs of [6] to the costs used in our study using the (12), and then compared the converted costs.

IV. RESULTS AND DISCUSSION

Table 2 shows the problem instances used in our study along with their optimization results. Three data were obtained from SNDLib datasets [12] and the number of circuits limited to those shown in Table 2 were used as problem instances. The present method (PM) in our study was implemented using an Intel Xeon E-2276G CPU@3.80 GHz with six physical cores. The execution time for all problem instances in Table 2 is less than 3 s, which can be considered very fast compared

TABLE 2. Results of network circuit and device removal problem by several methods.

	Circuit	Edge	PM	DA	LS	LB
	80	313	1011	1011	1011	933
France	100	379	1311	1304	1303	1175
($\|B\| = 25$)	200	716	3022	3020	3012	2438
	290	1054	5020	5012	5028	3642
	80	305	1449	1449	1449	1382
India	100	386	1913	1908	1908	1787
($\|B\| = 35$)	200	779	3946	3957	3943	3519
	290	1121	6010	6078	6012	5096
	595	2346	14284	–	14307	–
	80	361	1672	1675	1672	1646
Pióro	100	458	2112	2097	2095	2004
($\|B\| = 40$)	200	884	4510	4550	4483	4022
	290	1276	7023	7068	6913	5825
	780	3365	21494	–	21682	–

to the execution time of [6]. Table 2 includes the cost of DA and PM.

The "Edge" column in Table 2 shows the number of dependency relationships, and the "DA" column shows the cost of the DA in [6]. Sugimura et al. [6] did not handle large problem instances with a high number of circuits for India and Pióro datasets, so these instances are indicated with a hyphen in Table 2. The "PM" column represents the cost of the method proposed in this paper. In addition to the results of [6], we also show the optimization results for the same problem instances using LocalSolver (LS) [13] with a runtime of 30 s each in the "LS" column. LS is a mathematical optimization solver that excels in solving a wide range of scheduling problems, including single-machine scheduling problems. As the network circuit and device removal problem considered here represent the removal order of A objects as a permutation, it can be solved using LS. The "LB" column represents a lower bound value obtained through Integer Programming (IP) as formulated by Sugimura et al. [6] and solved using Gurobi. This lower bound guarantees that the true optimal solution will be greater than or equal to the LB value. However, for large-scale problems, the IP formulation becomes computationally intractable, and the lower bound remains unknown. For problems where lower bounds are available, the gap between the LB value and the PM is observed to be at most 30%. This suggests that the PM performs reasonably well, even in the absence of a lower bound for comparison. To compare the effectiveness of each method, the cost ratio of DA and LS to the cost of PM is shown in Fig. 2, with the cost of our method set to 1, for the 14 problem instances in Table 2. The horizontal axis in Fig. 2 represents the number of dependency relationships, serving as an indicator of problem size. Among the 12 problem instances, there were five instances where the cost of PM was lower than that of the DA and two instances with the same cost. In comparison with LS, there were four instances where the cost of our method was lower and three instances with the same cost among the 14 problem instances.

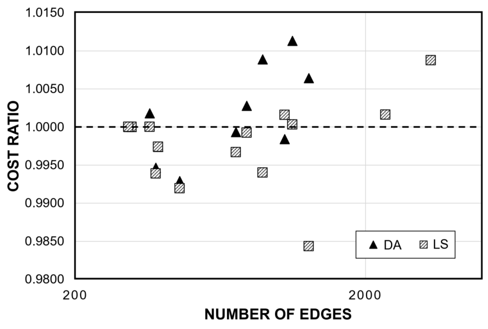

FIGURE 2. Plots of relative costs of DA and LS compared to proposed method.

The results in Fig. 2 show that the effectiveness of the proposed method tends to increase as the problem size increases. Compared to DA, the effectiveness of the proposed method becomes more significant when the number of edges is 800 or more; compared to LS, it becomes more significant when the number of edges is 1400 or more. As the problem size increases, the problem-solving performance decreases, so we think that it might become more challenging to find good solutions with methods like DA and LS without extending the solution time. However, as the proposed method is based on a greedy algorithm, it can obtain good solutions in a shorter time than other methods, even for larger problem size. However, there is no guarantee that this solution is exact. Conversely, for smaller problem sizes, methods such as DA and LS, which conduct more thorough searches, are expected to have an advantage. Based on these results, the proposed method can be considered as a method that reaches good solutions in a shorter time compared to other methods. However, it is important to note that a system for solving real-world BGS problems requires selecting the appropriate method for solving problems depending on the problem size.

V. SUMMARY

We proposed a novel problem class called BGS for controlling the scheduling of systems represented by bipartite graphs with the goal of minimizing costs and maximizing profits in combinatorial optimization problems. We also proposed heuristics for BGS using a greedy algorithm. Using the network circuit and device removal problem, which is included in the BGS problem class, we demonstrated that our heuristics can obtain solutions that are equivalent or better than those in previous studies in a shorter time.

APPENDIX

RELATIONSHIP BETWEEN BGS AND ONE MACHINE SCHEDULING PROBLEM

BGS is a type of single-machine scheduling problem [14], [15]. There are numerous variations of the single-machine scheduling problems [16], [17]. A single-machine scheduling problem, which is closely related to BGS, is the single-machine weighted completion time sum minimization problem. The formulation of the single-machine weighted completion time sum minimum problem is as follows:

$$min. \sum_{j=1}^{N} W_j s_j + \sum_{j=1}^{N} W_j P_j, \tag{13}$$

$$s.t.\ s_j + P_j - M(1 - x_{jk}) \leq s_k, \forall j \neq k, \tag{14}$$

$$x_{jk} + x_{kj} = 1, \quad \forall j < k, \tag{15}$$

$$s_j \leq 0, \quad \forall j = 1, 2, \ldots, N, \tag{16}$$

$$x_{jk} \in \{0, 1\}, \quad \forall j \neq k. \tag{17}$$

Here, W_i, s_j, and P_i represent the weight, start time, and processing time of the job j, respectively. N represents the total number of jobs. Constraints (13) to (17) indicate that the machine can only process one job at a time. BGS can be a single-machine total weighted completion time sum minimization problem with each object as a job and the following additional conditions:

- The processing time of jobs corresponding to the objects in the A-group is 1
- The processing time of jobs corresponding to the objects in the B-group is 0.
- For AND relationships, the start time constraint for object b in the B-group is $s_b = \max_{a \in A_b}(s_a + P_a)$.
- For OR relationships, the start time constraint for object b in the B-group is $s_b = \min_{a \in A_b}(s_a + P_a)$.

Owing to the nonexistent variation of the single-machine scheduling problem with such conditions, BGS can be considered a new problem class.

REFERENCES

[1] K. Jain, M. Mahdian, E. Markakis, A. Saberi, and V. V. Vazirani, "Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP," *J. ACM*, vol. 50, no. 6, pp. 795–824, Nov. 2003.

[2] C. Cheng, Y. Adulyasak, and L.-M. Rousseau, "Robust facility location under demand uncertainty and facility disruptions," *Omega*, vol. 103, Sep. 2021, Art. no. 102429.

[3] S. H. Owen and M. S. Daskin, "Strategic facility location: A review," *Eur. J. Oper. Res.*, vol. 111, no. 3, pp. 423–447, Dec. 1998.

[4] C. S. ReVelle and H. A. Eiselt, "Location analysis: A synthesis and survey," *Eur. J. Oper. Res.*, vol. 165, no. 1, pp. 1–19, Aug. 2005.

[5] R. Z. Farahani and M. Hekmatfar, *Facility Location: Concepts, Models, Algorithms and Case Studies*. Cham, Switzerland: Springer, 2009.

[6] M. Sugimura, M. Kobayashi, H. Matsumura, X. Wang, and P. Palacharla, "Accelerate optical network modernization through quantum-inspired digital annealing," in *Proc. Eur. Conf. Opt. Commun. (ECOC)*, Sep. 2022, pp. 1–4.

[7] A. Klose and A. Drexl, "Facility location models for distribution system design," *Eur. J. Oper. Res.*, vol. 162, no. 1, pp. 4–29, Apr. 2005.

[8] G. E. Vieira, J. W. Herrmann, and E. Lin, "Rescheduling manufacturing systems: A framework of strategies, policies, and methods," *J. Schedul.*, vol. 6, no. 1, pp. 39–62, 2003.

[9] A. M. Ghaithan, "An optimization model for operational planning and turnaround maintenance scheduling of oil and gas supply chain," *Appl. Sci.*, vol. 10, no. 21, p. 7531, Oct. 2020.

[10] A. S. Akopov, "Designing of integrated system-dynamics models for an oil company," *Int. J. Comput. Appl. Technol.*, vol. 45, no. 4, p. 220, 2012.

[11] S. Matsubara, M. Takatsu, T. Miyazawa, T. Shibasaki, Y. Watanabe, K. Takemoto, and H. Tamura, "Digital annealer for high-speed solving of combinatorial optimization problems and its applications," in *Proc. 25th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2020, pp. 667–672.

[12] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski, "SNDlib 1.0—Survivable network design library," *Networks*, vol. 55, no. 3, pp. 276–286, May 2010.

[13] T. Benoist, B. Estellon, F. Gardi, R. Megel, and K. Nouioua, "LocalSolver 1.x: A black-box local-search solver for 0–1 programming," *4OR*, vol. 9, no. 3, pp. 299–316, Sep. 2011.

[14] K. R. Baker and J. Cole Smith, "A multiple-criterion model for machine scheduling," *J. Scheduling*, vol. 6, pp. 7–16, Aug. 2003.

[15] S. Dauzère-Pérès and M. Sevaux, "An exact method to minimize the number of tardy jobs in single machine scheduling," *J. Scheduling*, vol. 7, no. 6, pp. 405–420, Nov. 2004.

[16] M. O. Adamu and A. O. Adewumi, "A survey of single machine scheduling to minimize weighted number of tardy jobs," *J. Ind. Manag. Optim.*, vol. 10, no. 1, pp. 219–241, 2013.

[17] P. Senthilkumar and S. Narayanan, "Literature review of single machine scheduling problem with uniform parallel machines," *Intell. Inf. Manage.*, vol. 2, no. 8, pp. 457–474, 2010.

HIROSHI IKEDA received the B.S. and M.S. degrees in physics from Kyoto University, Japan, in 1992 and 1994, respectively. He joined Fujitsu Ltd., Japan, in 1994, and has been working in various fields with Fujitsu Laboratories Ltd., Japan, since 2009, including statistical and mathematical modeling, financial engineering, and human posture recognition. Since 2021, he has been a Researcher with Fujitsu Ltd., with a focus on combinatorial optimization.

TATSUYA TAKANAGA received the master's degree from Osaka University, Japan, in 2021. He joined Fujitsu Ltd., Japan, in 2021. He is mainly engaging in research of vehicle routing problem and scheduling problem. He specialize in heuristics design for large combinatorial optimization problems and application of mathematical optimization models and algorithms to real problems.