IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON GREEN COMMUNICATIONS ON WIRELESS NETWORKS

Received January 30, 2020, accepted February 14, 2020, date of publication February 20, 2020, date of current version March 3, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2975248

Green Elevator Scheduling Based on IoT Communications

LAN-DA VAN, (Senior Member, IEEE), YI-BING LIN, (Fellow, IEEE), TSUNG-HAN WU, AND TZU-HSIANG CHAO

Department of Computer Science, National Chiao Tung University , Hsinchu 300, Taiwan

Corresponding author: Lan-Da Van (ldvan@cs.nctu.edu.tw)

This work was supported in part by the Ministry of Science and Technology (MOST) under Grant 108-2218-E-009-012, and Grant 106-2221-E-009-028-MY3, in part by the MOST Pervasive Artificial Intelligence Research (PAIR) Labs under Contract MOST 109-2634-F-009-026, and in part by the Center for Open Intelligent Connectivity from the Featured Areas Research Center Program within the framework of the Higher Education Sprout Project by the Ministry of Education (MOE) of Taiwan.

ABSTRACT In this paper, we propose an energy-saving elevator scheduling algorithm to reduce the car moving steps to achieve motor energy saving and green wireless communications. The proposed algorithm consisting of six procedures can attain fewer Internet of Things (IoT) message exchanges (i.e. communication transmissions) between the Scheduler subsystem and the Car subsystem via the core function *AssignCar*(r). The function *AssignCar*(r) is capable of assigning a request to the nearest car through car search globally. From the emulation results for four cars, this work shows that the proposed algorithm outperforms the previous work named as aggressive car scheduling with initial car distribution (ACSICD) algorithm with energy consumption reductions by 49.43%, 47.68%, 37.89%, and 47.65% for up-peak, inter-floor, down-peak, and all-day request patterns, respectively.

INDEX TERMS Car moving step, car scheduling, communication transmissions, elevator system, energy saving, green communications, internet of things (IoT), sensor, waiting/journey time.

I. INTRODUCTION

Due to modern building construction methods and new government regulations of lifting equipment, the elevator technologies have been significantly advanced in the past decade, especially for the purpose of carrying the passengers. During technology revolution, energy saving for green buildings in smart cities is highly demanded. The modern elevators are the potential energy consumption source since the elevator cars carry many passengers every day under the commands exchanged through the exiting wired communication networks as a part of a smart building. Since the cars and the scheduler of the elevator system interact with each other for car moves, inefficient elevator scheduling will lead to redundant communication transmissions and unnecessary car moves such that more energy will be consumed. Herein, we design an energy-saving elevator scheduling that reduces the car moving steps to attain energy reduction and green communications under wireless communications.

There exist many elevator group control studies [1]–[20] including the single-car elevator [1], [7], [8], multi-car elevator [9], [10], [13], [16] and double-decker elevator [14]. Most elevator system studies focus on car scheduling to save waiting times or journey times of the passengers. Approaches based on genetic [12], neural network [3], [5], fuzzy [4], [11], and reinforcement learning scheduling (RLS) [15] attempt to minimize the waiting/journey times. Other approaches attempt to reduce the energy consumption from the viewpoints of circuit structure, circuit control, and scheduling control. Several of them [17]–[20] address the energy issue using the scheduling control algorithms. In [19], Zhang *et al.* proposed an ant colony optimization method considering energy and scheduling for the elevator group control system (the details will be described in the Appendix). However, these scheduling control algorithms cannot easily exercise online scheduling in practice due to batch request demand and iterative learning behavior. Furthermore, to our best knowledge, the Internet of Things (IoT) communications associated with green elevator scheduling have not yet been explored. In particular, existing elevator systems utilize wired communications. In an elevator system called ElevatorTalk [1], we have shown that IoT technologies can be used to implement distributed elevator scheduling, and wireless communication can also be used in the elevator systems without degrading the

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

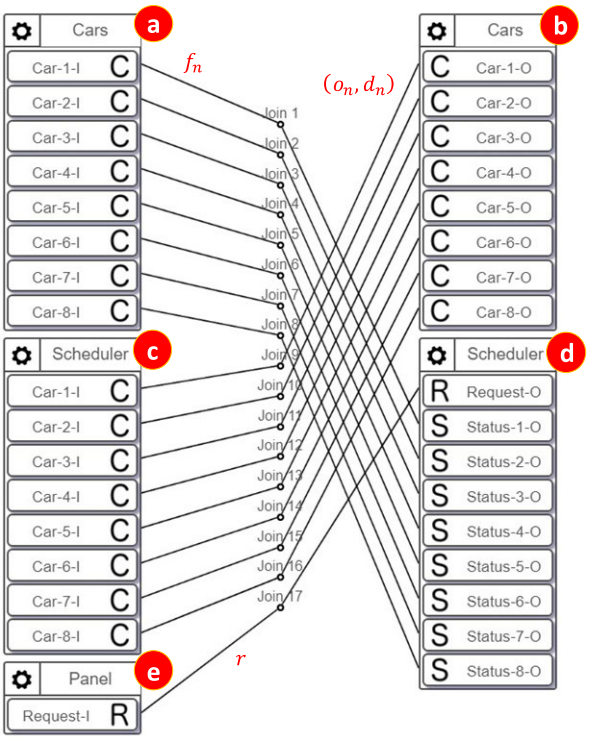

FIGURE 1. Configuring an 8-car elevator system through the ElevatorTalk GUI ($N = 8$).

real-time performance of the cars. We have measured the wireless delay [1], where the expectation is 59.14 ms and the variance is 9.094.

ElevatorTalk has excellent passenger journey time performance as compared with the previous approaches [1]. Another elevator performance measure is ''moving steps'' that determine energy consumption for car motor and wireless IoT communication. Specifically, reducing the car moves will result in fewer IoT message exchanges to achieve green wireless communications. Unfortunately, how to reduce the car moving steps in a systematic way is still open. An interesting question is: how to manage the cars as ''green elevators'' by saving moving steps under the acceptable journey time. To achieve this goal, we propose an energy-saving elevator scheduling algorithm that assigns a request to the nearest car through car search globally. Therefore, the energy consumption of the elevator can be accordingly reduced due to fewer car moving steps and wireless IoT message exchanges.

The remainder of this paper is organized as follows. Section II demonstrates the proposed energy-saving elevator scheduling algorithm. Section III evaluates and compares energy saving of the developed system with other solutions. The last section summarizes our work.

II. ELEVATORTALK PROCEDURES AND ENERGY-SAVING ELEVATOR SCHEDULING ALGORITHM

In [1], we proposed the ElevatorTalk platform that allows one to implement a distributed car scheduling algorithm based on wireless IoT communication. Figure 1 illustrates how an 8-car elevator system is configured through the ElevatorTalk graphical user interface (GUI). In this figure, ElevatorTalk includes the Panel subsystem, the Scheduler subsystem, and the Car subsystem. Every subsystem except for the Panel subsystem consists of an input part (represented by an icon on the left-hand side in Figure 1) and an output part (represented by an icon on the right-hand side in Figure 1). The Panel subsystem (Figure 1 (e)) issues requests to the output part of the Scheduler subsystem (Figure 1 (d)) and the input part of the Scheduler subsystem (Figure 1 (c)) sends instructions to the output part of the Car subsystem (Figure 1 (b)). After finishing the instruction from the Scheduler subsystem (Figure 1 (c)), the input part of the Car subsystem (Figure 1 (a)) sends current floor to the output part of the Scheduler subsystem (Figure 1 (d)). The instructions and the responses are implemented by IoT message exchanges. The proposed energy-saving elevator scheduling algorithm implemented in [1] consists of six procedures including *Panel*, *ReqArrival*, *SchCar*(n), *UpTaskCar*(n), *DownTaskCar*(n), and *Car*(n). Procedure *ReqArrival* has three functions which are *AssignCar*(r), *MoveUp*(r), and *MoveDown*(r). In our previous work [1], the aggressive car scheduling with initial car distribution (ACSICD) stores new arrival requests in the global structures. Then, the cars select the requests to serve independently. ElevatorTalk is an IoT platform where three subsystems are implemented as IoT devices that communicate with each other through wired or wireless communication. Although most elevator systems are centralized with wired communication, wireless communication allows the structure of distributed cars better. In a distributed car structure, it is essential to reduce the number of message exchanges for car moves. In other words, it is important to reduce car moves to achieve green wireless communication as well as car motor energy saving. This paper proposes the energy-saving elevator scheduling algorithm that reduces the car moving steps to achieve the goal of energy saving for IoT communication. Unlike the previous proposed solution ACSICD, as long as a new request is issued by Procedure *Panel* in the Panel subsystem, the energy-saving elevator scheduling algorithm immediately assigns the request to a ''carefully'' selected car n by Procedure *ReqArrival* in the Scheduler subsystem. Then Procedures *SchCar*(n), *UpTaskCar*(n), and *DownTaskCar*(n) in the Scheduler subsystem direct the n-th car to serve this request. When the n-th car reaches the floor that has a request with the same direction, the n-th car serves the request even if the request is not assigned to the n-th car. The system symbols used to describe the algorithm are listed in Table 1.

A. PROCEDURE Panel AND ReqArrival

Procedure *Panel* shown in Figure 2 interacts with the elevator car operating panels (ECO) that can be a traditional panel in the hall of each floor or a smart user interface (UI) interpreted by web browsers in cell phones, personal computers, and laptops.

TABLE 1. System symbols.

Symbol	Description
$C_{s,u}(n)$	The set of all unhandled $f_{s,r}$ that are assigned to the n-th car, where D_r is *Up*
$C_{s,d}(n)$	The set of all unhandled $f_{s,r}$ that are assigned to the n-th car, where D_r is *Down*
$C_t(n)$	The set of the target floors where the n-th car has to stop in the current moving direction; $C_t(n) = \{f_{t,c}(n,1), f_{t,c}(n,2), \ldots, f_{t,c}(n,j_n)\}$
d_n	The current moving direction of the n-th car; $d_n \in \{Up, Down, Idle\}$
D_r	The moving direction of request r; $D_r \in \{Up, Down\}$
f_n	The current floor of the n-th car
$f_{s,r}$	The start floor of request r
$f_{t,c}(n,j)$	The j-th target floor in the n-th car's stop list $C_t(n)$, where $1 \le j \le j_n$
$f_{t,r}(k)$	The k-th target floor of request r, where $1 \le k \le k_r$
F	The number of floors in the building
j_n	The number of target floors in $C_t(n)$
k_r	The number of target floors of request r
L_u	The set Uplist of all unhandled $f_{s,r}$, where D_r is *Up*
L_d	The set Downlist of all unhandled $f_{s,r}$, where D_r is *Down*
M	The number of requests issued during the observation period
N	The number of cars in the elevator system
o_n	A flag that indicates if the n-th car should open door or not; $o_n \in \{0,1\}$
r	A request from an ECO panel in the Panel subsystem; $r = (f_{s,r}, D_r, S_r)$
r_m	The m-th request from an ECO panel in the Panel subsystem
S_r	The set of target floors of request r; $S_r = \{f_{t,r}(1), f_{t,r}(2), \ldots, f_{t,r}(k_r)\}$

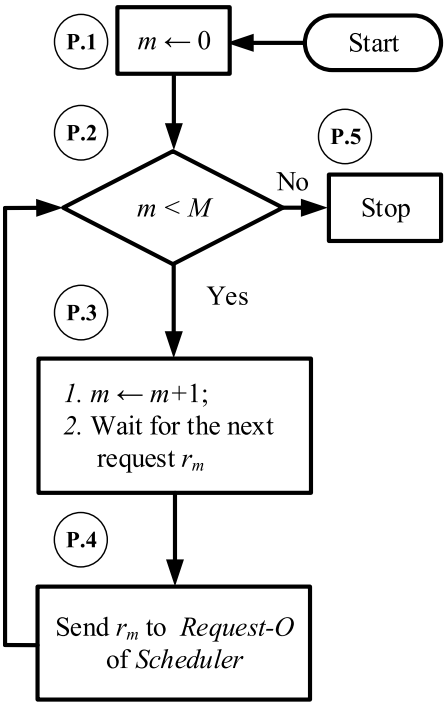

FIGURE 2. Flowchart for Procedure *Panel*.

An ECO panel issues a request when a passenger pushes the button, and then the request is received by Procedure *Panel*. The format of the m-th request is $r_m = (f_{s,r_m}, D_{r_m}, S_{r_m})$, where f_{s,r_m} is the start floor from which r_m is issued. $D_{r_m} \in \{Up, Down\}$ is the moving direction of r_m. S_{r_m} is the set of target floors of the m-th request r_m. In the loop (Steps P.2-P.5), the procedure waits for request r_m from the ECO panel at Step P.3, and then sends the request to the Scheduler subsystem at Step P.4. We assume that the elevator system is shut down for maintenance after it has served M requests.

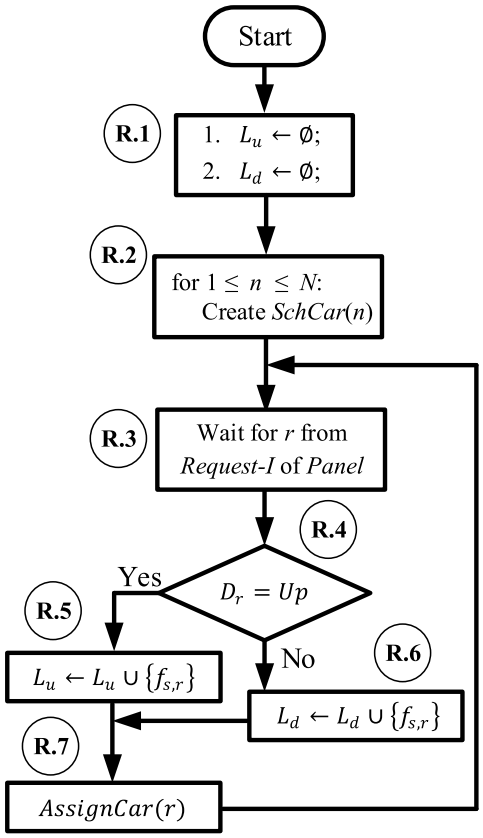

FIGURE 3. Flowchart for Procedure *ReqArrival*.

Procedure *ReqArrival* illustrated in Figure 3 is responsible for receiving requests from Procedure *Panel* and dispatching the requests to proper cars. Step R.1 initializes the related variables. Step R.2 creates N threads of *SchCar*(n) to process the requests assigned to the n-th car, where $1 \le n \le N$. The procedure maintains the global data structures Uplist L_u and Downlist L_d, where L_u stores all unserved $f_{s,r}$ that are the start floors of the requests with $D_r = Up$ and L_d stores all unserved $f_{s,r}$ with $D_r = Down$ (Steps R.4-R.6). At Step R.7, the core function *AssignCar*(r) is responsible for assigning each request to a proper car in the Scheduler subsystem, which is elaborated in the next subsection.

B. FUNCTION AssignCar(r)

Function *AssignCar*(r) searches for the car that takes minimum car moving steps to handle r. The flowchart is illustrated in Figure 4, where the symbols for Function *AssignCar*(r) are defined in Table 2. Two variables n and X^* are initialized at Step A.1, where n is set to 1 to represent the first car and

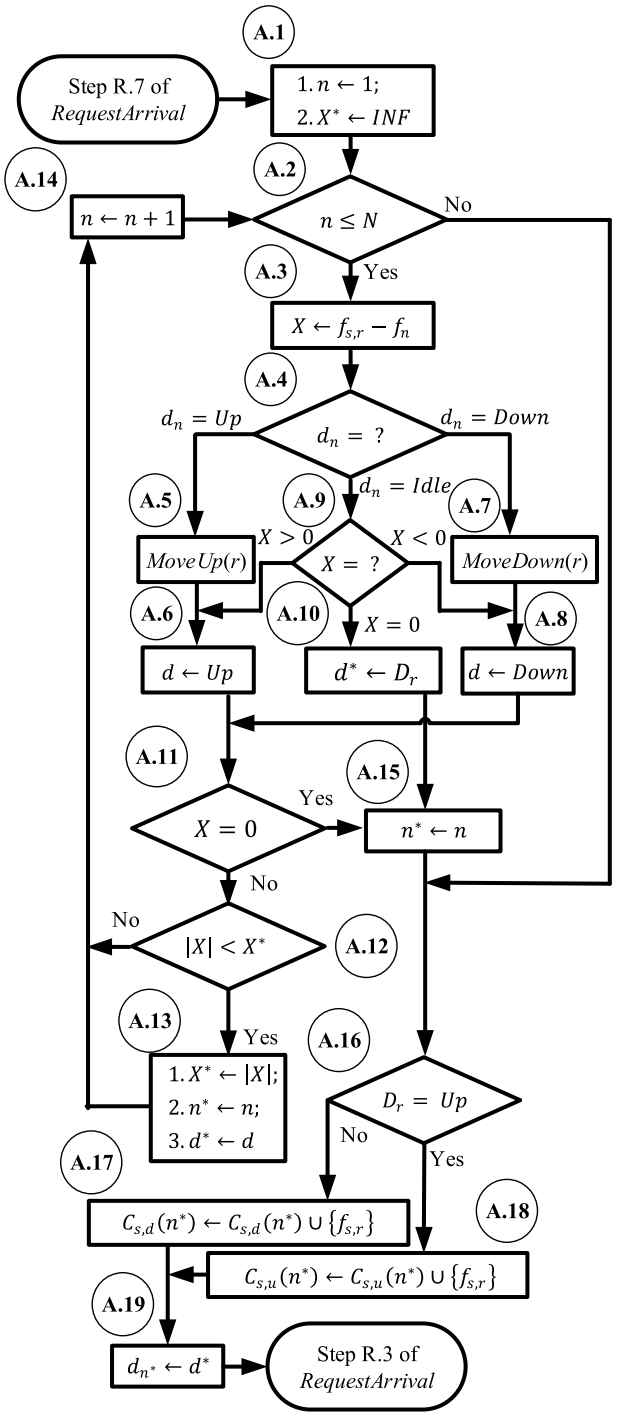

FIGURE 4. Flowchart for Function *AssignCar(r)*.

X^* is set to INF. X^* is used to store the temporary minimum moving steps to pick up request r. Steps A.2-A.14 are a loop to find the car to handle the request. Three local variables X, n^*, and d^* are used. The sign and absolute value of X denote the relative locations between request r and the n-th car and the actual moving steps for the n-th car to pick up request r, respectively. n^* denotes the tentative nearest car to request r and d^* is used to determine d_{n^*} (i.e. the moving direction of the n^*-th car) when leaving Function $AssignCar(r)$.

TABLE 2. Symbols for Function *AssignCar*(*r*).

Symbol	Description
d^*	The assigned direction of the n^*-th car
d	The tentative car direction
f_h	The highest floor that the n-th car will reach for current requests it will handle; $f_{s,r} \le f_h \le F$
f_l	The lowest floor that the n-th car will reach for current requests it will handle; $1 \le f_l < f_{s,r}$
n^*	The nearest car for request r
X	The relative locations and moving steps between $f_{s,r}$ and f_n
X^*	The temporary minimum moving steps between $f_{s,r}$ and f_{n^*}

For a car n, Step A.3 sets X to $f_{s,r} - f_n$, where $f_{s,r}$ is the start floor of request r and f_n is the current floor of the n-th car. Step A.4 uses the current moving direction of the n-th car d_n to check if the n-th car is moving. If d_n is Up, Function $MoveUp(r)$ at Step A.5 is invoked to update X (to be elaborated in Section II.C). Step A.6 sets d to Up. Step A.11 checks if X is 0. If so, Step A.15 sets n^* to n and breaks the loop. If X is not 0 at Step A.11, Step A.12 checks whether the absolute value of X is smaller than X^*. If so, Step A.13 sets X^* to $|X|$, n^* to n, and d^* to d. At Step A.14, n is incremented by 1. If d_n is $Down$ at Step A.4, then Step A.7 is executed to update X. Step A.8 sets d to $Down$ and the procedure proceeds to Step A.11. If d_n is $Idle$ at Step A.4, Step A.9 uses the sign of X to determine the relative locations between request r and the n-th car. If $X > 0$, the n-th car moves up to handle the request, the flow proceeds to Step A.6 and sets d to Up. If $X < 0$, to the contrary, the flow proceeds to Step A.8 and sets d to $Down$. If X equals 0 at Step A.9, it implies that the n-th car is $Idle$ on $f_{s,r}$ and no other car can be closer to $f_{s,r}$ than the n-th car. Thus, Step A.10 sets d^* to D_r and then Step A.15 is executed to set n^* to n and breaks the loop. If $n > N$ at Step A.2, Step A.16 is executed to check whether D_r is Up. If so, $f_{s,r}$ is included in $C_{s,u}(n^*)$. The set $C_{s,u}(n^*)$ stores the move-up requests to be handled by the n-th car. Otherwise, $f_{s,r}$ is included in $C_{s,d}(n^*)$ for the move-down requests. Step A.19 sets d_{n^*} to d^*. Then we exit $AssignCar(r)$ and go back to Step R.3 of Procedure *ReqArrival*. Note that if the n-th car is moving, d_{n^*} will remain unchanged after Step A.19.

C. FUNCTION MoveUp(r)

The X value calculated at Step A.3 of $AssignCar(r)$ does not represent the number of floors that the n-th car moves if this car needs to change its direction to pick up request r. Figure 5 illustrates the relationship between a car and a request. Function $MoveUp(r)$ recalculates X to obtain the actual moving steps between $f_{s,r}$ and f_n according to four cases. Cases 1.1 & 1.2 (Figure 5 (a) and (b)) represent that $f_{s,r}$ of the request is higher than or equal to f_n of the n-th move-up car (i.e. $f_{s,r} \ge f_n$). Cases 2.1 & 2.2 (Figure 5 (c) and (d)) represent that $f_{s,r}$ of the request is lower than f_n of the n-th move-up car (i.e. $f_{s,r} < f_n$). In Cases 1.1 & 2.2, the directions of the n-th car and the request are the same. Conversely, in Cases 1.2 & 2.1, the directions of the n-th car

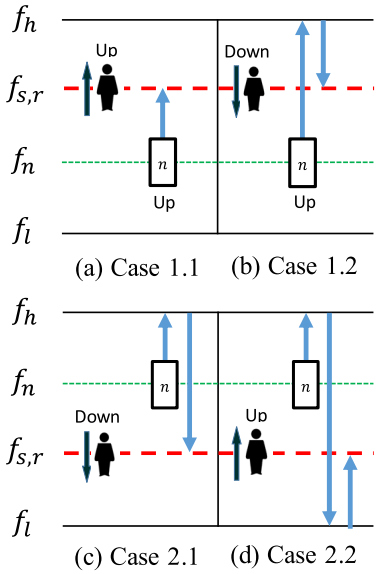

FIGURE 5. Four cases in Function *MoveUp(r)* for the request assignment.

and the request are opposite. Note that in Cases 1.2 & 2.2, the n-th car needs one more direction change before the n-th car picks up request r compared with Cases 1.1 & 2.1, respectively. The flowchart of Function $MoveUp(r)$ is shown in Figure 6. If the value of X at Step MU.1 is positive or zero, the function proceeds to Step MU.2 (Case 1); otherwise, the function proceeds to Step MU.4 (Case 2). Two variables f_h and f_l, where $f_{s,r} \le f_h \le F$ and $1 \le f_l < f_{s,r}$, are used to store the highest and the lowest floors to be reached by the n-th car. Note that f_l is set to F (the height of the building) at Step MU.3 to indicate that f_l is unused when setting X at Step MU.12. The four cases are detailed as follows:

Case 1.1. ($f_{s,r}$ of request r with $D_r = Up$ is above or equal to f_n). Step MU.2 checks if $D_r = Down$. In this case, D_r is Up. We exit $MoveUp(r)$ and go back to Step A.6 of $AssignCar(r)$ (i.e. $X = f_{s,r} - f_n$, which has been set at Step A.3 of $AssignCar(r)$ already).

Case 1.2. ($f_{s,r}$ of request r with $D_r = Down$ is above or equal to f_n). Step MU.3 sets f_l to F. Then, the function proceeds to compute the f_h value. Step MU.8 checks if there exists any $f_{s,r}$ in $C_{s,u}(n)$ that is higher than f_n. If so, f_h is set to F at Step MU.9. Otherwise, f_h is set to $\max\{f_{s,r}, \max C_t(n), \max C_{s,d}(n)\}$ at Step MU.10. At Step MU.11, since f_l is F, the function goes to Step MU.12 to recalculate X as $(f_h - f_n) + (f_h - f_{s,r})$.

Case 2.1. ($f_{s,r}$ of request r with $D_r = Down$ is below f_n). Step MU.3 sets f_l to F. The function proceeds to set f_h at Steps MU.8-MU.10 as described in Case 1.2. Because f_l equals F at Step MU.11, X is set to $(f_h - f_n) + (f_h - f_{s,r})$ at Step MU.12.

Case 2.2. ($f_{s,r}$ of request r with $D_r = Up$ is below f_n). Step MU.5 checks if there exists any $f_{s,r}$ in $C_{s,d}$. If so, Step MU.6 sets f_l to 1. Otherwise, f_l is set to $\min\{f_{s,r}, \min C_t(n), \min C_{s,u}(n)\}$. The function proceeds to set f_h at Steps MU.8-MU.10 as described in Case 1.2. Since f_l is not equal to F at Step MU.11, Step MU.13 updates X to $(f_h - f_n) + (f_h - f_l) + (f_{s,r} - f_l)$.

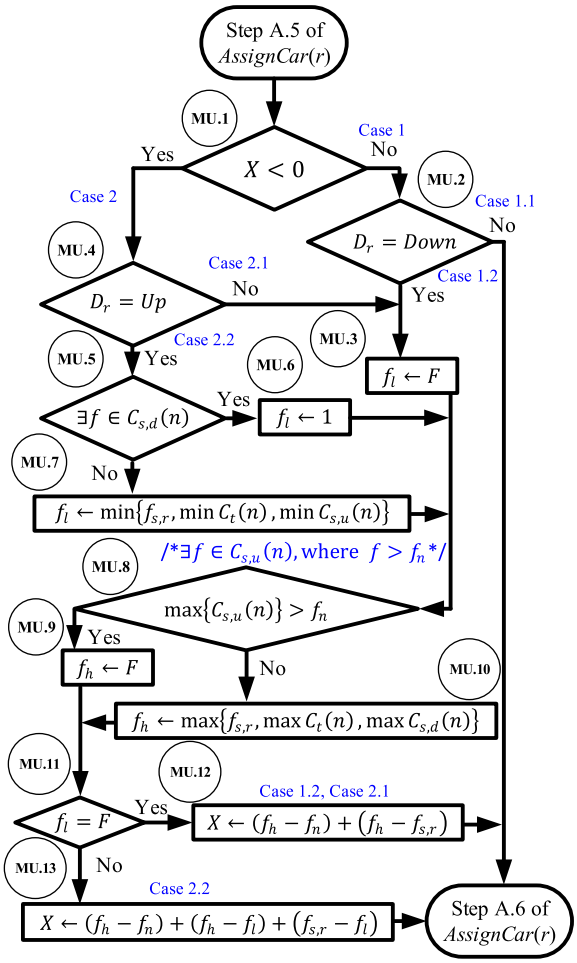

FIGURE 6. Flowchart for Function *MoveUp(r)*.

Function $MoveDown(r)$ is the same as Function $MoveUp(r)$ except for the direction reverses. This part is omitted.

D. PROCEDURE SchCar(n)

Procedure $SchCar(n)$ instructs the n-th car to pick up and deliver the passengers to the destinations. Figure 7 illustrates the flowchart using four global data structures. d_n is the current moving direction of the n-th car, where $d_n \in \{Up, Down, Idle\}$. The up-request set $C_{s,u}(n)$ indicates the set of all unhandled $f_{s,r}$ assigned to the n-th car with $D_r = Up$. Similarly, the down-request set $C_{s,d}(n)$ contains all unhandled $f_{s,r}$ assigned to the n-th car with $D_r = Down$. The set $C_t(n)$ includes the target floors that the n-th car has to stop in the current moving direction. Herein, $C_t(n) = \{f_{t,c}(n,1), f_{t,c}(n,2), \ldots, f_{t,c}(n,j_n)\}$, where $f_{t,c}(n,j)$ denotes the j-th target floor in the n-th car's stop

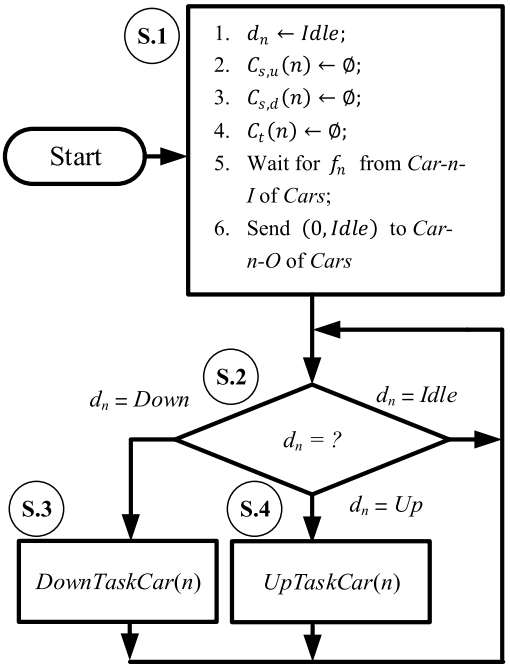

FIGURE 7. Flowchart for Procedure *SchCar(n)*.

list $C_t(n)$ for $1 \leq j \leq j_n$ and $j_n = |C_t(n)|$. At Step S.1, d_n is set to *Idle*, and $C_{s,u}(n)$, $C_{s,d}(n)$, and $C_t(n)$ are set to empty. At Step S.1.5, the procedure waits to receive the updated status f_n of the n-th car from Procedure $Car(n)$ (to be elaborated in Section II.F). Procedure $SchCar(n)$ of the Scheduler subsystem and Procedure $Car(n)$ of the Car subsystem must be synchronized. Therefore, Step S.1.6 sends message $(0, Idle)$ to acknowledge Procedure $Car(n)$ that the connection path is ready. After initialization, Procedure $SchCar(n)$ enters a loop to handle the direction of the n-th car (Steps S.2-S.4). Step S.2 checks the direction of the n-th car. If d_n is *Up*, the flow proceeds to Step S.4 and invokes Procedure $UpTaskCar(n)$ to instruct the n-th car to serve the assigned requests (to be elaborated in Section II.E). If d_n is *Down*, Procedure $DownTaskCar(n)$ at Step S.3 is invoked. Otherwise, when d_n is *Idle*, the procedure stays at Step S.2 until d_n is updated due to a new assigned request to the n-th car in Function $AssignCar(r)$ described in Section II.B.

E. PROCEDURE UpTaskCar(n)

Procedure $UpTaskCar(n)$ in Figure 8 handles the requests for the n-th car with $d_n = Up$. Step U.1 assumes that the n-th car will open the door on this floor ($o_n = 1$). When the n-th car arrives at a floor, the n-th car serves not only $f_{s,r}$ of the requests assigned to itself in $C_{s,u}(n)$ but also unserved $f_{s,r}$ of the requests in L_u. Therefore, at Step U.2, the procedure compares L_u and $C_{s,u}(n)$ to remove $f_{s,r}$ of the requests that have already been served by other cars. At Step U.3, the n-th car checks if f_n is in $C_t(n)$ or L_u. If so, the procedure goes to Step U.4 and then Steps U.4.1-U.4.3 remove current floor f_n from $C_t(n)$, L_u, and $C_{s,u}(n)$ (i.e. passengers exit the n-th car). Then Step U.4.4 performs the union of set $C_t(n)$ and S_r, where S_r is the set of the target floors of

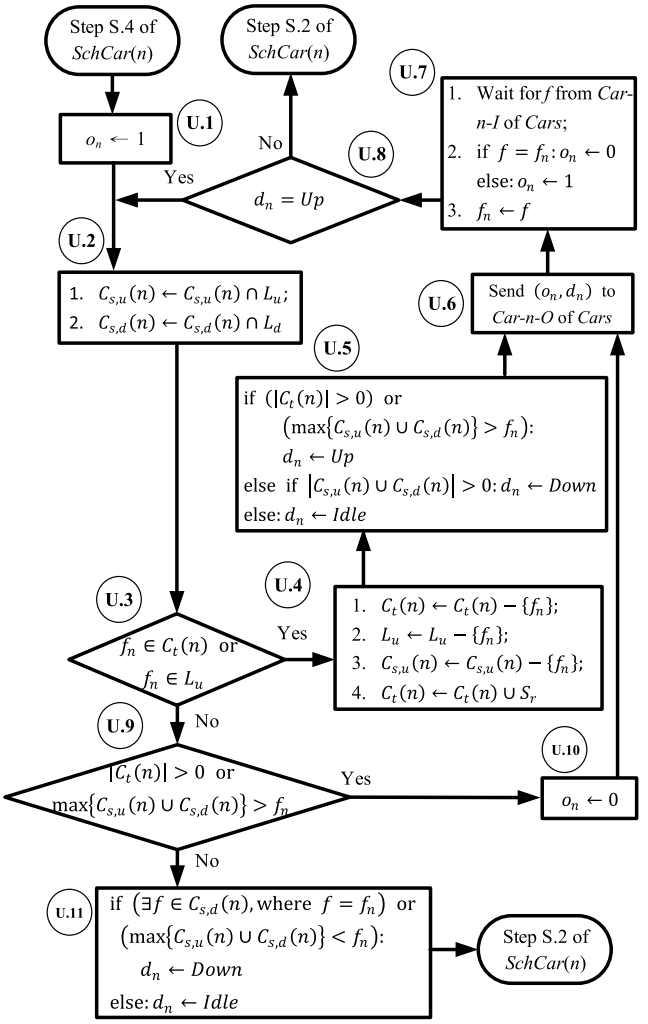

FIGURE 8. Flowchart for Procedure *UpTaskCar(n)*.

request r (i.e. passengers enter the n-th car and push the target floor buttons). At Step U.5, if $C_t(n)$ is not empty or there exists a request with $f_{s,r}$ that is higher than f_n (i.e. $\max\{C_{s,u}(n) \cup C_{s,d}(n)\} > f_n$), then d_n is set to *Up*. Otherwise, if $C_t(n)$ is empty and there still exists a request in $C_{s,u}(n)$ or $C_{s,d}(n)$ (i.e. $f_{s,r}$ of the request is below f_n), then d_n is set to *Down*. If $C_{s,u}(n)$, $C_{s,d}(n)$, and $C_t(n)$ are empty, d_n is set to *Idle*. Step U.6 sends message (o_n, d_n) to the n-th car of the Car subsystem and Step U.7.1 waits for f from the n-th car. Note that Steps U.7.2 and U.7.3 are designed to prevent redundant door open operations [1]. At Step U.8, if d_n is not *Up*, the procedure returns to Step S.2 of $SchCar(n)$. Otherwise, the procedure proceeds to Step U.2 and the loop goes on. At Step U.3, if f_n belongs to neither $C_t(n)$ nor L_u, the procedure proceeds to Step U.9. This step is the same as the first condition of Step U.5. If there exists a target floor or an unserved request whose $f_{s,r}$ is above the n-th car, the procedure proceeds to Step U.10 and sets o_n to 0 since no passenger enters/leaves f_n. Otherwise, the procedure proceeds to Step U.11 and checks if there exists any request in $C_{s,d}(n)$ whose $f_{s,r}$ is the same with f_n or there still exists any request

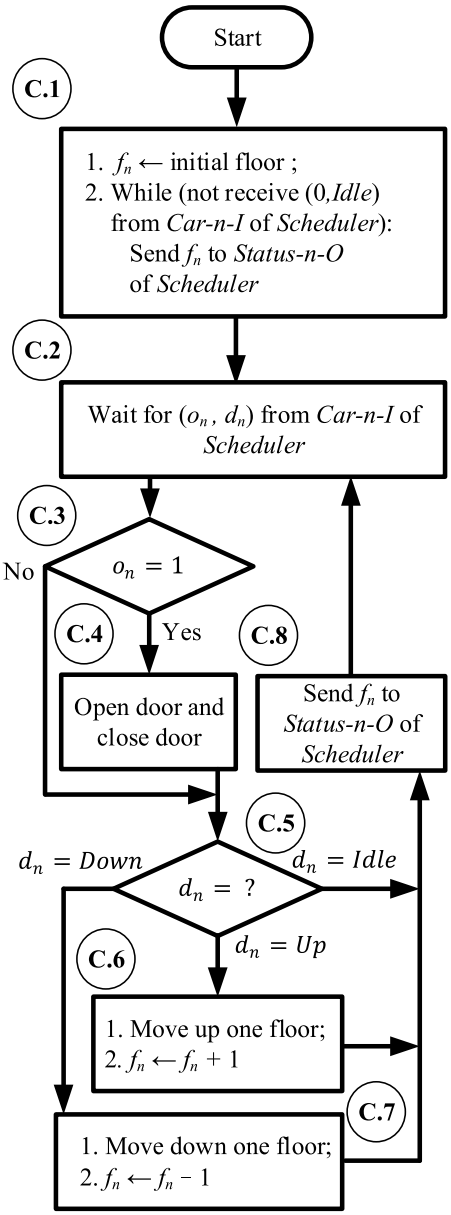

FIGURE 9. Flowchart for Procedure *Car(n)*.

in either the sets $C_{s,u}$ (n) or $C_{s,d}$ (n) whose $f_{s,r}$ is below f_n. If so, the direction d_n is set to *Down*. Otherwise, the n-th car becomes an idle car and the procedure returns to Step S.2 of $SchCar(n)$. Since Procedure $DownTaskCar(n)$ is the same as Procedure $UpTaskCar(n)$ except for the direction reverses, we do not detail this part herein.

F. PROCEDURE Car(n)

The car subsystem has N threads and each of them executes Procedure $Car(n)$ in Figure 9. For example, for an 8-car system, the Car subsystem has such threads for $1 \leq n \leq 8$. Procedures $SchCar(n)$ and $Car(n)$ synchronize the communication at Step C.1 by a handshaking. In the Car subsystem, Procedure $Car(n)$ continues to wait for the message (o_n,d_n) from Procedure $SchCar(n)$ in the Scheduler subsystem described in Sections II.D and E. If o_n= 1 at Step C.3, the car opens and closes the door to load/unload the passengers at Step C.4. If d_n is *Up* at Step C.5, the car moves up one floor and f_n is incremented by 1 at Step C.6. If d_n is *Down* at Step C.5, the car moves down one floor and f_n is decremented by 1 at Step C.7. After f_n is updated at Steps C.6 and C.7, through ElevatorTalk configuration in Figure 1, Step C.8 sends f_n to either Procedure $UpTaskCar(n)$ or Procedure $DownTaskCar(n)$ in the Scheduler subsystem described in Section II.E. If d_n is *Idle* at Step C.5, the flow proceeds to Step C.8 directly.

TABLE 3. Symbols used in the performance evaluation.

Symbol	Description
e_a	The energy for one motor acceleration or deceleration; $e_a = 22.5$kJ
E	The total energy consumption during the observation period T
$E_a(n)$	The acceleration and deceleration energy of the n-th car during T
E_a	The total acceleration and deceleration energy during T
$E_v(n)$	The moving energy of the n-th car during T
E_v	The total moving energy during T
g	Gravity acceleration, where $g = 9.8\,m/s^2$
$h(n,m,s)$	The distance between the s-th stop and its next stop when the n-th car is on the moving path to the start floor of request r_m
$\bar{q}$	The average mass of a passenger; $\bar{q} = 65kg$
q	The difference between the counterweight mass and the mass of the n-th car, where $q = 100kg$
R_n	The number of requests served by the n-th car during the observation period T
t_m	The arrival time of r_m; by default, $t_0 = 0$
T	The observation period
$T_w(m,n)$	The waiting time of r_m if it is served by the n-th car
T_w	The average waiting time during T
$T_j(m,n)$	The journey time of r_m if it is served by the n-th car
T_j	The average journey time during T
$\theta(n,m,s)$	The number of passengers in the n-th car on the moving path from the s-th stop to the start floor of request r_m
$\mu_{n,m}$	The number of stops between request $r_{m'}$ and request r_m, where $r_{m'}$ is the previous request served by the n-th car on its moving path to the start floor of request r_m
λ	The arrival rate in terms of the number of requests per second
τ_m	The inter-arrival time between request r_{m-1} and r_m, where r_0 is a null request, and $\tau_m = t_m - t_{m-1}$

III. PERFORMANCE EVALUATION

In this section, we show the detailed experimental results and performance evaluation. In Section III.A, we use an energy consumption model to evaluate the energy consumption of various scheduling algorithms. In Section III.B, we describe four request patterns used for performance evaluation. In Section III.C, performance comparisons of the energy-saving elevator scheduling algorithm and previous proposed algorithms are demonstrated.

A. ENERGY CONSUMPTION MODEL

We adopt the energy consumption model in [19] to evaluate the energy performance compared with other methods. The symbols used to describe the energy consumption model are listed in Table 3.

TABLE 4. The parameters of traffic types.

Traffic Type	Peak Arrival Rate (λ)	Peak Interval	Base Arrival Rate (λ)
up-peak	96 requests/5min	15 min	26.88 requests/5min
inter-floor	N/A	N/A	25.92 requests/5min
down-peak	168 requests/5min	15 min	0.89 requests/5min

The total energy consumption E of an elevator system during the observation period T is

$$E = E_a + E_v,$$

where E_a is the total acceleration and deceleration energy during T, which is defined by

$$E_a = \sum_{n=1}^{N} E_a(n) = \sum_{n=1}^{N} \sum_{m=1}^{R_n} 2 \times \mu_{n,m} \times e_a.$$

In the above equation, $E_a(n)$ is the acceleration and deceleration energy of the n-th car during T, R_n is the number of requests served by the n-th car during T, $\mu_{n,m}$ is the number of stops between request $r_{m'}$ and request r_m, where $r_{m'}$ is the previous request served by the n-th car on its moving path to the start floor of request r_m, and e_a is the energy consumed by one motor acceleration or deceleration ($e_a = 22.5$kJ). E_v is the total moving energy during T defined as follows:

$$\begin{aligned} E_v &= \sum_{n=1}^{N} E_v(n) \\ &= \sum_{n=1}^{N} \sum_{m=1}^{R_n} \sum_{s=1}^{\mu_{n,m}} [|\theta(n,m,s) \times \bar{q} - q| \times g \\ &\quad \times h(n,m,s)], \end{aligned}$$

where $\theta(n, m, s)$ is the number of passengers in the n-th car on the moving path from the s-th stop to the start floor of request r_m, $h(n, m, s)$ is the distance between the s-th stop and its next stop when the n-th car is on the moving path to the start floor of request r_m. $\bar{q}$ is the average mass of a passenger ($\bar{q} = 65kg$), q is the difference between the counterweight mass and the mass of the n-th car ($q = 100kg$), and g is the acceleration of gravity ($g = 9.8m/s^2$).

B. GENERATION OF REQUEST PATTERNS

In order to evaluate the performance of the proposed energy-saving elevator scheduling algorithm, we adopt three types of traffic including up-peak, inter-floor, and down-peak [2] for $T = 60$ minutes in a 16-floor building. The up-peak traffic occurs in the beginning of the workday while the down-peak traffic occurs at the end of the workday. Otherwise, the traffic type belongs to the inter-floor traffic. As shown in Table 4, three types of traffic are characterized by three parameters: peak arrival rate, peak interval, and base arrival rate.

According to the peak arrival rate in the peak interval (i.e. 15 minutes) and the base arrival rate in the remaining interval, the inter-arrival times (τ_m) of the requests are produced by

$$\tau_m = \frac{-lnU}{\lambda},$$

where λ denotes the arrival rate in terms of the number of requests per second and U represents a random value in (0,1). We describe four types of generated request patterns according to three traffic types with different arrival rates λ as follows:

Up-peak: The up-peak request indicates that the start floor is the base floor (i.e. 1F in our emulation) and target floor is randomly selected from 2F-16F with the same probability. The peak interval ranges from the 20-th minute to the 35-th minute during the 60-minute observation period.

Inter-floor: The inter-floor request indicates that the start floor and target floor are randomly selected from 1F-16F with the same probability, where the start floor and target floor are mutually exclusive.

Down-peak: The down-peak request indicates that the target floor is the base floor and start floor is randomly selected from 2F-16F with the same probability. The peak interval ranges from the 30-th minute to the 45-th minute during the 60-minute observation period.

All-day: The all-day request indicates that the start floor and target floor are selected according to the Poisson distribution defined in [15], where the all-day traffic consists of up-peak, inter-floor, and down-peak traffic. The request in all-day request pattern arrives at the interval from 7:00 AM-9:00 PM.

The up-peak, down-peak, and inter-floor request patterns have 522, 504, and 300 requests for 60 minutes, respectively, and all-day request pattern has 861 requests in our emulation experiment. Note that in our emulation, we assume each request r has only one target floor.

C. COMPARING ENERGY-SAVING ELEVATOR SCHEDULING ALGORITHM WITH THE PREVIOUSLY PROPOSED ALGORITHMS

We conduct timing emulation where Function *sleep* is utilized to emulate moving up/down one floor and door opening and closing in Procedure *Car*(n) in Section II.F. Based on the motor mechanical characteristics, the time for a car to move up/down one floor is not fixed [19]. Three car moving times are summarized as follows: 1) When a car moves up/down only one floor, the car moving time is 3.46s. 2) When a car moves up/down two floors, the car moving time is 4.9s. 3) When a car moves up/down more than two floors, the car moving time is set to 4.9s for the first two floors and 1.2s for each extra floor.

Through the above emulation setting, we emulate the proposed energy-saving elevator scheduling algorithm in

TABLE 5. Performance for three scheduling algorithms. ($N = 4, F = 16$).

Request Pattern	Algorithm	Average Waiting Time (s)	Average Journey Time (s)	Energy Consumption (kJ)	Communication Transmissions
Up-peak	Energy-saving	20.79	68.45	31378.79	2399
	ACSICD	13.59	52.03	62055.43	3373
	Ant colony	49.80	N/A	33314.89	N/A
Inter-floor	Energy-saving	11.36	29.77	28954.70	2218
	ACSICD	11.94	29.45	55337.49	3051
	Ant colony	37.13	N/A	21338.74	N/A
Down-peak	Energy-saving	29.71	76.97	19155.02	936
	ACSICD	33.19	72.44	30841.01	1319
	Ant colony	32.31	N/A	36364.63	N/A
All-day	Energy-saving	19.99	56.56	79866.43	5711
	ACSICD	21.50	56.78	152558.92	7155

TABLE 6. Performance for two scheduling algorithms. ($N = 8, F = 16$).

Request Pattern	Algorithm	Average Waiting Time (s)	Average Journey Time (s)	Energy Consumption (kJ)	Communication Transmissions
All-day	Energy-saving	17.87	53.78	80081.62	5670
	ACSICD	18.54	53.68	155009.28	7420

Section II and ACSICD [1] using the request patterns described in Section III.B. In our emulation experiments, we evaluate the performance by the average waiting/journey time and the energy consumption. The average waiting time T_w is defined by

$$T_w = \sum_{m=1}^{M} \sum_{n=1}^{N} T_w(m, n)/M,$$

where $T_w(m, n)$ denotes the duration between the arrival time t_m of the m-th request and the time that the n-th car picks up the m-th request. The average journey time T_j is formulated by

$$T_j = \sum_{m=1}^{M} \sum_{n=1}^{N} T_j(m, n)/M,$$

where $T_j(m, n)$ represents the duration between the arrival time t_m of the m-th request and the time that the n-th car which handles the m-th request arrives at the target floor of the request. The energy consumption E is described in Section III.A. Table 5 shows the average journey times and the energy consumption of various algorithms with the up-peak, inter-floor, down-peak, and all-day request patterns for 4 cars. The proposed energy-saving elevator scheduling algorithm outperforms ACSICD with energy consumption reductions by 49.43%, 47.68%, 37.89%, and 47.65% in terms of up-peak, inter-floor, down-peak, and all-day request patterns, respectively. On the contrary, compared with the energy-saving elevator scheduling algorithm, ACSICD reduces the average journey times in terms of up-peak, inter-floor, down-peak, and all-day request patterns by 23.99%, 1.07%, 5.89%, and -0.39%, respectively. Compared with the performance in [19], our approach outperforms the ant colony algorithm in terms of the energy consumption saving for up-peak and down-peak request patterns by 5.81% and 47.33%, respectively, where the detailed algorithm is described in Appendix.

For the emulations with 8 cars as shown in Table 6, the proposed energy-saving elevator scheduling algorithm outperforms ACSICD with energy consumption reduction by 48.34% for all-day request pattern. On the contrary, compared with energy-saving elevator scheduling algorithm, ACSICD reduces the average journey time for all-day request pattern by 0.19%.

The communication transmission in Tables 5 and 6 is the number of message exchanges between the Scheduler subsystem and the Car subsystem to control the cars' behavior. In terms of the communication transmissions, the energy-saving elevator scheduling algorithm can attain the reductions by 28.88%, 27.3%, 29.04%, and 20.18% for $N = 4$ for up-peak, inter-floor, down-peak, and all-day request patterns and 23.58% for $N = 8$ for all-day request pattern compared with ACSICD, respectively. Thus, the proposed algorithm can achieve green communications.

From Tables 5 and 6, compared with ACSICD, the energy-saving elevator scheduling algorithm has better energy saving in all request patterns because of the fewer car moving steps and communication transmissions with a small amount of increased average journey time.

IV. CONCLUSION

In this paper, the energy-saving elevator scheduling algorithm is proposed and verified to reduce the energy consumption in an elevator system with IoT communications. Four types of request patterns including up-peak, inter-floor, down-peak, and all-day are used to verify the performance comparisons for 4 cars and the all-day request pattern is used to evaluate the performance comparison for 8 cars. Through these comparisons, this study outperforms ACSICD [1] in all request patterns in terms of energy saving due to fewer car moving steps and communication transmissions

(i.e. IoT message exchanges) with acceptable average waiting/journey time. Although making cars busier in the elevator system could achieve less average journey time, without carefully selecting a car for a request could incur more energy consumption. Furthermore, the evaluation of communication transmissions shows that the energy-saving elevator scheduling algorithm could achieve green communications in the ElevatorTalk system.

APPENDIX

In [19], the authors proposed an elevator scheduling using the ant colony optimization method to minimize energy consumption. Each elevator car and request are regarded as an ant and a target (i.e. food), respectively, where the requests and the cars are mutually connected to establish many paths. The path $\pi_{n,m}$ possessing the pheromone ($\varphi^k_{n,m}$) represents the path from the n-th ant (i.e. car) to the m-th food (i.e. request). When the n-th car passes path $\pi_{n,m}$ to serve the m-th request, the pheromone is strengthened. That means that the n-th car with a larger pheromone value and a lower energy consumption on the path $\pi_{n,m}$ has a higher probability to serve the m-th request. This algorithm computes all pheromones iteratively. After k searching iterations, the car system has the converged scheduling. We use the configuration (4-car system with $k = 200$) in [19] to demonstrate the ant colony optimization for elevator scheduling. The algorithm periodically collects requests. In each period, the collected requests are divided into two categories that are move-up requests and move-down requests. After receiving the requests, the algorithm starts to optimize the scheduling iteratively. The algorithm executes the following steps to find an optimized solution:

Step 1. After collecting the requests, set k to one, n to one, and E^k to zero. E^k is the total energy consumption for the N-car system in the k-th iteration.

Step 2. Sort the move-up requests of L_u in an ascending-order list and the move-down requests of L_d in a descending-order list. Then, append the sorted list of the move-up requests to the sorted list of the move-down requests and construct a new request list Z. Set M to the number of requests in Z.

Step 3. For $1 \le n \le N$ and $1 \le m \le M$, initialize $\varphi^k_{n,m}$ to a positive constant A, and set $p^k_{n,m}$ and $E^k_{n,m}$ to zero. $p^k_{n,m}$ is the probability for the n-th car to serve the m-th request in the k-th iteration and $E^k_{n,m}$ is the energy consumption produced by the n-th car when serving the m-th request in the k-th iteration.

Step 4. Generate an N-car scheduling for the request list Z in the k-th iteration.

Step 4.1. Set m to 1 (traverse from the first element of the request list Z).

Step 4.2. For $1 \le n \le N$, calculate the energy consumption of the k-th iteration $E^k_{n,m}$ by

$$E^k_{n,m} = 2 \times \mu_{n,m} \times e_a + \sum_{s=1}^{\mu_{n,m}} [|\theta\,(n,m,s) \times \bar{q} - q| \times g \times h(n,m,s)].$$

Step 4.3. For $1 \le n \le N$, calculate probabilities $p^k_{n,m}$ by

$$p^k_{n,m} = \frac{(\varphi^k_{n,m})^\alpha \times (1/E^k_{n,m})^\beta}{\sum_{m=1}^{M} (\varphi^k_{n,m})^\alpha \times (1/E^k_{n,m})^\beta},$$

where α and β are coefficients to control the influence of $\varphi^k_{n,m}$ and $1/E^k_{n,m}$, respectively.

Step 4.4. For $1 \le n \le N$, select the car with the highest probability $p^k_{n,m}$ to serve the m-th request.

Step 4.5. Calculate the waiting time of the m-th request served by the selected car.

Step 4.6. Check if the waiting time exceeds the threshold. If so, go back to Step 4.4 and select the car with the next highest probability. Otherwise, go to Step 4.7.

Step 4.7. Check if all the requests in Z are scheduled. If so (i.e. $m = M$), go to Step 5. Otherwise, increase m by 1 and go back to Step 4.2.

Step 5. Calculate total energy consumption E^k of the scheduling derived from Step 4 by

$$\begin{aligned} E^k &= E^k_a + E^k_v = \sum_{n=1}^{N} E^k_a(n) + \sum_{n=1}^{N} E^k_v(n) \\ &= \sum_{n=1}^{N} \sum_{m=1}^{R^k_n} 2 \times \mu_{n,m} \times e_a \\ &\quad + \sum_{n=1}^{N} \sum_{m=1}^{R^k_n} \sum_{s=1}^{\mu_{n,m}} [|\theta\,(n,m,s) \times \bar{q} - q| \times g \\ &\quad \times h(n,m,s)], \end{aligned}$$

where R^k_n is the number of requests served by the n-th car in the k-th iteration.

Step 6. For $1 \le n \le N$ and $1 \le m \le M$, update all the pheromones by

$$\varphi^{k+1}_{n,m} = \rho \varphi^k_{n,m} + \Delta\varphi^k_{n,m},$$

where ρ denotes the weight in $(0, 1]$ and $\Delta\varphi^k_{n,m}$ denotes the incremental difference of $\varphi^k_{n,m}$, which is defined as follows:

$$\Delta\varphi^k_{n,m} = \begin{cases} \dfrac{C}{E^k}, & \text{if the car } n \text{ passes } \pi_{n,m} \\ 0, & \text{otherwise,} \end{cases}$$

where C is a constant.

Step 7. Check if $k = 200$. If so, terminate the process and dispatch the requests. If no, increase k by 1 and go back to Step 4.

For example, as shown in Figure 10, in the final iteration (i.e. $k = 200$), all car-choosing probabilities are calculated as follows: $p^{200}_{1,1} = \max\{p^{200}_{1,1}, p^{200}_{2,1}, p^{200}_{3,1}, p^{200}_{4,1}\}$, $p^{200}_{1,2} = max\{p^{200}_{1,2}, p^{200}_{2,2}, p^{200}_{3,2}, p^{200}_{4,2}\}$, $p^{200}_{1,6} = max\{p^{200}_{1,6}, p^{200}_{2,6}, p^{200}_{3,6}, p^{200}_{4,6}\}$, $p^{200}_{2,3} = max\{p^{200}_{2,3}, p^{200}_{1,3}, p^{200}_{3,3}, p^{200}_{4,3}\}$, $p^{200}_{2,4} = max\{p^{200}_{2,4}, p^{200}_{1,4}, p^{200}_{3,4}, p^{200}_{4,4}\}$ and $p^{200}_{2,5} = \max\{p^{200}_{2,5}, p^{200}_{1,5}, p^{200}_{3,5}, p^{200}_{4,5}\}$. Therefore, after reaching the final iteration, each car has its exclusive sequence of requests, where r_1, r_2 and r_6 are dispatched to Car 1 and r_3, r_4 and r_5 are dispatched to Car 2.

Although the convergence of the ant colony algorithm for elevator scheduling is proven in [19], it is not a real-time

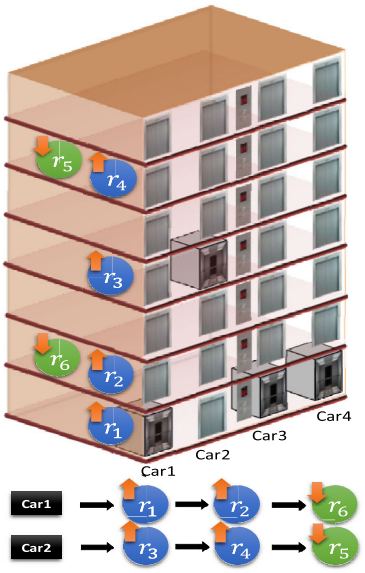

FIGURE 10. The optimized scheduling proceeded by the ant colony optimization.

scheduler because it must wait for a batch of request arrivals before it can conduct the scheduling optimization. Therefore, the ant colony algorithm may not be easily practiced in the real-time scenario with heavy passenger traffic.

ACKNOWLEDGMENT

The authors would like to thanks for Associate Editor's review handling and Reviewers' comments.

REFERENCES

[1] L.-D. Van, Y.-B. Lin, T.-H. Wu, and Y.-C. Lin, "An intelligent elevator development and management system," *IEEE Syst. J.*, to be published.
[2] G. Barney and S. D. Santos, *Elevator Traffic Analysis, Design and Control*, 2nd ed. Stevenage, U.K.: Peregrinus, 1985.
[3] B. L. Whitehall, D. J. Sirag, and B. A. Powell, "Elevator control neural network," U.S. Patent 5 672 853, Sep. 30, 1997.
[4] R. Gudwin, F. Gomide, and M. A. Netto, "A fuzzy elevator group controller with linear context adaptation," in *Proc. IEEE Int. Conf. Fuzzy Syst., IEEE World Congr. Comput. Intell.*, vol. 1, May 1998, pp. 481–486.
[5] B. L. Whitehall, T. M. Christy, and B. A. Powell, "Method for continuous learning by a neural network used in an elevator dispatching system," U.S. Patent 5 923 004, Jul. 13, 1999.
[6] M. Brand and D. Nikovski, "Optimal parking in group elevator control," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 1, Apr. 2004, pp. 1002–1008.
[7] S. Tanaka, Y. Uraguchi, and M. Araki, "Dynamic optimization of the operation of single-car elevator systems with destination Hall call registration: Part I. Formulation and simulations," *Eur. J. Oper. Res.*, vol. 167, no. 2, pp. 550–573, Dec. 2005.
[8] S. Tanaka, Y. Uraguchi, and M. Araki, "Dynamic optimization of the operation of single-car elevator systems with destination Hall call registration: Part II. The solution algorithm," *Eur. J. Oper. Res.*, vol. 167, no. 2, pp. 574–587, Dec. 2005.
[9] T. Miyamoto and S. Yamaguchi, "MceSim: A multi-car elevator simulator," *IEICE Trans. Fundamentals Electron., Commun. Comput. Sci.*, vols. E91–A, no. 11, pp. 3207–3214, Nov. 2008.
[10] A. Valdivielso and T. Miyamoto, "Multicar-elevator group control algorithm for interference prevention and optimal call allocation," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 41, no. 2, pp. 311–322, Mar. 2011.
[11] J. Fernandez, P. Cortes, J. Munuzuri, and J. Guadix, "Dynamic fuzzy logic elevator group control system with relative waiting time consideration," *IEEE Trans. Ind. Electron.*, vol. 61, no. 9, pp. 4912–4919, Sep. 2014.
[12] E. O. Tartan, H. Erdem, and A. Berkol, "Optimization of waiting and journey time in group elevator system using genetic algorithm," in *Proc. IEEE Int. Symp. Innov. Intell. Syst. Appl. (INISTA)*, Jun. 2014, pp. 361–367.
[13] H. Ishihara and S. Kato, "The effectiveness of dynamic zoning in multi-car elevator control," in *Proc. IEEE 3rd Global Conf. Consum. Electron. (GCCE)*, Oct. 2014, pp. 601–604.
[14] J. R. Fernandez and P. Cortes, "A survey of elevator group control systems for vertical transportation," *IEEE Control Syst. Magzine*, to be published.
[15] L. Marcus and A. Elias, "Impact of machine learning on elevator control strategies," KTH Roy. Inst. Technol., Stockholm, Sweden, Tech. Rep., 2015.
[16] Accessed: May 2019. [Online]. Available: https://multi.thyssenkrupp-elevator.com/en/
[17] S. Lee and H. Bahn, "An energy-aware elevator group control system," in *Proc. 3rd IEEE Int. Conf. Ind. Informat. (INDIN)*, Perth, WA, Australia, Aug. 2005, pp. 639–643.
[18] T. Zhang, S. Mabu, L. Yu, J. Zhou, X. Zhang, and K. Hirasawa, "Energy saving elevator group supervisory control system with idle cage assignment using genetic network programming," in *Proc. IEEE ICCAS-SICE*, Aug. 2009, pp. 994–999.
[19] J.-L. Zhang, J. Tang, Q. Zong, and J.-F. Li, "Energy-saving scheduling strategy for elevator group control system based on ant colony optimization," in *Proc. IEEE Youth Conf. Inf., Comput. Telecommun.*, Nov. 2010, pp. 37–40.
[20] Z. Hu, Y. Liu, Q. Su, and J. Huo, "A multi-objective genetic algorithm designed for energy saving of the elevator system with complete information," in *Proc. IEEE Int. Energy Conf.*, Dec. 2010, pp. 126–130.

LAN-DA VAN (Senior Member, IEEE) received the Ph.D. degree from National Taiwan University (NTU), Taipei, Taiwan, in 2001. Since February 2006, he has been the Faculty Member with the Department of Computer Science, National Chiao Tung University (NCTU), Hsinchu, Taiwan, where he is currently an Associate Professor. Since 2015, he has been the Deputy Director of NCTU M2M/IoT Research and Development Center. His research interests are in digital signal processing and adaptive/machine learning computation algorithms, architectures, chips, and systems and applications. He received the Best Poster Award in the iNEER Conference for Engineering Education and Research (iCEER), in 2005, and the Best Paper Award in the IEEE International Conference on Internet of Things (iThings2014), in 2014. He served as the Chairman for the IEEE NTU Student Branch, in 2000, the IEEE Award for Outstanding Leadership and Service to the IEEE NTU Student Branch, in 2001. In 2014, he was a Track Co-Chair of the 22nd IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), a Technical Track Co-Chair of the 2018 IEEE International Midwest Symposium on Circuits and Systems (MWSCAS), a Special Session Co-Chair of the 2018 IEEE International Conference on DSP, an Area Co-Chair, in 2019, the Best Paper Award Committee Member of the IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS 2019), and a Publicity Co-Chair of the 32nd IEEE International System-on-Chip Conference (SOCC 2019). In 2020, he serves as a Tutorial Co-Chair for the 33rd IEEE International System-on-Chip Conference (SOCC 2020). He served as an Associate Editor for the IEEE TRANSACTIONS ON COMPUTERS, from 2014 to 2018. He has been serving as an Associate Editor for IEEE ACCESS, since 2018, and an Associate Editor for *ACM Computing Surveys*, since 2020.

YI-BING LIN (Fellow, IEEE) received the Ph.D. degree from the University of Washington, USA, in 1990. From 1990 to 1995, he was a Research Scientist with Bellcore. He then joined National Chiao Tung University (NCTU), Taiwan, where he became a Lifetime Chair Professor, in 2010, and the Vice President, in 2011. From 2014 to 2016, he was a Deputy Minister with the Ministry of Science and Technology, Taiwan. Since 2016, he has been a coauthor of the books *Wireless and Mobile Network Architecture* (Wiley, 2001), *Wireless and Mobile All-IP Networks* (John Wiley, 2005), and *Charging for Mobile All-IP Telecommunications* (Wiley, 2008). He is an AAAS Fellow, ACM Fellow, and IET Fellow.

TSUNG-HAN WU received the B.S. and M.S. degrees in computer science from National Chiao Tung University, Hsinchu, China, in 2012 and 2014, respectively, where he is currently pursuing the Ph.D. degree. His current research interests include the Internet of Things, machine learning, and elevator scheduling.

TZU-HSIANG CHAO received the B.S. degree in computer science from National Chiao Tung University, Hsinchu, Taiwan, China, in 2018, where he is currently pursuing the M.S. degree. His current research interests include neural networks and reinforcement learning.