Accepted Manuscript - Version of Record at: https://doi.org/10.1109/TSMC.2023.3237027

A Game-Theoretic Approach to Energy-Efficient Elevator Scheduling in Smart Buildings

Erfan Farhangi Maleki, *Student Member, IEEE,* Dixit Bhatta, *Student Member, IEEE,* and Lena Mashayekhy, *Senior Member, IEEE*

Abstract—Buildings, producing more carbon footprints than the transportation sector, account for a significant portion of the United States' total energy consumption. By designing modern automation techniques, smart buildings can significantly reduce energy consumption, protect the environment, and consequently improve quality of life. This paper focuses on the automation of elevator scheduling, which is an NP-Hard problem, to reduce energy usage in smart buildings and improve users' quality of experience. We propose an optimal mathematical model for the elevator scheduling problem using integer programming. We then propose a novel game-theoretic approach that captures interactions within the elevator system to reduce energy consumption and enhance user experience. We propose a request coalition formation game, where non-overlapping coalitions of user requests are served by elevators to minimize their movements and energy consumption, while reducing service time and stops for users. We analyze the performance of our proposed approach using the optimal solution as a benchmark and Nearest Car and Fixed Sectoring algorithms as rivals. The experiments show that our approach is significantly efficient in terms of energy consumption and service time, making it suitable for smart buildings.

Index Terms—Elevator Scheduling, Smart Buildings, Cooperative Game Theory.

1 Introduction

Residential and commercial buildings account for about 40% of total energy consumption worldwide [1]. Smart buildings are therefore meant for enhancing productivity of the occupants while reducing energy consumption by automating heating/cooling, ventilation, lighting, safety controls, and in-building navigation [2]. Elevators have been pervasive means of navigation within buildings for many years now. With the advent of high-rise buildings, they have impacted lives in residences and workplaces to the extent that people spend significant amount of time simply waiting for elevators. In 2010, New York City office workers alone spent a cumulative 16.6 years waiting for elevators. Globally, more than 12 million elevators make seven billion trips and move over one billion people every day [3].

As we transition into smart buildings, serving requests as fast as possible is no longer the only optimization criterion. It is also essential to consider energy consumption for operating elevators efficiently day in day out. In general, elevator systems lose efficiency because of a bad scheduling strategy; and in fact more from it than from physical architecture or crowding [4]. Consequently, we must consider both energy and productivity implications when designing elevator scheduling strategies for smart buildings.

Scheduling elevators optimally is an NP-hard problem [5]. Consequently, there is a body of research literature available on scheduling elevators as a well-known problem. Most existing solutions are proposed using basic rule-based algorithms. They use simple heuristics and only dictate the elevator assignments and order of serving requests without considering any optimization criteria. Other heuristic approaches have been proposed to reduce service time by focusing on different aspects such as load balancing, extreme unanswered floors, and time spent in the request queue [6]. Artificial Intelligence-based and machine learning-based approaches have also been proposed with similar objectives, where they mostly rely on reinforcement learning (RL), neural networks, and genetic algorithms [7].

Many newer strategies are inspired from these known approaches, and they additionally use information gathered from Internet-of-Things (IoT) devices such as cameras and sensors as their decision parameters [8]. Only a few of them specifically cater to smart buildings by adding energy consumption as an optimization criterion. An inherent issue with these existing studies is that they rely too heavily on predictive modeling of requests and processing of IoT-generated data. As a result, processing the data (i.e., camera images or video) is a significant overhead. Moreover, these approaches require some time to adapt to the navigation patterns within buildings to be efficient enough. If any changes have to be made to the building or floor plans or if the tenants change, they have to re-adapt every time. Furthermore, any interactions or coordination within the elevator system are effectively ignored.

We consider game theory as a suitable tool for modeling the elevator scheduling problem and designing automated elevator scheduling. Game theory has been widely used in multi-agent coordination problems [9]. In task planning in multi-robot systems, a game-theoretic framework is proposed to design a solution, while the limitations of the conventional control system framework are acknowledged [9]. The same limitations apply to our problem in a smart building scenario. Other studies [10], [11] in process planning

- *E. F. Maleki, D. Bhatta, and L. Mashayekhy are with the Department of Computer and Information Sciences, University of Delaware, Newark, DE, 19716.*
  E-mail: erfanf@udel.edu, dixit@udel.edu, mlena@udel.edu
- *E. F. Maleki and D. Bhatta have contributed equally.*

Accepted Manuscript - Version of Record at: https://doi.org/10.1109/TSMC.2023.3237027

and scheduling in manufacturing process have used game theory to allow cooperation. Similarly, we also need to both plan and schedule elevators in a single cooperative process. The application of game theory has been advocated for production flow planning in manufacturing networks to capture flexibility in demands, frequent changes in the system, distributed scheduling, resource efficiency, and reduction of costs [11]. However, unlike these approaches, where agents can have both competitive and cooperative motives, an inherent difference in our problem is that non-cooperative elevators do not necessarily lead to system equilibrium or optimal schedule. For this reason, cooperative game theory enables us to capture interactions and coordination within the elevator system better, and it is our choice of solution for this problem.

In this paper, we focus on the automation of elevator scheduling for smart buildings and propose a dynamic and energy-efficient elevator scheduling approach using cooperative game theory, called Request Coalition Formation Mechanism (RCFM), that reduces users' waiting time and energy consumption. We perform extensive experiments to show that RCFM obtains efficient schedules that significantly improve energy consumption and service time. To show the robustness of RCFM in determining the elevator schedules, we perform sensitivity analysis to validate its properties and applicability in smart buildings.

1.1 Related Work

Most of the simple elevator scheduling algorithms are heuristic rule-based approaches such as Nearest Car and Fixed Sectoring algorithms [12]. Nearest Car assigns elevator calls (or requests) to the nearest elevator at the time of the request. In Fixed Sectoring, the building has an equal number of sectors and elevators, where a sector is defined as a group of contiguous floors. Elevators in each sector prefer to serve calls in that sector. Both of these algorithms are easy to implement, however, they just assume requests are served in some order and do not directly seek to specify any optimization criteria. Moreover, these approaches do not have service path planning.

A set of studies focused on elevator group control strategies. Valdivielso and Miyamoto [13] provided a group control algorithm to prevent interference for multi-car scenarios, where a single shaft contains multiple elevators. Utgoff and Connell [14] designed an algorithm to reduce the amount of time that a user spends in the system. However, these studies do not consider energy consumption and a smart building scenario. Vodopija *et al.* [15] explored the elevator group control problem using a multiobjective optimization algorithm (MOA) considering minimizing two objectives, i.e., the proportion of floors with waiting passengers and the proportion of elevator stops, with constraints on the maximum number of elevator skips. However, they used a state transition table, causing scalability issues for large-scale scenarios. In addition, constraining the maximum elevator skips may not optimize the overall service path of the elevators.

A few studies built optimization models to solve the elevator scheduling problem. Zhang *et al.* [16] proposed an energy-saving scheduling optimization method to select elevators to serve according to minimum scheduling energy criteria. However, their method is restricted to only up-peak traffic. Wu *et al.* [17] formulated the problem of optimizing passenger-to-car assignment and car routing as mixed-integer linear programming (MILP) to minimize the average waiting time of all passengers. Xu *et al.* [18] modeled a single elevator scheduling problem as MILP. However, these studies do not propose an algorithmic solution and only depend on a commercial integer programming solver.

TABLE 1: Comparison with existing research

Study	Energy Consumption	Service Time	Different Req. Arrivals	Stop Control	Load Balancing	User Data Independence	Method
Crites *et al.* [19]		✓	✓	✓			RL
Kim *et al.* [20]	✓		✓				Fuzzy Theory
Kwon *et al.* [21]	✓		✓				Rule-based
Van *et al.* [8]	✓	✓	✓			✓	Rule-based
Vodopija *et al.* [15]	✓		✓	✓		✓	MOA
Zhang *et al.* [16]	✓		✓			✓	Rule-based
Wu *et al.* [17]		✓	✓			✓	MILP
Xu *et al.* [18]		✓	✓			✓	MILP
Wang *et al.* [22]		✓	✓				DL
Zhang *et al.* [23]	✓		✓			✓	Greedy
Our Study	✓	✓	✓	✓	✓	✓	Game Theory

More complex algorithms use different optimization criteria to make selection decisions and are generally based on artificial intelligence and machine learning concepts such as reinforcement learning (RL), deep learning (DL), genetic algorithm, and heuristics. The application of RL to the elevator dispatching problem has been discussed by Crites and Barto [19]. They proposed a team of learning agents, each of which is responsible for controlling one elevator. Wang *et al.* [22] proposed real-time occupancy-aware dispatching for the elevator group control, in which occupancy awareness is implemented via a convolutional-neural-network-based object detection to estimate elevator capacity. Kim *et al.* [20] used fuzzy logic to reduce waiting time and power consumption using passenger traffic and current system parameters. However, these studies heavily rely on traffic patterns and user data, or they need a dedicated task and data management module. In addition, they do not plan how the elevators need to travel to serve the requests.

Considering specific energy saving studies for group elevator control, Zhang *et al.* [23] proposed energy-efficient elevator scheduling based on ant-colony optimization. However, this approach is not suitable under heavy passenger traffic. Moreover, ant-colony optimization is known to suffer from premature convergence and output stagnation. Van *et al.* [8] proposed an energy saving green scheduling algorithm that reduces elevator car moving steps by globally searching for the nearest elevator to serve each incoming request. The global search for the nearest car as well as only 8 elevators in performance evaluation of their approach indicate scalability issues.

Research in the domain of smart buildings has attracted a great deal of attention in the past few years. Cui *et*

Accepted Manuscript - Version of Record at: https://doi.org/10.1109/TSMC.2023.3237027

al. [24] proposed a peer-to-peer energy sharing approach among smart buildings based on distributed transactions. Zarikas and Tursynbek [25] used Bayesian network formulation and pre-defined fuzzy rules to drive schedules of elevators. Bapin and Zarikas [26] extended this work by including a probabilistic decision model to reduce waiting time. Similar to our work, Kwon *et al.* [21] considered both waiting time and energy efficiency. However, their approach relies heavily on sensor data for making predictions, and thus any changes in the building or user behaviors will make the approach inefficient. Moreover, their elevator call procedures are simply rule-based.

Regarding the application of game theory, Ramalingam *et al.* [27] proposed a method for group elevator scheduling based on submodularity, a concept in game theory. However, their approach itself is not based on game theory. The closest mention of game theory in solving the elevator scheduling problem is in the survey paper by Koehler and Ottiger [6], where they discussed a simple auction model (without any implementation details) for allocating passengers to elevators. However, value gained from serving a request may change extremely quickly in an auction-based elevator system and desirable economic properties do not offer performance gains.

This paper presents a novel approach to elevator scheduling by tackling the challenges in the existing work. We show a detailed comparison of our work with the most relevant studies based on several criteria in Table 1.

1.2 Our Contribution

We first propose an optimal mathematical model for the Elevator Scheduling Problem (ESP) using integer programming, which we use as a performance benchmark in our experiments. We then propose a coalitional game to model the elevator system based on a unique preference relation to reduce the overall movement and service time. We design a computationally-efficient and dynamic mechanism, called Request Coalition Formation Mechanism (RCFM), that finds a stable request coalition structure for our game. RCFM captures the overall elevator system behavior under different request patterns and efficiently shares the load among elevators to find suitable schedules efficiently. To the best of our knowledge, this is the first study proposing a coalitional game and a dynamic mechanism for elevator scheduling in smart buildings considering coordination among elevators in serving the requests.

1.3 Organization

The rest of the paper is organized as follows. In Section 2, we describe the elevator system model, our optimal mathematical model for ESP, and our proposed coalitional game for elevator scheduling. In Section 3, we present our proposed mechanism to solve the request coalition formation game efficiently. In Section 4, we evaluate the proposed mechanism by extensive experiments. In Section 5, we summarize our results and present possible directions for future research.

2 Elevator Scheduling Framework

This section describes the elevator system model, our optimal mathematical formulation, and our proposed coalitional game.

2.1 System Model

The system consists of smart elevators serving users. The finite set of smart elevators in the system is denoted by E. Each elevator $e_i \in E$ has a property l_i indicating its start floor (initial location before it moves). We use L to represent the location set denoting the start floors of the elevators. The number of elevators at each start floor $l_m \in L$ is denoted by f_m.

The set of user requests is denoted by R. Each request $r_j \in R$ represents the current floor number of a user who initiates an elevator call. With smart elevators, traditional landing call buttons have been eliminated. Instead, users enter their destination floor using their smart devices or a touch-screen input device during an elevator call. The destination floors are denoted by D. Since one request floor can have users with multiple destinations, the request and destination associations $(r_j \in R, d_k \in D)$ are maintained as ordered pairs in the collection set S. The union of all floors that must be covered in the system is denoted by Z (i.e., union of R, D, and L).

There is a cost associated with visiting each request floor and destination floor from any other floor. We formulate the cost in this problem in terms of movements to capture the notion of energy awareness. Reducing elevator movements directly leads to a decrease in energy consumption [28]. As a result, we define the travel cost as a function of the distance covered by the elevators. We denote this travel cost based on the distance between two arbitrary floors i and j as b_{ij}, while capturing the changes in elevator speeds during travel between floors using function $\mathcal{P}$. Note that the costs are asymmetric, i.e., $b_{ij} \neq b_{ji}$, $(0 < b_{ij} < \infty)$ (for example, traveling upward can consume more energy than downward). We also use p to indicate the maximum number of stops that can be made by each elevator to capture the notion of service time. Limiting the stops with p prevents each elevator from stopping too much thereby improving the overall service time.

The elevators travel from floor to floor and stop to serve requests and destinations (floors). The stopping floors made by an elevator is defined as its *service path* or *schedule*. The overall path covered by the elevators serving all requests and destinations is defined as the overall service path. We define the Elevator Scheduling Problem (ESP) as the problem of finding groups of users each to be served by an elevator and determine the elevator schedule with minimum cost. Solving ESP, any elevator serves only certain floors based on its allocated requests and destinations.

To model ESP optimally, we use the Multiple-Depot multiple-Traveling Salesman Problem (MD-mTSP) as a basis, which is a variant of standard Traveling Salesman Problem [29]. The multiple traveling salesman problem (mTSP) is a generalization of the well-known traveling salesman problem (TSP) in which each city must be visited by exactly one traveling salesman. When there is a single depot for all the salesmen, the problem is called the single depot mTSP. When the salesmen are initially at different depots, the problem is referred to as the multi-depot mTSP (MD-mTSP). We use MD-mTSP due to the similarity of finding a tour (overall service path) of target cities (requests and destinations) from multiple depots (elevator start floors).

Accepted Manuscript - Version of Record at: https://doi.org/10.1109/TSMC.2023.3237027

However, it should be noted that our system model has markedly different assumptions, constraints, and travel cost matrix compared to a standard MD-mTSP formulation.

Theorem 1. Multiple-Depot multiple-Traveling Salesman Problem (MD-mTSP) is polynomial-time reducible to the Elevator Scheduling Problem (ESP).

Proof: Given an instance of MD-mTSP = $(\mathcal{D}, \mathcal{T}, c_{ij})$, where $\mathcal{D}$ is the set of depots, $\mathcal{T}$ is the set of target cities/nodes, and c_{ij} is the cost of travel between node i and node j. Likewise, an instance of ESP = $(L, R \cup D, b_{ij})$, where L is the set of elevator start floors, $R \cup D$ is the set of requests and destinations, and b_{ij} is the travel cost between floor i and floor j. We can linearly map $\mathcal{D}$ to L, $\mathcal{T}$ to $R \cup D$, and polynomially reduce c_{ij} to b_{ij}. Therefore, MD-mTSP is polynomial-time reducible to ESP. □

Theorem 1 proves the NP-hardness of ESP because the cost matrix c_{ij} can be transformed to travel cost matrix b_{ij} in polynomial time using the following rules for any elevator initially at floor $j \in L$:

1) $b_{ik} = \infty$ $\quad \forall i, k \in Z : i = k$
2) $b_{ij} = 0$ $\quad \forall i \in R$
3) $b_{kj} = 0$ $\quad \forall k \in D$
4) $b_{ij} = \infty$ $\quad \forall i \in L$
5) $b_{jk} = \infty$ $\quad \forall k \in D$
6) $b_{ik} = \mathcal{P}(c_{ik})$ $\quad \forall (i, k) \in S$
7) $b_{ki} = \infty$ $\quad \forall (i, k) \in S$
8) $b_{ik} = \mathcal{P}(c_{ik})$ $\quad$ *for all other* $i, k \in Z$

Our defined rules present some of the differences between ESP and MD-mTSP. Rule (1) means there are no reflexive paths. The path from a floor to itself should never be selected. Rules (2) and (3) show that the cost of returning back to the initial elevator floor j from any floor is zero. In other words, the elevators do not need to return back to their initial floor after serving their final requests. Rule (4) shows that elevators do not cover another elevator's start floor. Rule (5) shows that elevators do not directly visit destination floors. The path eventually reaches there from a request floor. Rule (6) shows that there is a path from a request floor to a destination floor. Rule (7) shows that elevators do not stop again at the request floor after leaving for the destination. Rule (8) shows that all other costs between two floors i and k are simply the function of the cost of travel between corresponding nodes in MD-mTSP (i.e., $\mathcal{P}(c_{ik})$). In the next subsection, we present the optimal formulation of ESP considering these rules.

2.2 ESP Formulation

We now formulate ESP optimally as an Integer Program, called IP-ESP, based on a reduction from MD-mTSP. IP-ESP is time-dependent, defined for every time window t. For the sake of simplicity and readability of the paper, we eliminate time (t) from the formulation. IP-ESP is offline, meaning that the elevator system parameters (e.g., requests, destinations, and elevators) are deterministic (not random) and known during optimization time window. However, it does not know the requests coming in during the next time window.

We define binary decision variable x_{ij}^t (represented as x_{ij}) that becomes 1 if an elevator stops at two floors i and j with a direct path between them in its service path during time window t. We formulate IP-ESP(E, R) as a deterministic optimization problem as follows:

$$\Theta = min \sum_{i \in Z} \sum_{j \in Z} b_{ij} x_{ij} \tag{1}$$

Subject to:

$$\sum_{j \in R} x_{ij} \leq f_i \qquad \forall i \in L \tag{2}$$

$$\sum_{j \in R} x_{ji} \leq f_i \qquad \forall i \in L \tag{3}$$

$$\sum_{i \in Z} x_{ij} = 1 \qquad \forall j \in R \tag{4}$$

$$\sum_{i \in Z} x_{ij} = 1 \qquad \forall j \in D \tag{5}$$

$$\sum_{j \in Z} x_{ij} = 1 \qquad \forall i \in D \tag{6}$$

$$\sum_{i \in Z} x_{ij} = \sum_{i \in Z} x_{ji} \qquad \forall j \in R \tag{7}$$

$$+ \textit{ Subtour Elimination Constraints} \tag{8}$$

$$+ \textit{ Ordering Constraints} \tag{9}$$

$$x_{ij} = \{0, 1\} \qquad \forall i, j \in Z \tag{10}$$

The objective (1) of our IP-ESP(E, R) is to minimize the total travel cost of the elevators in E while serving all the requests in R. Constraints (2) and (3) guarantee that the number of elevators leaving and returning to every start floor must be less than or equal to the number of elevators initially at that floor. These constraints are defined to allow a subset (or all) of elevators available in the system to serve the requests. Constraints (4) guarantee that each request must be served by one elevator. Constraints (5) and (6) ensure that each destination floor is visited by exactly one elevator. Constraints (7) ensure that the number of incoming elevators at any request floor is equal to the number of outgoing elevators. Constraints (8) are subtour elimination constraints to avoid any shortcuts that lead to sub-optimal solutions. Constraints (9) are to ensure ordering for requests and destinations. We will provide more details about these constraints. Constraints (10) are integrality constraints ensuring the decision variables x_{ij} are either 0 or 1.

For optimal formulation, constraints (8) can be based on subsets proposed by Dantzig *et al.* [30], as presented in Equation 11:

$$\sum_{i \in C} \sum_{j \in C} x_{ij} \leq |C| - 1 \qquad \forall C \subseteq Z \setminus L, C \neq \emptyset \tag{11}$$

However, the number of subtours (and consequently the number of these constraints) increases exponentially with the increase in the number of requests. As a result, this approach is not suitable for solving IP-ESP. Hence, we replace constraints (8) with subtour elimination constraints introduced by Miller-Tucker-Zemlin, called MTZ-SECs. MTZ-SECs introduce $O(n^2)$ additional continuous variables and

Accepted Manuscript - Version of Record at: https://doi.org/10.1109/TSMC.2023.3237027

eliminate the exponential expansion problem [31]. We define the subtour elimination constraints of IP-ESP based on MTZ-SECs as follows:

$$h_i - h_j + p\,x_{ij} \leq p - 1 \qquad \forall i, j \in Z \setminus L : i \neq j \quad (12)$$

The new constraints (12) are used to enable the ordering of the requests in which they are served. This is done by introducing dummy decision variables. The value of h_j gives the order in which request r_j is served by an elevator. The constraints also enforce p, which we defined earlier in our model. The value of p is an input to limit the number of stops that can be made by an individual elevator in its service path.

To guarantee request floors are visited before their corresponding destination floors by the same elevator, we based Constraints (9) on new constraints (13) to (17) in our model.

$$h_a - h_i + 1 <= 0 \qquad \forall (a, i) \in S \quad (13)$$
$$u_i - i = 0 \qquad \forall i \in L \quad (14)$$
$$u_i - u_j + x_{ij}\,M \leq M \qquad \forall i \in Z,\, j \in Z \setminus L : i \neq j \quad (15)$$
$$u_j - u_i + x_{ij}\,M \leq M \qquad \forall i \in Z,\, j \in Z \setminus L : i \neq j \quad (16)$$
$$u_a - u_i = 0 \qquad \forall (a, i) \in S \quad (17)$$

Constraints (13) ensure that for a request floor a and its corresponding destination floor i (i.e., $(a, i) \in S$), the visit order of the request floor, denoted by h_a, is prior to the visit order of the destination, denoted by h_i. To ensure request floors and their corresponding destination floors are visited by the same elevator, we define another set of dummy decision variables for labeling. The value of u_i plays as a *label* for the elevator visiting floor $i \in Z$ in its service path. Constraints (14) ensure that the elevator's index (i.e., i) is assigned as its label. Therefore, every floor visited by this elevator must have the same label. Constraints (15) and (16) ensure that if there is a path between floor i and floor j (i.e., $x_{ij} = 1$), they have the same label. In other words, these floors are visited by the same elevator. To define these constraints, we use M, which is $\max_{i,j \in L} |u_i - u_j|$. The value of M ensures that the difference between the labels of two floors cannot be larger than the maximum difference between the labels of elevators. Having $x_{ij} = 1$, constraints (15) ensure $u_i \leq u_j$ and constraints (16) ensure $u_j \leq u_i$, and together they guarantee the labels are the same ($u_i = u_j$). Constraints (17) ensure the request floor and its corresponding destination floor must have the same label (must be visited by the same elevator). Therefore, constraints (13)-(17) guarantee that request floors are visited before their destination floors by the same elevator in the service path.

IP-ESP(E, R) finds the optimal schedule with a fixed number of elevators. However, this problem is computationally infeasible since it is NP-hard. Moreover, it is possible that only a subset of elevators can actually serve all requests at a lower cost (see Theorem 2). Enumerating and solving IP-ESP for all possible subsets of elevators is not scalable. As a result, we introduce our coalitional game next to solve ESP efficiently.

2.3 Coalitional Game Formulation For Elevator Scheduling

Our approach comprises a coalitional game with transferable utility. Coalitional game theory studies the interactions between groups of decision-makers (i.e., players) [32]. In coalitional games, players can cooperate and form alliances while ultimately trying to maximize utility.

We define the *Request Coalition Game* (R, g), where R is the set of requests as players in the game, and g is the characteristic cost function defined over all possible request coalitions $C \subseteq R$. A *request coalition* C is a set of requests in R that form a group and are served by the same elevator. From the coalitional game-theoretic perspective, the characteristic cost function is the cost incurred when requests are grouped in one coalition. This function is a real-valued function such that $g : C \to \mathbb{R}^+$ and $g(\emptyset) = 0$. We define the characteristic cost function of our proposed game as follows:

$$g(e_c \leftarrow C) = \begin{cases} \infty & \text{IP-ESP}(e_c, C) \text{ is not feasible} \\ \Theta = \text{IP-ESP}(e_c, C) & \text{otherwise.} \end{cases} \quad (18)$$

That means for a coalition C that is assigned to elevator $e_c \in E$, the cost of serving its requests is Θ, which is calculated by calling IP-ESP with elevator e_c and coalition C. The cost obtained by IP-ESP is defined in terms of the travel cost incurred by the elevator serving its requests. The cost of a coalition is infinity if IP-ESP is not feasible. Based on our game, the objective of each request is to be a member of a coalition with minimum cost.

In game theory, a solution concept (e.g., Nash equilibrium in non-cooperative games) is a systematic description of how the game will be played by employing the best possible strategies and what the outcomes might be. A solution concept of a coalitional game is an outcome of the game defined as a set of cost allocations representing acceptable distributions of the cost among players, satisfying two main properties: *fairness* and *stability* [33].

A cost allocation is a division of the request coalition's cost to all participating requests. To ensure fairness in our game, cost sharing among requests is based on dividing the cost $g(e_c \leftarrow C)$ equally among the participating requests in C. This is due to the fact that scheduling each request plays a critical role in minimizing the overall cost of the coalition. We define $\psi_{r_i}(e_c \leftarrow C)$ as the *allocated* or *shared cost* of request r_i when is a member of coalition C served by elevator e_c. We simply use a cost vector to represent each request's cost share. For the grand request coalition (the coalition with all requests R served by any elevator e_r), a cost vector is defined as: $\Psi(e_r \leftarrow R) = (\psi_{r_1}(e_r \leftarrow R), \ldots, \psi_{r_v}(e_r \leftarrow R))$ (v is the total number of requests), with the understanding that each component of the vector is the allocated cost to serve its corresponding request.

The most prominent solution concept that formalizes the notion of stability in coalitional games is *the Core*. An outcome (a cost allocation) is in the Core if no subset of players has an incentive to leave the grand coalition and form a coalition of their own. To define the Core, we first need to introduce the concept of imputation as follows:

Definition 1 (Imputation). An imputation is a cost vector $(\psi_{r_1}(e_r \leftarrow R), \ldots, \psi_{r_v}(e_r \leftarrow R))$ satisfying two conditions:

(i) $\psi_{r_i}(e_r \leftarrow R) \leq g(e_c \leftarrow \{r_i\}), \forall r_i \in R$, and
(ii) $\sum_{r_i \in R} \psi_{r_i}(e_r \leftarrow R) = g(e_r \leftarrow R)$.

Accepted Manuscript - Version of Record at: https://doi.org/10.1109/TSMC.2023.3237027

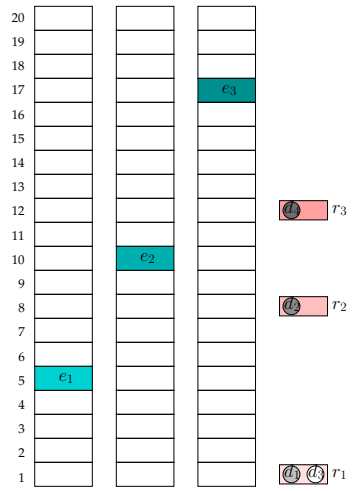


Fig. 1: Example: Elevators and Requests

Condition (i) guarantees that the cost obtained by each request r_i participating in the grand request coalition served by e_r is not greater than its cost obtained by acting alone, when it is served by $e_c \in E$. Condition (ii) ensures that the entire cost of the grand request coalition is divided among all requests.

Based on the definition of the imputation, we define the Core of our game as follows:

Definition 2 (Core). The Core of the Request Coalition Game is a set of imputations satisfying:
$\sum_{r_i \in C} \psi_{r_i}(e_r \leftarrow R) \leq g(e_c \leftarrow C), \forall C \subseteq R.$

If the Core exists, it ensures that the cost of any request coalition C served by e_c, i.e., $g(e_c \leftarrow C)$, is not less than the sum of the corresponding allocated costs of the participating requests in the grand request coalition, served by e_r. A cost vector is in the Core if there is no incentive for any subset of requests to leave the grand request coalition and form a smaller request coalition with a lower cost to serve the requests. However, we prove that the Core of the proposed game may be empty, meaning that the grand request coalition is not stable to be formed and serve all the requests with the minimum cost.

Theorem 2. The Core of the Request Coalition Game (R, g) may be empty.

Proof: The proof is by a counterexample. The scenario is depicted in Figure 1. Let consider a building with 20 floors and 3 elevators; at time t, the elevators e_1, e_2 and e_3 are at floors $l_1 = 5$, $l_2 = 10$, and $l_3 = 17$, respectively. In addition, the request set $R = \{1, 8, 12\}$, the destination set $D = \{3, 5, 6, 14\}$, and the collection set $S = \{(1,3),(1,6),(8,5),(12,14)\}$. Also, p is set to 4 to avoid more than 4 stops by an elevator. The cost of each coalition is calculated based on the total travel cost incurred by the serving elevator moving up and down. For example, $e_1 \leftarrow \{r_3, d_1, d_3\}$ means e_2 is in charge of serving request r_1 (floor 1) to destinations d_1 and d_3 (floors 3 and 6). To this aim, e_1 first requires to move down to the request's location in floor 1 from its current place and then move up as it serves its two destination floors. If we assume a travel cost function $b_{ij} = c_{ij}^{1:1} + \gamma$, where $c_{ij} \geq 0$ represents the floor distance and $\gamma = \{1 \text{ for down, } 2 \text{ for up}\}$ is a constant, it exhibits a travel cost $b_{ij} = 13.47$.

TABLE 2: Costs for different Request Coalitions

Coalition C & elevator assignment	**Cost $g(e_c \leftarrow C)$**
$e_1 \leftarrow \{r_2, d_2, r_1, d_1, d_3, r_3, d_4\}$	33.65
$\boldsymbol{e_2 \leftarrow \{r_3, d_4, r_2, d_2, r_1, d_1, d_3\}}$	**32.27**
$e_3 \leftarrow \{r_3, d_4, r_2, d_2, r_1, d_1, d_3\}$	36.69
$e_1 \leftarrow \{r_2, d_2, r_1, d_1, d_3\}$	22.73
$e_2 \leftarrow \{r_3, d_4\}$	6.59
$e_1 \leftarrow \{r_2, d_2, r_1, d_1, d_3\}$	22.73
$e_3 \leftarrow \{r_3, d_4\}$	11.02
$e_2 \leftarrow \{r_2, d_2, r_1, d_1, d_3\}$	20.08
$e_3 \leftarrow \{r_3, d_4\}$	11.02
$e_1 \leftarrow \{r_1, d_1, d_3\}$	13.47
$e_2 \leftarrow \{r_2, d_2\}$	6.87
$e_3 \leftarrow \{r_3, d_4\}$	11.02

The optimal cost for each request coalition is shown in Table 2. The grand request coalition is when all requests are served by one elevator with the minimum cost among possible choices. In this example, $e_2 \leftarrow \{r_3, d_4, r_2, d_2, r_1, d_1, d_3\}$ has the minimum cost of 32.27 for serving all requests and is selected as the grand coalition. However, there are two request coalitions $e_1 \leftarrow \{r_2, d_2, r_1, d_1, d_3\}$ and $e_2 \leftarrow \{r_3, d_4\}$ with the minimum total cost of 29.32 for serving all the requests. Since the grand request coalition has a higher cost than the sum of the smaller request coalitions of all requests, these requests do have incentives to deviate from forming the grand request coalition. Therefore, they break the grand request coalition to form coalitions of their own. Consequently, the grand request coalition is not stable, and the Core is empty. □

Since the Core may be empty, the grand request coalition does not form, and smaller disjoint coalitions of requests will form. The partitions of an unstable grand coalition are called a *coalition structure*. In the next section, we design a Request Coalition Formation Game and Mechanism to obtain a stable coalition structure, where each coalition is served by an elevator.

3 Request Coalition Formation Game and Mechanism

In this section, we propose our Request Coalition Formation Game in coalition structure form, introduce our novel mechanism, and describe its properties.

3.1 Concepts and Models

A request coalition structure $\Pi = \{C_1, \ldots, C_{|E|}\}$ is a partition of the set of requests R to non-overlapping coalitions such that each request is a member of exactly one coalition and each coalition is served by an elevator. That is, $C_m \cap C_n = \emptyset, \forall C_m, C_n \in \Pi$ and $\bigcup_{C_n \in \Pi} C_n = R$. We denote by $\mathbf{\Pi}$ the set of all coalition structures. Finding the optimal coalition structure in $\mathbf{\Pi}$ is NP-complete. The objective is to obtain a stable request coalition structure, where requests do not have incentives to join other coalitions to achieve a lower cost. Therefore, each request must be able to compare the request coalitions and indicate which one is more preferred to be a member of. We define the concept of a preference relation as follows:

Definition 3 (Preference relation). For any request $r_i \in R$, a preference relation $\succeq_i$ is defined as a complete, reflexive,

Accepted Manuscript - Version of Record at: https://doi.org/10.1109/TSMC.2023.3237027

and transitive binary relation on $\mathbf{\Pi}_i$, where $\mathbf{\Pi}_i \subset \mathbf{\Pi}$ is the set of all possible request coalitions containing request r_i.

Meaning that, for a request $r_i \in R$ that is a member of two request coalition C and C' ($\forall C, C' \in \mathbf{\Pi}_i$), $C \succeq_i C'$ indicates that request r_i prefers to be member of coalition C served by $e_c \in E$, over being a member of coalition C' served by $e_{c'} \in E$ (or at least, prefers both equally). We define the preference relations of any request $r_i \in R$ as follows:

$$C \succeq_i C' \Leftrightarrow \phi_i(e_c \leftarrow C) \leq \phi_i(e_{c'} \leftarrow C'), \tag{19}$$

where ϕ_i is a preference function defined for any request $r_i \in R$ as follows:

$$\phi_i(e_c \leftarrow C) = \begin{cases} \psi_{r_i}(e_c \leftarrow C) & \text{if } C \in \mathbf{\Pi}_i, C \notin H(i) \\ \infty & \text{otherwise.} \end{cases} \tag{20}$$

Note that $\psi_{r_i}(e_c \leftarrow C)$ represents the allocated or shared cost of request r_i when it is a member of the request coalition C served by elevator e_c. In addition, $H(i)$ is the history set of r_i, which contains all the coalitions that r_i has visited in the previous coalition structures. The function ϕ_i implies that any request $r_i \in C$ receives a weight equal to its allocated cost (when it is in coalition C and served by elevator e_c) if coalition C is not in the history of request r_i. Otherwise, r_i receives a weight of infinity because the request r_i has no incentive to revisit previously visited coalitions due to an increase in its allocated cost. Using the preference relation, each request can evaluate its preferences over the possible coalitions that it can join.

Given the preference relation $\succeq_i$ for every request $r_i \in R$, we defined the request coalition formation game, as a type of hedonic game [34] as follows:

Definition 4 (Request Coalition Formation Game). *The Request Coalition Formation Game* is a hedonic game with a pair (R, $\succeq$), such that R is a set of requests and $\succeq$ is a set of preference relations ($\succeq_1$, ..., $\succeq_R$) for all requests $r_i \in R$.

Our goal is to design an algorithm that allows the requests to make coalition formation decisions. We first define a coalition indicator and then a switch rule that enables requests to decide which coalitions to join.

Definition 5 (Coalition Indicator). Assuming a partition Π of R, the notion $C_\Pi(r_i)$ denotes the coalition that request $r_i \in R$ belongs to, i.e., $C_\Pi(r_i) = C_k$, where $C_k \in \Pi$ and $r_i \in C_k$.

We define the *switch rule* for the requests to follow in forming the request coalitions.

Definition 6 (Switch Rule). Given a partition $\Pi = \{C_1, \ldots, C_u\}$ of the set of requests R, a request r_i decides to leave its current coalition $C_\Pi(r_i) = C_m$, where $C_m \in \Pi$, and to join another coalition $C_n \in \Pi$, where $C_n \neq C_\Pi(r_i)$, if and only if $C_n \cup \{r_i\} \succeq_i C_\Pi(r_i)$.

Therefore, by switch rule, $\{C_m, C_n\}$ in Π changes to $\{C_m \setminus \{r_i\}, C_n \cup \{r_i\}\}$ in a new coalition structure Π'. That is, a single switch rule transforms a partition Π to $\Pi' = \Pi \setminus \{C_m, C_n\} \cup \{C_m \setminus \{r_i\}, C_n \cup \{r_i\}\}$. This rule enables any request r_i to leave its current coalition $C_\Pi(r_i)$ and to join another coalition $C_n \in \Pi$ if the new coalition is preferred (or at least with the same preference) over its current coalition $C_\Pi(r_i)$ using the preference relation defined in (19). In this transition, $C_\Pi(r_i) = C_m$ is stored in r_i's history set $H(i)$.

Algorithm 1 Request Coalition Formation Mechanism (RCFM)

1: **Input:** Request-Destination Set S, Elevator Set E, Control Parameter p
2: **for** $e_k \in E$ **do**
3: $C_k = \emptyset$ /* coalition to be served by elevator e_k */
4: **for** $r_i \in R$ **do**
5: $H(i) = \emptyset$
6: $e_j \leftarrow$ Select a random elevator from E
7: $C_j = C_j \cup \{r_i\}$ /* add to the coalition served by e_j */
8: **for** $e_k \in E$ **do**
9: $(\mathcal{E}, \zeta) = \mathit{BuildPath}(e_k, C_k)$
10: $\eta = \sum_{(u,v)\in\mathcal{E}} 1 \,|\, b_{uv} \neq 0$
11: **if** $\eta \leq p$ **then**
12: $g(e_k \leftarrow C_k) = \zeta$
13: **else**
14: $g(e_k \leftarrow C_k) = \infty$
15: **for** $r_i \in C_k$ **do**
16: $\phi_i(e_k \leftarrow C_k) = \frac{g(e_k \leftarrow C_k)}{|C_k|}$
17: $\Pi_0 = \{e_1 \leftarrow C_1, e_2 \leftarrow C_2, \ldots, e_{|E|} \leftarrow C_{|E|}\}$ /*initial partition*/
18: **repeat**
19: **for** $r_i \in R$ considering partition Π **do**
20: $C = C_\Pi(r_i)$ /*current coalition*/
21: $C^* = C_\Pi(r_i)$ /*The most preferred coalition*/
22: **for** $C_n \in \Pi, C_n \neq C, C_n \notin H(i)$ and $g(e_n \leftarrow C_n) \neq \infty$ **do**
23: $(\mathcal{E}, \zeta) = \mathit{BuildPath}(e_n, C_n \cup \{r_i\})$
24: $\eta = \sum_{(u,v)\in\mathcal{E}} 1 \,|\, b_{uv} \neq 0$
25: **if** $\eta \leq p$ **then**
26: $g(e_n \leftarrow C_n \cup \{r_i\}) = \zeta$
27: $\psi_i(e_n \leftarrow C_n \cup \{r_i\}) = \frac{g(e_n \leftarrow C_n \cup \{r_i\})}{|C_n \cup \{r_i\}|}$
28: **if** $C_n \cup \{r_i\} \succeq_i C^*$ **then**
29: $C^* = C_n \cup \{r_i\}$
30: **if** $C^* \neq C$ **then**
31: $H(i) = H(i) \cup C$
32: $C = C \setminus \{r_i\}$
33: $C^* = C^* \cup \{r_i\}$
34: **until** Convergence to a final stable Π_f
35: **Output:** Π_f and elevator assignments

Next, we introduce our proposed request coalition formation mechanism using the switch rule.

3.2 Request Coalition Formation Mechanism

The *Request Coalition Formation Mechanism* (RCFM), presented in Algorithm 1, is an iterative approach using switch rule to find a stable request coalition structure, where each formed coalition in the obtained structure is served by an elevator.

RCFM is time-dependent and dynamically finds a stable request coalition structure for one instance of the elevator scheduling during the time window t. Notation t is eliminated for simplicity and readability. RCFM uses the request-destination set S, the elevator set E, and the control parameter p as inputs. This control parameter allows at most p stops by any elevator.

Initially, $|E|$ empty coalitions are created, each assigned to an elevator (lines 2-3). The algorithm initials the history

Accepted Manuscript - Version of Record at: https://doi.org/10.1109/TSMC.2023.3237027

set of each request to an empty set (line 5) and then allocates this request randomly to an existing coalition (lines 6-7).

For each elevator e_k serving a request coalition C_k, RCFM then calls *BuildPath* to find the service path of the elevator. This function, described in Section 3.3, returns $(\mathcal{E}, \zeta)$, where $\mathcal{E}$ is the obtained service path and ζ is its cost. RCFM then finds the number of stops (η) needed in the service path to serve all requests in C_k. This value is calculated by finding the number of edges with positive weights in the obtained path (line 10). RCFM sets the characteristic function of coalition C_k to the obtained cost only if η is less than or equal to p to satisfy the maximum stop requirement of the coalition (lines 11-12). Otherwise, it is set to infinity (lines 13-14). Then, RCFM calculates the allocated or shared cost for each request $r_i \in C_k$ by dividing the total cost equally among the coalition's requests (lines 15-16). After this step, RCFM forms Π_0, as an initial request coalition structure, with coalitions that each mapped to an elevator (line 17). For the example shown in Fig 1, one possible initial coalition structure is $\Pi_0 = \{e_1 \leftarrow \{r_1, r_2\}, e_2 \leftarrow \{r_3\}, e_3 \leftarrow \emptyset\}$.

RCFM then iteratively improves the request coalition structure until it converges a stable coalition structure Π_f (lines 18-34). For any request r_i, currently in coalition C, RCFM investigates a possible switch using the preference relation in equation (19). The most preferred coalition of request r_i is represented by C^* (line 21).

The algorithm first selects a candidate coalition C_n for the switch operation that differs from the current coalition of the request, never visited, and has a finite cost (line 22). RCFM finds the path and its corresponding cost assuming the request joins the candidate coalition and served by its assigned elevator e_n (line 23). It then calculates η that indicates the new number of stops made by elevator e_n when serving r_i (line 24). If η satisfies the number of stops restriction, then coalition C_n can check whether to accept r_i as a new member (lines 26-29). RCFM calculates the allocated cost of request r_i (lines 26-27). Then, RCFM updates C^* if the candidate coalition C_n is more preferred to serve the request (lines 28-29). After exploring all candidate coalitions, if C^* is different than the current coalition C, RCFM performs the switch operation. In such a case: i) the request's history is updated by adding C to $H(i)$ (line 31); ii) the request leaves its current coalition C (line 32); and iii) the request joins the new coalition C^* (line 33).

After RCFM converges, it returns the final coalition structure Π_f and the elevator assignments and schedules (line 34).

3.3 Finding Path

In this section, we describe our method to find a feasible cost-efficient path for an elevator to serve all of the requests in its coalition. *BuildPath* shown in Algorithm 3 is a function that finds such a path and its cost.

BuildPath models the problem as a weighted graph. It calls *Buildgraph* function, presented in Algorithm 2, to build a graph for an elevator and its serving request coalition, where nodes represent request floors, their destination floors, and the elevator's start floor; and edges represent a possible direct path between two floors. *Buildgraph* starts with an empty graph with all the nodes and without any

Algorithm 2 BuildGraph(Elevator e, Coalition C)

1: Create Graph $G(V, \mathcal{E})$, where $\forall\, v_m \in V : \{\, v_m \in \{C \cup \{e\} \cup \{v_m \in D \mid v_n \in C \cup \{e\}, (v_n, v_m) \in S\}\}\}$ and $|\mathcal{E}| = 0$
2: **for** $v_n \in V$ **do**
3: **for** $v_m \in V$ **do**
4: **if** $v_n = e$ and $v_m \in C$ **then**
5: G.addEdge(v_n, v_m, b_{nm})
6: **if** $v_n = e$ and $v_m \in D$ and $((v_n, v_m) \in S)$ **then**
7: G.addEdge(v_n, v_m, b_{nm})
8: **if** $v_n \in C$ and $v_m \in C$ **then**
9: G.addEdge(v_n, v_m, b_{nm})
10: **if** $v_n \in C$ and $v_m \in D$ and $((v_n, v_m) \in S)$ **then**
11: G.addEdge(v_n, v_m, b_{nm})
12: **if** $v_n \in D$ and $v_m \in D$ and $((v_a, v_n) \in S$, $(v_a, v_m) \in S$, $v_a \in C \cup \{e\}$) **then**
13: G.addEdge(v_n, v_m, b_{nm})
14: **if** $v_n \in D$ and $v_m \in C$ and $((v_a, v_n) \in S$, $v_a \neq v_m$, v_a, $v_m \in C \cup \{e\})$ **then**
15: G.addEdge(v_n, v_m, b_{nm})
16: **Output:** Graph G

edge (line 1). Then, valid edges in the form of (v_n, v_m, b_{nm}) are added to the graph, where v_n is the source, v_m is the destination, and b_{nm} is the weight (travel cost) (lines 2-15). The conditions to add an edge are based on rules (1) to (8), defined in Section 2.1, and are as follows:

- An edge (direct path) exists from elevator e to each of its request floors (lines 4-5).
- An edge exists from elevator e to each of its destination floors (lines 6-7).
- An edge exists between request floors (lines 8-9). This allows the elevator to move from a request floor to another request floor.
- An edge exists from a request floor to its corresponding destination floor (lines 10-11).
- An edge exists between destination floors originating at the same request floor (lines 12-13).
- An edge exists from a destination floor to another request floor with no request or call for that destination. (lines 14-15).

Finally, the created graph is returned (line 16).

BuildPath maintains the created graph as $G(V, \mathcal{E})$ (line 1). It uses a greedy approach to find a cost-efficient path traversed by elevator e. It sorts edges in $\mathcal{E}$ based on the edge weights (travel costs) in increasing order and keeps them in $\bar{\mathcal{E}}$ (line 2). It then creates a new graph $G'(V', \mathcal{E}')$ with $V' = V$ and $|\mathcal{E}'| = 0$ (line 3). Graph G' will contain the final service path to be traversed by elevator e. Initially, no edge exists in the service path, and thus $|\mathcal{E}'| = 0$. Then, the edges are added to G' (lines 5-21). The variable ζ holds the path cost and is set to zero initially (line 4). *BuildPath* chooses an unselected edge (v_n, v_m) with the minimum cost from $\bar{\mathcal{E}}$ (line 6). It uses a variable *Valid* for the validity of the selected edge (line 7) and sets it to TRUE initially. *BuildPath* checks five conditions to decide whether to add the edge or not. If one of these conditions occurs, the selected edge is not valid, and thus, it is not appended to the path (lines 8-17). Note that δ^+ and δ^- denote out-degree and in-degree of a node, respectively. The conditions are as follows:

- *Valid* is FALSE if out-degree of v_n or in-degree of v_m is 1. This condition guarantees that no node in the path

Accepted Manuscript - Version of Record at: https://doi.org/10.1109/TSMC.2023.3237027

Algorithm 3 BuildPath(Elevator e, Coalition C)

1: $G(V, \mathcal{E}) =$ *BuildGraph*(Elevator e, Coalition C)
2: $\bar{\mathcal{E}}$ = Sort edges in $\mathcal{E}$ based on their costs in an increasing order
3: Create Graph $G'(V', \mathcal{E}')$ where $V' = V$, $|\mathcal{E}'| = 0$
4: ζ = 0
5: **repeat**
6: Choose an unselected minimum edge $(v_n, v_m) \in \bar{\mathcal{E}}$
7: *Valid* = TRUE
8: **if** $\delta^+(v_n \in V')$ = 1 or $\delta^-(v_m \in V')$ = 1 **then**
9: *Valid* = FALSE
10: **if** $v_n \in V'$ is visited and $v_m \in V'$ is visited **then**
11: *Valid* = FALSE
12: **if** $v_n \in R$ and $\delta^-(v_n \in V')$ = 0 **then**
13: *Valid* = FALSE
14: **if** $v_n \in D$ and $\delta^-(v_n \in V')$ = 0 **then**
15: *Valid* = FALSE
16: **if** $v_m \in D$ and $\delta^-(v_m \in V') = 0$ and $\exists\, v_a : (v_a, v_m) \in S$ and $v_a \in V'$ is not visited **then**
17: *Valid* = FALSE
18: **if** *Valid* = TRUE **then**
19: G'.addEdge(v_n, v_m, b_{nm})
20: $\zeta = \zeta + b_{nm}$
21: **until** $\nexists$ unvisited $\hat{v} \in V' : \delta^+(\hat{v})$=$\delta^-(\hat{v})$=0
22: $\mathcal{E}'$ contains edges of the path for elevator e to serve C, and ζ is its cost
23: **Output:** $\mathcal{E}'$ and ζ

is visited more than once (lines 8-9).

- *Valid* is FALSE if v_n and v_m are already visited, i.e., they are already in the path. This condition eliminates subtour from the path (lines 10-11).
- *Valid* is FALSE if $v_n \in R$ and its in-degree is zero. This condition guarantees that no path can continue from a request floor unless there exists an incoming edge to that request floor. In other words, an elevator first must reach a request floor and then the path continues (lines 12-13).
- *Valid* is FALSE if $v_n \in D$ and its in-degree is zero. This condition guarantees that no path can start from a destination floor unless there exists an incoming edge to that destination floor (lines 14-15).
- *Valid* is FALSE if $v_m \in D$ and its in-degree is zero and its request floor is not visited. This condition guarantees that a destination floor is never served unless its request floor is in the path (lines 16-17).

After checking these conditions, if *Valid* is TRUE, the selected edge is appended to the path, and the value of ζ is updated by the edge cost (lines 18-20). The algorithm continues until there is no unvisited node in V' (line 21). An unvisited node is a node with both in-degree and out-degree of zero in G'. After the procedure is done, $\mathcal{E}'$ contains all the edges that form a cost-efficient path starting from elevator e to serve all requests in the coalition, and ζ is the total cost of the path (line 22). *BuildPath* returns the path and its cost (line 23).

3.4 RCFM Properties

In this section, we prove RCFM converges to a final request coalition structure and any resulted partition is stable.

Theorem 3. Starting from an initial coalition structure $\Pi_{initial}$, RCFM always converges to a final coalition structure Π_f.

TABLE 3: Experiment Scenarios

Exp.	# E (u)	# floors	# R (v)	# D (w)	# stops (p)
A	2	10	3	5	3
B	3	20	8	15	6
C	10	20	15	48	7
D	15	35	20	74	8
E	20	50	30	85	8

Proof: RCFM utilizes the switch operation. Every switch operation transforms current coalition structure Π into Π'. The preference relation defined in (19) implies that every new switch operation leads to a preferred coalition structure that has never occurred. Since the total number of coalition structures is finite, RCFM always converges. □

Theorem 4. RCFM produces a stable request coalition structure.

Proof: The proof is by contradiction. Let us assume Π_f is the final coalition structure resulted from RCFM, and it is not stable. Hence, a request r_i exists that can perform a switch rule and joins a better request coalition. This contradicts the fact that Π_f is the result of the convergence of RCFM. Therefore, in the final request coalition structure, no request has the incentive to leave its request coalition. □

4 Experimental Results

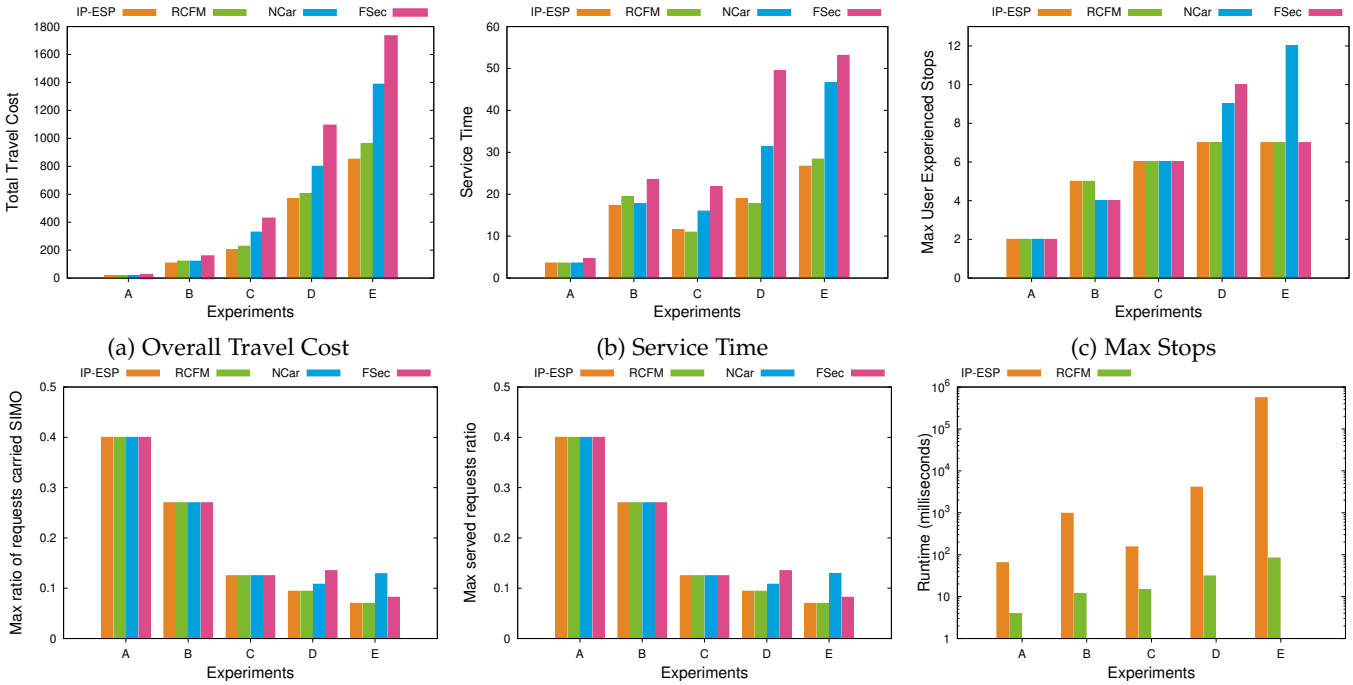
We performed a set of experiments to show the effectiveness of our approach in forming stable request coalitions for efficient elevator scheduling.

4.1 Experimental Setup

We implemented our proposed RCFM in C++. As a benchmark, we also implemented IP-ESP using IBM ILOG Concert Technology API for C++ [35]. All experiments were done on a standalone system with 16 GB RAM and 2.80 GHz, 11th Gen Intel(R) Core(TM) i7-1165G7 processor. The data for our experiments (request set R, destination set D, and collection set S) were generated based on elevator traffic patterns presented in different studies. Peters and Haddon [36] presented several metrics that help in explaining traffic patterns such as building type, population of occupants, and use of stairs. Liang *et al.* [37] presented elevator traffic patterns in an office building in terms of door-open events and elevator loads. We have adapted these patterns to generate realistic elevator traffic for different building scenarios.

In each of the scenarios, the elevators in E are initially placed at random floors within the buildings. The size of the request set and destination set may seem modest, but the resulting collection set S is much larger because it can be as large as the Cartesian product of R and D. The Poisson process plays a fundamental role in modeling systems involving arrivals of certain events at a specific rate in a purely random manner [38]. We model the request arrival pattern as a Poisson process with the rate $\lambda = n_r/300$, where n_r denotes the number of user requests arriving in the elevator system within 5 minutes. The elevators are rescheduled to provide service for the available requests in every time window (30 seconds). The number of times needed for rescheduling depends on the demands (number

Accepted Manuscript - Version of Record at: https://doi.org/10.1109/TSMC.2023.3237027



(a) Overall Travel Cost (b) Service Time (c) Max Stops

(d) Max Simultaneous Requests (e) Max Load (f) Runtime

Fig. 2: Experimental Results for Different Scenarios

of request arriving). Table 3 summarizes different scenarios chosen for the experiment. Experiment A represents a small-sized residential building with low demand. Experiment B represents a residential building with more calls from and to the first floor. Experiment C represents a commercial building with more elevators and requests, and a uniform distribution of requests and destinations. Experiment D represents an educational building, where specific floors have more traffic than others (e.g., classroom floors have more traffic compared to the administrative floors). Experiment E represents a significantly large building with heavy demand to evaluate the scalability.

We compare our approach, Request Coalition Formation Mechanism (RCFM), with the optimal Elevator Scheduling Problem (IP-ESP), Nearest Car (NCar), and Fixed Sector (FSec). NCar and FSec approaches are simple and well-known rule-based heuristic approaches used to make online elevator scheduling decisions.

4.2 Experimental Analysis

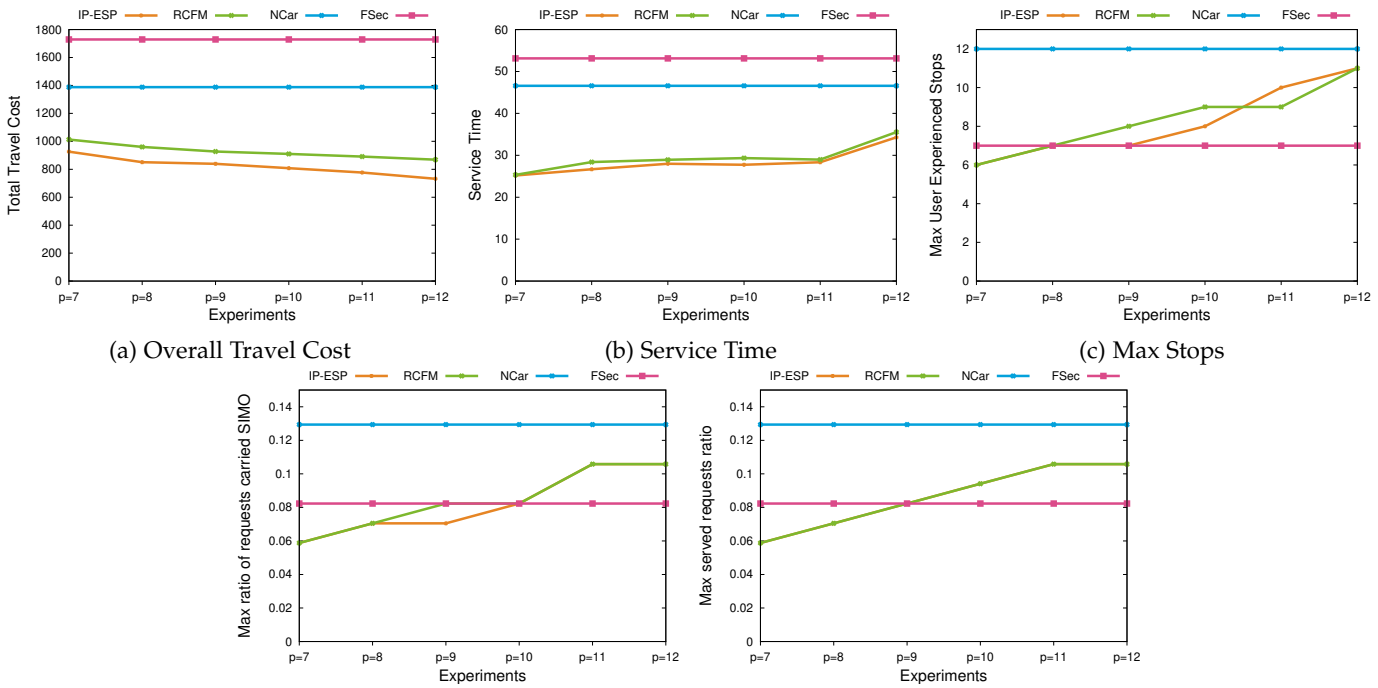
Figure 2a shows the elevators' overall travel cost. The results show that our proposed approach, RCFM, obtains close results to that of IP-ESP. Note that the feasible time to find a solution is set to 60 minutes for IP-ESP per time window of the available requests. In general, RCFM obtains significantly lower overall travel costs than NCar and FSec, except for small-size Experiments A and B, where those approaches display comparable results with our approach. The reason is that NCar and FSec are simple heuristic approaches and do not guarantee near-optimal results, especially in experiments with higher traffic.

Figure 2b shows the comparison of the average service time, where the service time for a user is a travel cost from requesting an elevator until it is served and reaches the destination. The results indicate RCFM obtains a much lower service time than NCar and FSec in the experiments with higher traffic. This is due to the fact that RCFM, using the switch rule, controls the serving of a single request to add the least amount to the overall travel cost and, consequently, overall service time. IP-ESP, as expected, obtains significantly lower service time compared to NCar and FSec. However, RCFM obtains a better service time than IP-ESP in some experiments since minimizing the service time is not a primary goal of the objective function, while the main purpose is to minimize the total travel cost. Figure 2c compares the max number of stops experienced by users from when they enter the elevator until they arrive at their destinations. The results indicate that both RCFM and IP-ESP obtain a fewer or equal number of stops with larger-scale scenarios since both approaches use this metric in finding the schedules. However, NCar and FSec only obtain a comparable number of stops in low traffic scenarios since, in general, as they are not sensitive to the number of stops, therefore, allowing elevators to take more stops in their service paths.

Figure 2d compares the maximum ratio of all requests served simultaneously by an elevator to analyze the approaches in dealing with overcrowdedness in elevators. Figure 2e compares the maximum ratio of requests served by a single elevator to analyze the load balancing among elevators. Both results indicate RCFM and IP-ESP obtain comparable results. At the same time, NCar and FSec never outperform RCFM and IP-ESP in forming less crammed and more balanced elevator schedules since RCFM and IP-ESP control the maximum number of users and load of the elevators by using p.

Figure 2f compares the average running time of RCFM

Accepted Manuscript - Version of Record at: https://doi.org/10.1109/TSMC.2023.3237027



(a) Overall Travel Cost (b) Service Time (c) Max Stops

(d) Max Simultaneous Requests (e) Max Load

Fig. 3: Sensitivity Analysis for p in Experiment E

with IP-ESP per a time window of arriving requests since both NCar and FSec are trivial approaches. The results indicate RCFM is significantly faster than IP-ESP for all experiments. Our approach can converge in a reasonable time, roughly 83ms per time window for the largest experiment (Experiment E) and is proper for larger buildings and a higher number of requests and destinations.

We perform sensitivity analysis for the number of stops, parameter p, to analyze its effects on the results. Figure 3 summarizes the results of sensitivity analysis for experiment E over the value of p. In all these experiments, NCar and FSec are not impacted by the value of p and show the same results.

Figure 3a shows that increasing the number of stops reduces the overall distance for both IP-ESP and RCFM approaches. This is expected as the elevators are allowed to take better energy-efficient paths with more stops. Figure 3b indicates increasing the number of stops that are allowed in the service path negatively impacts the service time for the users. As the number of stops increases, service path length increases, leading to a longer service time for the users. Figures 3c, 3d, 3e show in general allowing a higher number of stops increases a) the max stops experienced by users, b) the maximum ratio of users in an elevator, and c) the maximum load of the elevators, respectively. By increasing the value of p, the elevators can choose paths with more stops, and thus, users are likely to experience more stops in their service path to their destination. Besides, the number of requests that can be served by a single elevator and the maximum number of users served simultaneously by an elevator increase.

To summarize, the obtained results validate the effectiveness of RCFM and applying coalitional games to tackle the elevator scheduling problem. Multiple scenarios with different sizes and even complicated request patterns show that our approach is robust.

5 Conclusion and Future Work

This paper investigates the energy-efficient elevator scheduling problem for smart buildings. We proposed a novel approach, called RCFM, based on coalitional game concepts capturing interactions within the elevator system to reduce elevator movements in smart buildings and service time for users. The requests can join or leave a request coalition to establish efficient and stable request coalitions. We showed that our approach obtains near-optimal results and provides significantly more efficient schedules than NCar and FSec. For the future work, we plan to investigate stochastic optimization to capture probabilistic and uncertain future requests. In addition, we plan to design an online approach using machine learning models and coalitional games to create and modify request coalitions in realtime when serving smart buildings.

Acknowledgment

This research was supported in part by NSF grant CNS-2145268.

References

[1] D. Minoli, K. Sohraby, and B. Occhiogrosso, "IoT considerations, requirements, and architectures for smart buildings-energy optimization and next-generation building management systems," *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 269–283, 2017.

[2] L. Klein, J.-y. Kwak, G. Kavulya, F. Jazizadeh, B. Becerik-Gerber, P. Varakantham, and M. Tambe, "Coordinating occupant behavior for building energy and comfort management using multi-agent systems," *Automation in construction*, vol. 22, pp. 525–536, 2012.

[3] "Truly game-changing elevator maintenance," [Online]: https://max.thyssenkrupp-elevator.com/en/, 2019, accessed: 2019-05-30.

[4] W. Spaniel, "The game theory of MPSA elevators," [Online]: https://williamspaniel.com/2014/04/03/the-game-theory-of-mpsa-elevators/, 2014, accessed: 2018-11-30.

[5] J. Koehler, "From theory to practice: AI planning for high performance elevator control," in *Proc. of the Annual Conference on Artificial Intelligence*, 2001, pp. 459–462.

[6] J. Koehler and D. Ottiger, "An AI-based approach to destination control in elevators," *AI magazine*, vol. 23, no. 3, p. 59, 2002.

[7] A. Jansson and K. Uggla Lingvall, "Elevator control using reinforcement learning to select strategy," Degree Project. First Level, KTH Royal Institute of Technology, Stockholm, Sweden, 2015.

[8] L.-D. Van, Y.-B. Lin, T.-H. Wu, and T.-H. Chao, "Green elevator scheduling based on IoT communications," *IEEE Access*, vol. 8, pp. 38 404–38 415, 2020.

[9] K. Skrzypczyk, "Game theory based task planning in multi robot systems," *Intl. Journal of Simulation*, vol. 6, no. 6, pp. 50–60, 2005.

[10] W. Li, L. Gao, X. Li, and Y. Guo, "Game theory-based cooperation of process planning and scheduling," in *Proc. of the IEEE 12th Intl. Conference on Computer Supported Cooperative Work in Design*, 2008, pp. 841–845.

[11] G. Zhou, Z. Xiao, P. Jiang, and G. Q. Huang, "A game-theoretic approach to generating optimal process plans of multiple jobs in networked manufacturing," *Intl. Journal of Computer Integrated Manufacturing*, vol. 23, no. 12, pp. 1118–1132, 2010.

[12] J. Edlund and F. Berntsson, "Constructing a scheduling algorithm for multidirectional elevators," Degree Project. First Level, KTH Royal Institute of Technology, Stockholm, Sweden, 2015.

[13] A. Valdivielso and T. Miyamoto, "Multicar-elevator group control algorithm for interference prevention and optimal call allocation," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 41, no. 2, pp. 311–322, 2010.

[14] P. E. Utgoff and M. E. Connell, "Real-time combinatorial optimization for elevator group dispatching," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 42, no. 1, pp. 130–146, 2011.

[15] A. Vodopija, J. Stork, T. Bartz-Beielstein, and B. Filipič, "Elevator group control as a constrained multiobjective optimization problem," *Applied Soft Computing*, vol. 115, p. 108277, 2022.

[16] J. Zhang and Q. Zong, "Energy-saving scheduling optimization under up-peak traffic for group elevator system in building," *Energy and Buildings*, vol. 66, pp. 495–504, 2013.

[17] Y. Wu and S. Tanaka, "A mixed-integer programming approach to group control of elevator systems with destination hall call registration," in *Proc. of the IEEE Intl. Conference on Industrial Engineering and Engineering Management*, 2020, pp. 26–31.

[18] J. Xu and T. Feng, "Single elevator scheduling problem with complete information: An exact model using mixed integer linear programming," in *Proc. of the IEEE American Control Conference*, 2016, pp. 2894–2899.

[19] R. H. Crites and A. G. Barto, "Improving elevator performance using reinforcement learning," in *Proc. of the 8th Intl. Conference on Neural Information Processing Systems*, 1995, pp. 1017–1023.

[20] C. Kim, K. A. Seong, H. Lee-Kwang, and J. O. Kim, "Design and implementation of a fuzzy elevator group control system," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 28, no. 3, pp. 277–287, 1998.

[21] O. Kwon, E. Lee, and H. Bahn, "Sensor-aware elevator scheduling for smart building environments," *Building and Environment*, vol. 72, pp. 332 – 342, 2014.

[22] S. Wang, X. Gong, M. Song, C. Y. Fei, S. Quaadgras, J. Peng, P. Zou, J. Chen, W. Zhang, and R. J. Jiao, "Smart dispatching and optimal elevator group control through real-time occupancy-aware deep learning of usage patterns," *Advanced Engineering Informatics*, vol. 48, p. 101286, 2021.

[23] 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. of the IEEE Youth Conference on Information, Computing and Telecommunications*, 2010, pp. 37–40.

[24] S. Cui, Y.-W. Wang, and J.-W. Xiao, "Peer-to-peer energy sharing among smart energy buildings by distributed transaction," *IEEE Transactions on Smart Grid*, vol. 10, no. 6, pp. 6491–6501, 2019.

[25] V. Zarikas and N. Tursynbek, "Intelligent elevators in a smart building," in *Proc. of Future Technologies Conference*, 2017, pp. 126–133.

[26] Y. Bapin and V. Zarikas, "Smart building's elevator with intelligent control algorithm based on bayesian networks," *Intl. Journal of Advanced Computer Science and Applications*, vol. 10, no. 2, pp. 16–24, 2019.

[27] S. Ramalingam, A. U. Raghunathan, and D. Nikovski, "Submodular function maximization for group elevator scheduling," in *Proc. of the 27th Intl. Conference on Automated Planning and Scheduling*, 2017, pp. 233–241.

[28] A. De Almeida, C. Patrão, J. Fong, U. Nunes, and R. Araújo, "E4-Energy Efficient Elevators and Escalators," [Online]: https://ec.europa.eu/energy/intelligent/projects/sites/iee-projects/files/projects/documents/e4_publishable_report_en.pdf, Accessed: 2020-12-27.

[29] T. Bektas, "The multiple traveling salesman problem: an overview of formulations and solution procedures," *Omega*, vol. 34, no. 3, pp. 209 – 219, 2006.

[30] G. Dantzig, R. Fulkerson, and S. Johnson, "Solution of a large-scale traveling-salesman problem," *Journal of the operations research society of America*, vol. 2, no. 4, pp. 393–410, 1954.

[31] C. E. Miller, A. W. Tucker, and R. A. Zemlin, "Integer programming formulation of traveling salesman problems," *Journal of the ACM*, vol. 7, no. 4, pp. 326–329, 1960.

[32] L. Mashayekhy, M. M. Nejad, and D. Grosu, "A trust-aware mechanism for cloud federation formation," *IEEE Transactions on Cloud Computing*, vol. 9, no. 4, pp. 1278–1292, 2021.

[33] ——, "Cloud federations in the sky: Formation game and mechanism," *IEEE Transactions on Cloud Computing*, vol. 3, no. 1, pp. 14–27, 2015.

[34] H. Aziz, F. Brandl, F. Brandt, P. Harrenstein, M. Olsen, and D. Peters, "Fractional hedonic games," *ACM Transactions on Economics and Computation*, vol. 7, no. 2, pp. 1–29, 2019.

[35] IBM ILOG CPLEX Optimizer, [Online]: https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-optimizer.

[36] R. Peters, P. Mehta, and J. Haddon, "Lift passenger traffic patterns: applications, current knowledge and measurement," *Elevator World*, vol. 48, no. 9, pp. 87–94, 2000.

[37] C.-J. M. Liang, J. Tang, L. Zhang, F. Zhao, S. Munir, and J. A. Stankovic, "On human behavioral patterns in elevator usages," in *Proc. of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings*, ser. BuildSys'13, 2013, pp. 29:1–29:2.

[38] G. Last and M. Penrose, *Lectures on the Poisson process*. Cambridge University Press, 2017, vol. 7.

BIOGRAPHIES

Erfan Farhangi Maleki is currently a Ph.D. candidate and a Doctoral Fellow in the Department of Computer and Information Sciences at the University of Delaware. His research interests include edge/cloud computing, 5G, and distributed systems.

Dixit Bhatta received his Ph.D. in Computer Science from the University of Delaware in 2022. His research interests include edge/cloud computing, Internet of Things, and distributed systems. He was a Doctoral Fellow in the Department of Computer and Information Sciences before graduation. He also received the Best Paper Award at the IEEE CloudCom 2019 and the 2020 Outstanding Graduate Student Award.

Lena Mashayekhy is an Associate Professor in the Department of Computer and Information Sciences at the University of Delaware. Her research interests include edge/cloud computing, Internet of Things, and game theory. She is a recipient of the 2016 IEEE TCSC Outstanding PhD Dissertation Award, the 2017 IEEE TCSC Early Career Researcher Award for Excellence in Scalable Computing, the 2022 NSF CAREER Award, and several best paper awards.