Minimizing Maximum Completion Time in EV Charging Station Scheduling

Submitted to:

Brian Denton[1]

[1]Stephen M. Pollock Collegiate Professor, Industrial and Operations Engineering
University of Michigan, 1205 Beal Avenue, Ann Arbor, MI 48109

Submitted by:

Kuo Wang[2], Daniel Wang[2], Aimeng Yang[3]

[2]Master Student, Industrial and Operations Engineering
[3]Undergraduate Student, Industrial and Operations Engineering
University of Michigan, 1205 Beal Avenue, Ann Arbor, MI 48109

Date Submitted:
04/25/2024

1 Introduction

1.1 Motivation

As Electric Vehicles (EVs) become more widespread, the efficient operation of EV charging stations has emerged as a critical challenge in sustainable transportation systems. One of the primary operational issues is how to schedule the charging of multiple vehicles across limited charging ports, particularly when vehicles arrive at different times. In this context, the scheduling problem $P_m|r_j|C_{\max}$, which seeks to minimize the maximum vehicle departure time (makespan) on parallel chargers with release times, provides a natural and practical modeling framework.

1.2 Problem Setting and Assumptions

We focus on a realistic scenario where all EVs arrive and must be charged within a predefined time window, and our objective is to optimize the scheduling strictly within this period. All chargers are assumed to have identical infrastructure (i.e., voltage levels) under our problem setting. We model the release times of EVs (i.e., interarrival times) using an exponential distribution (Huttunen, 2008), a common choice for representing memoryless arrival processes. Additionally, to capture the variability in charging requirements among different types of EVs (e.g., Tesla versus other EVs), we generate the processing times using two gamma distributions with different shape (α) and scale (θ) parameters.

The gamma distribution is well-suited for modeling positively skewed, non-negative data, which makes it a natural choice for characterizing EV charging durations. Studies have demonstrated its versatility in related applications. For example, Farhadi (Farhadi et al., 2024) showed that the gamma distribution can effectively represent random daily driving distances in plug-in hybrid electric vehicles (PHEVs). It can also be used to model aggregated EV charging station loads based on empirical data (Louie, 2015). Solving this problem effectively is crucial for minimizing customer wait times, maximizing charger utilization, smoothing station operations during peak demand periods, and supporting broader energy management goals.

1.3 Research Objective and Approach

While Integer Programming (IP) formulations can be used to solve parallel machine scheduling problems, it is well known that these problems are NP-hard. As a result, solving large instances exactly using IP becomes computationally expensive and often impractical due to long processing times. In this project, we aim to apply a Genetic Algorithm (GA) approach to produce solutions that are close to the optimal IP solutions, while significantly reducing computation time and improving scalability for larger datasets.

1.4 Overview of the Report

The remainder of this report presents the detailed modeling approaches and solution methods used to address the parallel machine scheduling problem $P_m|r_j|C_{max}$ within the context of EV charging operations. We develop both an IP model and a GA model to solve the problem, implementing both methods in Python. The performance of the two models is evaluated by comparing their resulting makespan values, either optimal (from IP) or near-optimal (from GA), across a range of instances. We categorize the results into two scenarios: (1) IP-solvable cases, where the exact optimal solution can be computed within a reasonable time, and (2) IP-unsolvable cases, where the solver fails to find a solution within a predefined time limit. Our findings demonstrate that the GA performs

remarkably well, providing high-quality approximations with significantly reduced computational time, particularly in large-scale instances where the IP model becomes impractical.

2 Model and Methods

In this study, all the EVs are acted as job and the EV chargers serve as machines. There is a set of n jobs need to be processed on m identical parallel machines. The job j can be processed by either of the machines i with same processing time. Each machine can process only one job at a time. Each job need to be processed without interruption. Each job j has its release time r_j following an exponential distribution. Processing time of job j on machine i is denoted by p_j regardless of which machine they are assigned. We consider there are two types of EVs, one is fast-charging EV (e.g., Tesla) and the other is slow-charging EV (e.g., other EVs), which follow two gamma distributions with different shape (α) and scale (θ) parameters respectively. It is desired to find a schedule for which the maximum completion time (makespan) is minimized.

2.1 Integer Programming (IP)

First, we formulated this $P_m|r_j|C_{\max}$ as an IP problem, which is a typical method to solve parallel machine scheduling problem with a small dataset, to find the optimal schedule with a minimum makespan. Based on the formulation in the lecture (Denton, 2025), our team modified it and added another decision variable and constraint that can meet the release time requirement r_j. Detailed IP formulation of problem $P_m|r_j|C_{\max}$ are shown in the following sections.

2.1.1 Decision Variables

To formulate the problem in IP, we need to first define some decision variables whose values are not known beforehand and are determined through the optimization process. This section shows the decision variables we are going to use in IP:

- $\boldsymbol{x_{ij}} = \begin{cases} 1, & \text{if job } j \text{ is assigned to machine } i \\ 0, & \text{otherwise} \end{cases}$
- $\boldsymbol{s_j}$, the actual start time of job j on machine i
- $\boldsymbol{y^i_{jk}} = \begin{cases} 1, & \text{if job } j \text{ is assigned before job } j \text{ on machine } i \\ 0, & \text{otherwise} \end{cases}$
- $\boldsymbol{C_{\max}}$, maximum completion time (makespan) among all machines

2.1.2 Parameters

The parameters are the inputs for the IP problem. We assume that below parameters are all known and generated randomly. The release time of job j, r_j, follows exponential distribution, and the processing time of job j on machine i, p_j, follows gamma distributions (different shape (α) and scale (θ) parameters represent different processing time of fast-charging EVs and slow-charging EVs):

- $\boldsymbol{N}$, set of all n jobs, index by j
- $\boldsymbol{M}$, set of all m machines, index by i

- $\boldsymbol{r_j}$, the release time of job j
- $\boldsymbol{p_j}$, the processing time of job j on machine i

2.1.3 Objective Function

Objective function is the goal that the model want to achieve. In this case, the objective is to minimize the maximum completion time (makespan) within a certain period. To achieve it, our objective function becomes:

$$\min \quad C_{\max} \tag{1}$$

2.1.4 Constraints

In the corresponding IP formulation, decision variables are used to assign each EV to a specific charger and determine its start time, subject to several constraints. These constraints ensure that each EV is assigned to exactly one charger, that no two EVs overlap on the same charger at any time, and that each EV's charging process cannot start before its release time. The detailed constraints are defined as follows:

- Each EV (job j) is assigned to exactly one charger:

$$\sum_{i=1}^{m} x_{ij} = 1, \quad \forall j \in N \tag{2}$$

- The starting time of chariging each EV (job j) should be no earlier than its release time:

$$s_j \geq r_j, \quad \forall j \in N \tag{3}$$

- The maximum completion time of the system should be larger or equal than any completion time of EV, i.e., the completion time of every EV (starting time s_j + processing time p_j) should be smaller or equal to the makespan ($C_{\max}$):

$$s_j + p_j \leq C_{\max}, \quad \forall j \in N \tag{4}$$

- (Big M Constraint) The big M method models constraints involving binary variables to linearize non-linear constraints, enabling or disabling certain constraints based on the value of the binary variable. In this case, if EV k is assigned to charger i ($x_{ik} = 1$) after EV j is completed on charger i ($x_{ij} = 1$ and $y^i_{jk} = 1$), then the starting time of EV k on charger i (s_k) should be later than the starting time (s_j) plus the processing time (p_j) of EV j on charger i. Any contradiction to this setting will lead this constraint disable. The formulation of big M constraint is shown below:

$$s_j + p_j \leq s_k + M(1 - y^i_{jk}) + M(1 - x_{ij}) + M(1 - x_{ik}), \quad \forall i,\ j \neq k \tag{5}$$

- (Big M Constraint) Similarly, if both EV k and EV j are assigned to charger i ($x_{ik} = 1$ and $x_{ij} = 1$) but EV j is assigned after EV k ($y^i_{jk} = 0$), then the constraint restricts that the starting time of EV j on charger i (s_j) should be later than the starting time (s_k) plus the processing time (p_j) of EV j on charger i. Any contradiction to this setting will lead this constraint disable. The formulation of big M constraint is shown below:

$$s_k + p_k \leq s_j + M \times y^i_{jk} + M(1 - x_{ij}) + M(1 - x_{ik}), \quad \forall i,\ j \neq k \tag{6}$$

2.2 Genetic algorithm (GA)

To address the parallel machine scheduling problem with the large dataset, we implement a GA framework that iteratively improves job assignments across multiple machines. In this approach, each solution, or individual, is encoded as a matrix representing the allocation of jobs to machines, ensuring that each job is assigned to exactly one machine.

The initial population is generated randomly under the feasibility constraint. Each individual's quality is evaluated based on the makespan, defined as the maximum completion time across all machines, incorporating both job release times (r_j) and processing durations (p_j).

The evolutionary process proceeds through selection, crossover, and mutation operators. Selection emphasizes individuals with superior performance (lower makespans), while crossover and mutation introduce new job-machine assignments to diversify the search space. Special attention is given to maintaining solution feasibility during these operations.

By iteratively applying these mechanisms, the GA explores the solution space and seeks to minimize the makespan, promoting balanced workloads across machines while respecting problem-specific constraints. Figure 1 shows the general structure of GA.

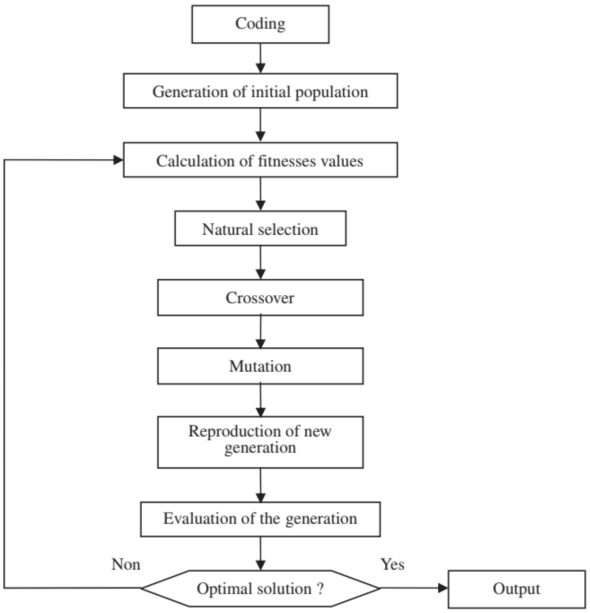

Figure 1: General structure of GA (Balin, 2011)

Following sections introduce the decision variables, parameters, objective, and the iteration process the team used to apply GA in our case with parallel machines scheduling.

2.2.1 Decision Variables

This section shows the decision variables we are going to use in GA:

- $\boldsymbol{x_{ij}} = \begin{cases} 1, & \text{if job } j \text{ is assigned to machine } i \\ 0, & \text{otherwise} \end{cases}$

- $\boldsymbol{s_{ij}}$, the actual start time of job j on machine i
- $\boldsymbol{C_i}$, the completion time of all job assigned on machine i
- $\boldsymbol{C_{\max}}$, maximum completion time (makespan) among all machines

2.2.2 Parameters

We assume that below parameters are all known:

- $\boldsymbol{N}$, set of all n jobs, index by j
- $\boldsymbol{M}$, set of all m machines, index by i
- $\boldsymbol{r_j}$, the release time of job j
- $\boldsymbol{p_{ij}}$, the processing time of job j on machine i

2.2.3 Objective

Same with the objective function in IP, here, the objective is also the goal we want to achieve when applying GA. Since each machine i has a completion time C_i:

$$C_i = \sum_{j=1}^{n} [x_{i,j} \max(r_j, s_{ij}) + p_{ij}] \tag{7}$$

The makespan is:

$$C_{\max} = \max_i \ C_i \tag{8}$$

2.2.4 Coding

The matrix $\mathbf{X}$ represents the scheduling of n jobs across m machines (Min and Cheng, 1999). The i-th row of the matrix $\mathbf{X}$ represents the jobs assigned to machine i. Each row is called a "gene" $(g_1, \ldots, g_i, \ldots, g_m)$, where the elements of the gene correspond to jobs processed on machine i; a job j is assigned to machine i if $x_{ij} = 1$. The completion time of machine i, denoted by C_i, is the sum of the processing times of the jobs assigned to it. This is referred to as the "value of gene i", and is defined by the following function:

$$f(g_i) = \sum_{j=1}^{n} \left[x_{ij} \max(r_j, s_{ij}) + p_{ij} \right], \quad i = 1, 2, \ldots, n \tag{9}$$

2.2.5 Generation of Initial Population

An initial solution of matrix $\mathbf{X}$ can be generated randomly by ensuring that each column of the matrix contains only one non-zero element, $x(i, j) \in \{0, 1\}$. Multiple initial solutions can be obtained by repeating this process, and each solution is referred to as a "chromosome". Chromosomes are numbered according to the order in which they are created, denoted as $k \in N$. The initial population consists of N chromosomes, and the number of chromosomes, or population size, is a key parameter in the Genetic Algorithm (GA). The chromosomes in the initial population serve as the parents for subsequent generations, and the algorithm's efficiency largely depends on the "quality" of these initial chromosomes.

2.2.6 Calculation of Fitness Values

Fitness is used to evaluate the performance of chromosomes. After generating a new population, the fitness value of each chromosome, denoted as F_k, is calculated. A higher fitness value indicates better performance, meaning chromosomes with higher fitness values are more likely to survive. Since the goal is to minimize the makespan, fitness values are determined using the following function:

$$F_k = \alpha * \ e^{-\beta * C_{\max}(k)} \tag{10}$$

where α and β are positive real numbers, and $C_{\max}(k)$ is the objective function value (makespan) of chromosome k (Min and Cheng, 1999).

2.2.7 Reproduction

Reproduction is a process where parents generate offspring based on probabilities that correspond to their fitness values. This means that individuals with higher fitness levels have a greater likelihood of passing on their genes to the next generation. To select parents, we employ the roulette wheel selection method, which ranks chromosomes according to their fitness scores and assigns them probabilities that favor the fittest individuals. This approach increases the chances of producing stronger offspring in subsequent generations. The steps of the roulette wheel selection method are as follows:

Calculate selection probability of chromosomes,

$$P(k) = \frac{F_k}{\sum_{\mathrm{i}=1}^{N} F(i)}, \qquad k = 1, 2, ..., N \tag{11}$$

Generate cut points, S(k),

$$S(0) = 0 \tag{12}$$

$$S(k) = P(1) + P(2) + ... + P(k), \qquad k = 1, 2, ..., N \tag{13}$$

Generate N random number uniformly distributed between 0 and 1, ζ_s for $s = 1, 2, .., N$. For each ζ_s, equation $S(k-1) < \zeta_s < S(k)$ gives chromosomes to be selected.

2.2.8 Crossover

The gene $g_{\mathrm{i'}}$ with the maximum completion time is identified:

$$f(g_{\mathrm{i'}}) = \max_i f(g_i) \tag{14}$$

Instead of selecting the shortest processing time job, we now select the job with the earliest available start time while ensuring its release time is met:

$$j' = \arg\min_j (max(r_j, S_{\mathrm{i'},\mathrm{j}}) + P(i', j)) \tag{15}$$

The job j' is moved to another machine z, selected based on:

$$z = \arg\max_i (f(g_{\mathrm{i'}}) - f(g_i)) \tag{16}$$

ensuring that the new start time respects r_j:

$$S_{\mathrm{z,j'}} \geq r_{\mathrm{j'}} \tag{17}$$

2.2.9 Mutation

The mutation operator moves a job from one machine to another only if its release time constraint is satisfied:

$$S_{z,j'} \geq r_{j'} \tag{18}$$

- If moving a job violates r_j, he mutation is rejected.
- If multiple valid moves exist, pick the one that minimizes $C_{\max}$.

3 Result

After formulating our parallel machine scheduling problem $P_m|r_j|C_{\max}$ as an IP model, we observed that the IP solver can obtain the exact optimal solution within a few seconds for small instances. However, as the instance size (i.e., the number of machines/chargers) increases, solving the IP model becomes significantly more time-consuming. For sufficiently large instances, the IP solver is unable to find a feasible solution within practical computational limits.

In contrast, when the problem is formulated using a GA model, the solutions obtained are approximate but closely match the optimal solutions produced by the IP model for smaller instances. Moreover, as the instance size increases, the runtime for the GA remains relatively stable and, in some cases, the GA achieves even better approximations to the optimal solutions obtained by the IP model.

The following section presents the optimal schedules under both IP-solvable and IP-unsolvable scenarios, along with a comparison of the runtime performance between the IP and GA approaches.

3.1 Results of Solvable Instances

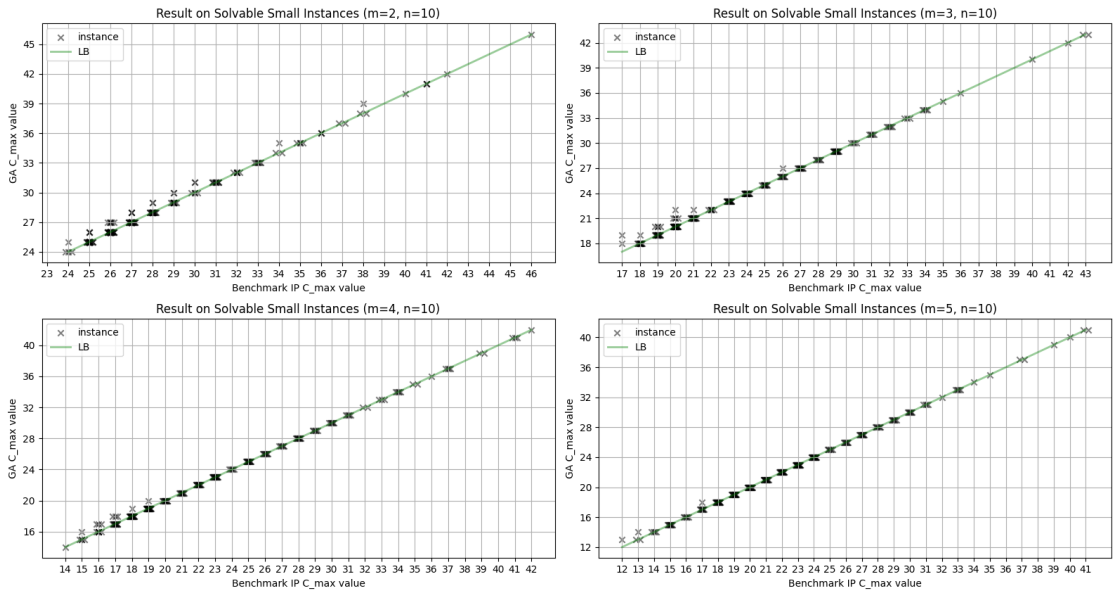

Figure 2: Makespan Comparison Between IP and GA Over 200 Instances (IP is Solvable)

Table 1: Average Runtime (in seconds) of GA and IP Algorithms on Solvable Instances

m	n	GA	IP
2	10	0.04	1.48
3	10	0.04	1.66
4	10	0.05	0.64
5	10	0.06	0.07

We consider a case where 10 electric vehicles (EVs) ($n = 10$) are waiting to charge, each with different release times (r_j). The number of available chargers varies from 2 to 5 ($m \in \{2, 3, 4, 5\}$). For each condition ($n = 10$ and $m = i$, where $i \in \{2, 3, 4, 5\}$), we randomly generate 200 instances to evaluate model performance.

In our simulation, the release times r_j for all EVs are sampled from a common exponential distribution to model random arrival patterns. The processing times, however, differ based on the EV type: supercharging EVs (e.g., Tesla vehicles) and standard EVs have their processing times drawn from two distinct gamma distributions. This setup captures practical variability in charging speeds and reflects differences in EV technologies.

The makespan for each instance is computed using both the IP and GA models to evaluate their performance under solvable conditions. As shown in Figure 2, the lower bound (LB) represents the optimal makespan obtained from the IP model, serving as the benchmark for comparison. Since the IP solutions are already optimal, the GA model cannot achieve a smaller makespan than the LB. The figure illustrates the approximate solutions generated by the GA model, where some instances align with the LB, indicating optimal performance, while others lie above the LB, reflecting a larger makespan.

While the GA and IP approaches produced comparable solution quality for solvable cases, their computational efficiency varied significantly. As illustrated in Table 1, GA consistently demonstrated substantially shorter runtimes, averaging approximately 0.04 to 0.06 seconds across different numbers of chargers. In contrast, IP runtimes varied between 0.07 and 1.66 seconds, underscoring GA's computational speed even when optimal solutions are easily obtainable. This advantage becomes increasingly relevant as instance complexity grows.

In conclusion, for smaller and solvable scenarios, both GA and IP models achieve high-quality scheduling solutions, with GA closely approximating IP's optimal results. However, GA significantly outperforms IP in computational efficiency, as evidenced by consistently shorter runtimes (around 0.04–0.06 seconds compared to IP's 0.07–1.66 seconds). This indicates that the GA algorithm not only maintains solution quality but also provides substantial advantages in runtime efficiency, even when exact optimal solutions are readily attainable by IP.

3.2 Results of Unsolvable Instances

$$GAP\% = \frac{C_{\max}(GA) - C_{\max}(IP)}{C_{\max}(IP)} \times 100\%$$

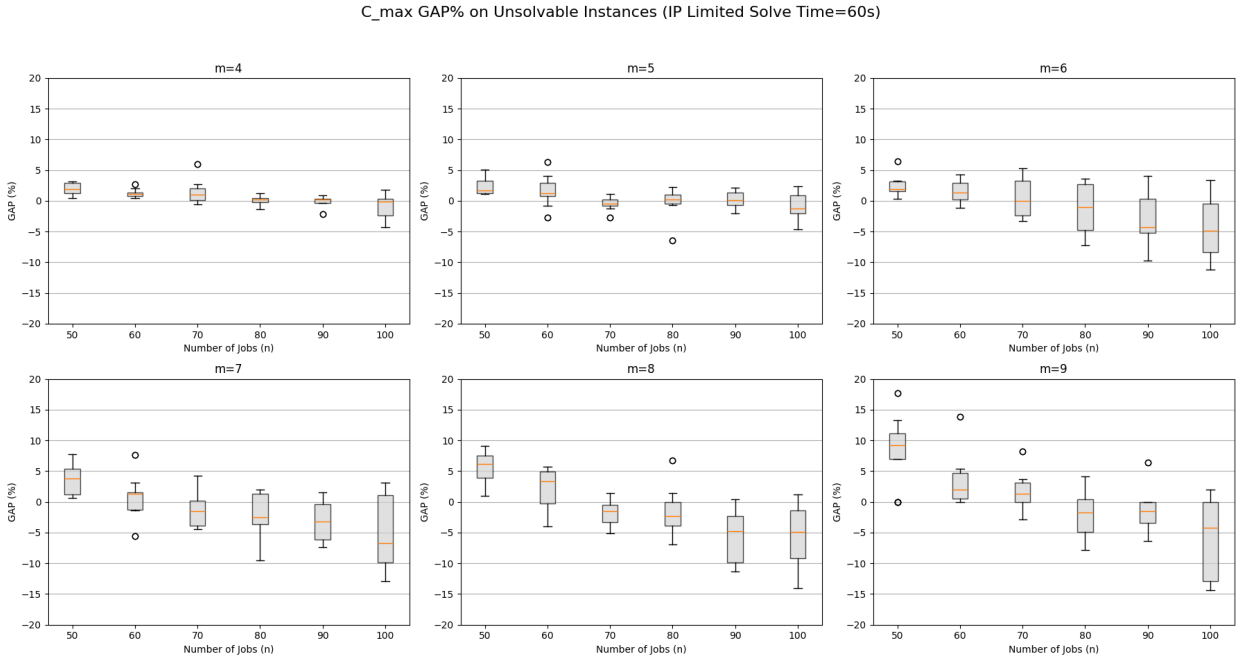

Figure 3: C_{max} GAP% between GA and IP Algorithms on Unsolvable Instances (IP Time Limit = 60s)

Table 2: Average Runtime (in seconds) of GA and IP Algorithms on Unsolvable Instances (IP Time Limit = 60s)

m	n	GA	IP
4	50	0.18	60.00
	60	0.22	60.00
	70	0.25	60.00
	80	0.28	60.00
	90	0.32	60.00
	100	0.35	60.00

m	n	GA	IP
5	50	0.22	60.00
	60	0.26	60.00
	70	0.30	60.00
	80	0.33	60.00
	90	0.37	60.00
	100	0.41	60.00

m	n	GA	IP
6	50	0.25	60.00
	60	0.29	60.00
	70	0.34	60.00
	80	0.38	60.00
	90	0.42	60.00
	100	0.48	60.00

m	n	GA	IP
7	50	0.29	60.00
	60	0.33	60.00
	70	0.39	60.00
	80	0.43	60.00
	90	0.48	60.00
	100	0.55	60.00

m	n	GA	IP
8	50	0.31	60.00
	60	0.37	60.00
	70	0.43	60.00
	80	0.48	60.00
	90	0.54	60.00
	100	0.64	60.00

m	n	GA	IP
9	50	0.34	60.00
	60	0.41	60.00
	70	0.48	60.00
	80	0.54	60.00
	90	0.60	60.00
	100	0.66	60.00

We also conducted an analysis compares the performance of GA against the IP solver in hard instances that the IP could not solve within a 60-second time limit. Each scenario, defined by the number of EV chargers m (ranging from 4 to 9) and the number of EVs n (ranging from 50 to

100), was evaluated across 100 independent trials.

In general, the GA algorithm closely matched or slightly exceeded the IP solver performance, particularly when the number of EVs n was smaller, with median GAP percentages generally close to 0%. However, as n increased, especially in the range of 90–100 EVs, the performance gap widened, showing increased variability and a tendency for GA to produce higher $C_{\max}$ values compared to IP. This trend was consistent across different numbers of EV chargers ($m = 4$ to $m = 9$), although the discrepancy in GAP was more pronounced for higher values of m.

Specifically, at $m = 4$ and $m = 5$, the median GAP remained around 0–2%, indicating that GA performs competitively even for larger problem sizes. In contrast, for larger values of m (such as $m = 7, 8, 9$), the median GAP shifted consistently below zero with increasing n, suggesting a relative deterioration of GA performance compared to IP under more complex scenarios. Furthermore, the interquartile range and the frequency of outliers increased notably for larger n and m, indicating greater uncertainty and variability in the effectiveness of the GA algorithm as problem complexity grew.

Considering the computational limitations observed in solvable scenarios, the runtime differences become even more pronounced in large, unsolvable instances. Table 2 demonstrates GA's efficiency, maintaining stable and short runtimes regardless of increasing problem size. In contrast, IP consistently reached the 60-second computational limit without successfully identifying feasible solutions. Thus, the GA method emerges as notably more scalable and practical for solving large-scale, complex scheduling problems.

Overall, for larger, unsolvable scheduling instances, the GA method demonstrates strong competitiveness by consistently approximating IP's solutions with minimal quality degradation, particularly for moderate-size scenarios. Crucially, GA significantly surpasses IP in computational efficiency, maintaining remarkably short and stable runtimes, whereas IP routinely hits the computational timeout limit without producing feasible solutions. Thus, considering both solution quality and runtime performance, GA is better suited and more practical for handling complex, large-scale EV scheduling problems.

4 Conclusion

In this report, we investigated the scheduling problem of electric vehicle charging under parallel charger configurations, specifically addressing the scheduling model denoted as $P_m|r_j|C_{\max}$. This problem reflects a practical scenario faced by charging station operators, wherein multiple EVs with varying arrival times and different charging requirements must be efficiently assigned to a limited set of available chargers. The objective of this scheduling is to minimize the overall completion time, or makespan, which directly impacts charger utilization efficiency.

To address this challenge, we formulated the problem as an Integer Programming model and developed a Genetic Algorithm heuristic as an alternative approach. The IP method provided exact solutions and served as the benchmark for evaluating the quality of the GA's approximate solutions. While the IP method reliably produced optimal schedules for smaller instances within acceptable computation times, its feasibility rapidly diminished as the size and complexity of the scheduling instances increased. In contrast, the GA method was designed to provide fast and near-optimal solutions, trading off strict optimality for improved computational speed and scalability.

Our experimental analysis considered two distinct scenarios: IP-solvable instances (small scale)

and IP-unsolvable instances (large scale). In solvable scenarios, GA consistently approximated the optimal IP solutions with minimal deviation, achieving comparable scheduling quality while significantly outperforming IP in terms of computational efficiency. For large-scale, IP-unsolvable scenarios, GA demonstrated strong scalability by consistently producing good-quality solutions within fractions of a second, whereas the IP method consistently reached its computational time limit without identifying feasible solutions. These findings indicate that the GA method is well-suited for practical EV charging scheduling problems where computational resources and rapid solution generation are critical.

Despite these promising results, several limitations exist within our current modeling approach and experimentation framework. First, our study assumes an offline scheduling context, where all EVs' release times and processing durations are fully known in advance. However, real-world EV charging scenarios are inherently dynamic (online), characterized by uncertain and continuously arriving vehicles. This limits the direct applicability of our results without further adaptation to real-time scheduling environments. Second, the experimental distributions we used for generating arrival and charging times, although representative, are simplified models of real-world variability. A more comprehensive and realistic modeling approach would include richer stochastic or empirical data sets drawn directly from operational EV charging stations. Lastly, our current GA design does not explicitly incorporate constraints often seen in practical charging scenarios, such as user priorities and charger compatibility restrictions.

Future research directions include adapting and extending our GA method to an online scheduling environment, thereby accommodating uncertainty and dynamic EV arrivals. Incorporating additional realistic constraints and utilizing actual operational data would further enhance the practical applicability of the proposed scheduling framework. By addressing these limitations, we aim to bridge the gap between theoretical modeling and real-world application, ultimately improving the efficiency and user experience of EV charging infrastructure.

References

Balin, S. (2011). Non-identical parallel machine scheduling using genetic algorithm. *Expert Systems with Applications*, *38*(6), 6814–6821.

Denton, B. (2025). *Ioe543: Scheduling - lecuture 9* [Accessed: 2025-04-10]. https://umich.instructure.com/courses/745895/files/folder/Lectures?preview=39709068

Farhadi, P., Moghaddas-Tafreshi, S.-M., & and, A. S. (2024). A comprehensive review on stochastic modeling of electric vehicle charging load demand regarding various uncertainties. *Smart Science*, *12*(4), 679–714. https://doi.org/10.1080/23080477.2024.2381332

Huttunen, H. (2008). *Stochastic processes: Lecture notes* [Chapter 7: Poisson Processes, p. 13]. https://www.netlab.tkk.fi/opetus/s383143/kalvot/english.shtml

Louie, H. M. (2015). Probabilistic modeling and statistical analysis of aggregated electric vehicle charging station load. *Electric Power Components and Systems*, *43*(20), 2311–2324. https://doi.org/10.1080/15325008.2015.1080770

Min, L., & Cheng, W. (1999). A genetic algorithm for minimizing the makespan in the case of scheduling identical parallel machines. *Artificial Intelligence in Engineering*, *13*(4), 399–403. https://doi.org/https://doi.org/10.1016/S0954-1810(99)00021-7