Bus Charging Schedule Simulated Annealing with MILP Constraints

Alexander Brown

October 30, 2022

Contents

1 Introduction

This document outlines the simulated annealing (SA) approach to the bus charging scheduling problem utilizing Mixed Integer Linear Programming (MILP) constraints as the method of determining feasible charging schedules. The problem statement is as follows: given a set of routes for a fleet of Battery Electric Buses (BEB), generate an optimal charging schedule to minimize the consumption cost (amount of electricity used over a certain time) and the demand cost (rate at which electricity is being used) within the constraints that the buses must maintain sufficient charge to complete the working day and do not have any delays in their respective routes.

Simulated Annealing (SA) shall be introduced and utilized as a means of finding the global optimum. The SA algorithm shall be constrained by a set of Mixed Integer Linear Program (MILP) constraints derived from the Position Allocation Problem (PAP). These constraints are set in place to ensure validity of the proposed charging schedules. A set of objective functions describing consumption cost and demand cost, as stated above, shall be minimized to reduce power consumption and total cost of using the BEBs.

2 Problem Description

Given a set of bus arrivals to a charging station $i \in \{1, ..., I\} = \mathcal{I} \subset \mathcal{Z}$ with a set of chargers to be queued $q \in \{1, ..., Q\} = \mathcal{Q} \subset \mathcal{Z}$ where the bus is indicated by an identification number $b \in \{1, ..., B\} = \mathcal{B} \in \mathcal{Z}$. Each bus arrival, i, can be represented by the tuple: $(b_i, a_i, e_i, u_i, d_i, v_i, \eta_i, \xi_i)$, in which the ordered elements denote the bus identification number arrival time to the station, departure time from the station, initial charging time, charge end time, the charger queue for the bus to be placed into, and the initial State Of Charge (SOC).

It is assumed that each visit occurs over the time horizon T. The set of all arrivals is represented by the set $\mathbb{I} = \{(b_i, a_i, e_i, u_i, d_i, v_i, \eta_i, \xi_i) : b_i, \xi_i \in B, a_i, e_i, u_i, d_i \in T, v_i \in Q\}$. The concept of "arrivals" is derived from the PAP [7]. This idea of arrivals is useful in the sense that it is easy to describe the state of any arbitrary arrival; however, it also assumes that each arrival is unique (i.e. no two bus arrives twice) therefore a system must be put in place to track each bus over each arrival. That is why a bus identifier is placed in the tuple, and in that way each bus can be tracked over each arrival.

For each arrival a bus must be placed in a singular queue, $v_i \in Q$. The charger q is assumed to be either a fast or slow charger. The bus is not allowed to change queues mid-charge. The amount of time the bus is allowed to charge is dictated by the scheduled arrival time and required departure time, $[a_i, e_i]$. Although an arrival must be placed in a queue, if a bus does not require much charge, or none at all, partial charges, or no charging, is allowed. It is not allowed for the bus to charge over its battery capacity limit. The battery charging rate is modeled as linear, which remains accurate up to about an SOC of 80% charge [5].

Each bus arrival, with the exception of the last arrival for each bus, has a paired "route" that the bus must perform. This route, as one would expect, causes the bus to discharge by some certain amount. This paper assumes an average discharge over a period of time allowing an estimation to be calculated for each route, Δ_i. The charge supplied while at the station is assumed to supply enough charge for each route (battery charge does not deplete to zero) with an additional battery capacity percentage, m, acting as a safety factor.

The scheduler's task shall be to schedule the set of arrivals $\mathbb{I}$ to fulfill the minimum charge requirements over the time horizon T as well as minimize the demand cost as well as minimize over the consumption cost. The objective function and constraints are discussed in further detail in section Subsection 3.2.

3 Optimization Problem

This sections introduces the problem in the form of the objective function as well MILP constraints. The objective function is required to allow comparisons between candidate solutions. In the context of this formulation, the objective function is broken down into two major components as alluded in the introduction: consumption cost and demand cost. The constraints ensure that candidate solutions are in the feasible region. They are composed of a series of equations defined by decision variables which are unknown variables that are manipulated in the attempt to optimize the objective functions and input variables predefined input variables that are assumed to be known. Furthermore, the decision variables have components that are directly and indirectly manipulated. This will be further discussed in Subsubsection 3.1.2.

3.1 Parameter Definitions

This section defines the input variables and decision variables used by the system. The input variables are the parameters that are assumed to be known prior to optimizing the system. The decision variables are the values that the SA algorithm has the freedom to manipulate. The values produced by the SA algorithm will be interpreted as a candidate charging solution. This is further described in Section 4.

Table 1: Table of variables used in the paper.

Variable	**Description**
Input constants	
C	Penalty method gain factor
B	Number of buses in use
I	Number of total visits
$J(v, u, d)$	Objective function
K	Local search iteration amount
Q	Number of chargers
$\mathcal{T}$	Time horizon
T	Temperature
Input variables	
Δ_i	Discharge of visit over route i
α_b	Initial charge percentage time for bus b
β_i	Final charge percentage for bus i at the end of the time horizon
δ_i	Discharge rate for vehicle i per mile
ϵ_q	Cost of using charger q
κ_b	Battery capacity for bus b
ρ_i	Route distance after visit i
ξ_i	Value indicating the next index visit i will arrive
a_i	Arrival time of visit i
b_i	ID for bus visit i
e_i	Time visit i must exit the station
k	Local search iteration k
m	Minimum charge percentage allowed for each visit
r_q	Charge rate of charger q
Decision Variables	
η_i	Initial charge for visit i
ϕ_i	Binary term to enable/disable charge penalty for visit i
ψ_{ij}	Tracks spatial overlap for visit pair (i, j)
σ_{ij}	Tracks temporal overlap for visit pair (i, j)
d_i	Detach time from charger for visit i
$p_{dem}(t)$	Demand cost
s_i	Amount of time spent on charger for visit i (service time)
u_i	Initial charge time of visit i
v_i	Assigned queue for visit i

3.1.1 Input Variables

The input values of any MILP system are defined prior to the solving of the system. They define initial conditions, known state properties, etc. Roughly following the order in Table 1, each variable will be introduced.

Δ_i is the amount power required to complete the bus route after visit i. Because there is no route after the last visit, $\Delta_I = 0$. The discharge for visit i is defined by equation Equation 1.

$$\Delta_i = \delta_i * \rho_i \tag{1}$$

Where δ_i is the amount of energy consumed by the bus per mile and ρ_i is the route mileage after visit i. As discussed before, since there is no route after the last visit $\rho_I = 0$. α_b is the initial SOC percentage of bus b at the beginning of the working day. The initial SOC for bus b can be represented as

$$\eta_{i_0^b} = \alpha_b * \kappa_b. \tag{2}$$

Where κ_b is the battery capacity for bus b, $\eta_{i_0^b}$ is special notation that will temporarily be used to indicate the initial charge for bus b. η_i will be further discussed in Subsubsection 3.1.2. ϵ_q is the cost for assigning a charger to queue q. This parameter is utilized by the objective function and is further discussed in Subsection 3.2. ξ_i represents the next arrival index for bus b_i. In other words, given a set of bus visit IDs $b = \{1, 2, 3, 1\}$. Using a starting index of 1, $\xi_1 = 4$. a_i and e_i are the arrival and departure times of bus visit i to the station, respectively. k represents the local iteration search for the SA algorithm. This is further discussed in Section 4. Lastly, r_q represents the rate of charge for the charger in queue q. As will be discussed in Subsection 3.2, fast chargers and slow chargers relate to high and low costs, ϵ_q, respectively.

3.1.2 Decision Variables

Decision variables are the defined by the optimizer and are therefore unknown prior to running the optimization algorithm. In this case the optimizer is SA. Once SA has been ran and each of the decision variables have been specified and the fitness of the solution is defined by the objection functions outlined in Subsection 3.2 are determined. The variables will be broken into two sections: direct and indirect decision variables. Decision variables that are direct are values that the system has direct control over and indirect variables are those that are influenced by the direct.

Direct Decision Variables Decision variables that are direct are variables that can be immediately chosen by SA. The first two variables are u_i and d_i. They represent the initial and final charging times. These values must remain within range of the arrival time and departure time for visit i, $[a_i, e_i]$. The last direct decision variable is the queue that bus visit i can be placed in to charge, $v_i \in q$.

Indirect Decision Variables Indirect decision variables are variables that are dependent on direct decision variables. For example η_i is the initial charge for visit i. These variables are chained together per bus by using the bus identifier, b, and next index, ξ_i. The initial charges must be chained so that the battery charge can be calculated per bus as it is charged and discharged over each visit, $[u_i, d_i]$. ϕ_i is a boolean decision variable, $\phi_i \in \{0, 1\}$, that either enables or disables the charge penalty defined in Subsection 3.2. σ_{ij} and ψ_{ij} are used to indicate whether a visit pair (i, j) overlap the same space as show in Figure 1. These variables will be further elaborated on in Subsection 3.3. p_{dem} is the demand cost of the overall charging schedule. It is calculated at after all the decision variables have been assigned. This is further described in Subsection 3.2.

3.2 Objective Function

The objective function is used to compare the fitness of different candidate solutions against one another. This objective function takes in a set input variables and decision variables to calculate some value of measure. The calculated objective function value can either be maximized or minimized. The desired option is dependent on the problem to be solved as well as the formulation of said objective function. Let J represent the objective function. The objective function for this problem has four main considerations: charger assignment, consumption cost, demand cost, and sufficient charge.

Suppose the objective function is of the form min $J = AC(u_i, d_i, v_i, \eta_i) + PC(u_i, d_i, v_i)$. $AC(u_i, d_i, v_i, \eta_i)$ is the assignment cost, and $PC(u_i, d_i, v_i)$ is the power usage cost. The assignment cost represents the costs of assigning a bus to a particular queue as well as the chosen charging period, $[u_i, d_i]$ as shown in Equation 3.

$$AC(u_i, d_i, v_i, \eta_i) = \sum_{i=1}^{I} \epsilon_{v_i}(d_i - u_i) + \frac{1}{2}C\phi_i(\eta_i - m\kappa_i)^2 \tag{3}$$

Where $v_i \in q$ is the charger index, u_i is the initial charge time, d_i is the detach time for visit i, ψ_i is a binary decision variable, m is the minimum charge percentage allowed, κ_i is the battery capacity for visit i, and η_i is the initial charge for visit i. The first term in the summation represents the calculation of the cost for assigning a bus to queue q (i.e. cost of using the charger multiplied by the usage time). The second term is the penalty function that is either enabled or disabled by ϕ_i which is discussed in Subsection 3.3. This form is the most common form that penalty methods are found in [6]. Note that the variables ψ_i and η_i are both decision variables that are being multiplied together. This is called a bilinear term. Using a traditional MILP solver, this would require linearization [9]; however, because SA handles nonlinearities easily these bilinear terms will be ignored [8].

The power cost contains the demand cost and the consumption cost. It can be divided into two components: demand cost and the consumption cost. The demand cost quantifies the amount of power being used over a given period and adjusts the cost accordingly. The consumption cost calculates the total amount of power being consumed by the chargers. The power cost is shown in Equation 4. Note that the demand cost is written as a function. This is because it is calculated post generation of the candidate solution with no obvious MILP representation.

$$PC(u_i, d_i, v_i) = DemandCost(schedule) + \sum_{i=1}^{I} r[v_i](d_i - u_i) \tag{4}$$

As stated before, the demand cost is calculated based on 15 minute increments (0.25 hours). This cost is also referred to as the peak 15. The peak 15 is represented by Equation 5.

$$p_{15}(t) = 0.25 \int_{t-15}^{t} p(\tau)d\tau \tag{5}$$

Which represents the energy used over the last 15 minutes. Because worst case must be assumed to always ensure enough power is supplied, the maximum value found is retained as represented in Equation 6.

$$p_{max}(t) = \max_{\tau\in[0,t]} p_{15}(\tau) \tag{6}$$

Because the cost has a minimum threshold, a fixed minimum cost is introduced. In a similar manner as p_{max}, the maximum value is retained.

$$p_{dem}(t) = \max(p_{fix}, p_{max}(t))s_r \tag{7}$$

Where s_r is the demand rate. Equation 7, again, retains the largest p_{15} value with a starting fixed value of p_{fix}. To calculate this numerically, an integration algorithm is required to iteratively calculate the $p_{15}(t)$. In turn, $p_{dem}(T)$ can be defined. This process is defined in Algorithm 1.

```
Algorithm: DemandCost
Input: Candidate solution: (schedule)
Output: Demand cost: (p-dem)
begin
    p15 ← ∅;
    for dt ← 0 to T do
        Union(p15, Integrate(schedule,(dt,dt+0.25)))
    end
    p-old ← p-new ← p-dem ← p-fix;
    foreach element p in p15 do
        p-old ← p-new;
        p-new ← p;
        if p-new > p-old then
            p-dem ← p-new;
            p-old ← p-new;
        end
    end
    return p-dem
end
```

Algorithm 1: Algorithm to calculate the demand cost.

Where `schedule` is the set $\mathbb{I} = \{(b_i, a_i, e_i, u_i, d_i, v_i, \eta_i) : b_i \in B, a_i, e_i, u_i, d_i \in T, v_i \in Q\}$ and `p-fix` is the initial, fixed cost.

3.3 Constraints

Now that a method of calculating the fitness of a schedule has been established, a method for determining the feasibility of a schedule must be established. Feasible schedules require that the schedule maintain a certain list of properties. These properties are enforced by a set of constraints derived from the MILP PAP. The constraints must ensure no overlap temporally or spatially, receives enough charge to complete route after each visit i, bus visit i cannot be over charged, each visit, i, departs on time. The aforementioned constraints are shown in Equation 8.

$$u_i - d_j - (\sigma_{ij} - 1)T \geq 0 \tag{8a}$$

$$v_i - v_j - (\psi_{ij} - 1)Q \geq 0 \tag{8b}$$

$$\sigma_{ij} + \sigma_{ji} \leq 1 \tag{8c}$$

$$\psi_{ij} + \psi_{ji} \leq 1 \tag{8d}$$

$$\sigma_{ij} + \sigma_{ji} + \psi_{ij} + \psi_{ji} \geq 1 \tag{8e}$$

$$\Delta_i = \delta_i(a_{\xi_i} - d_i) \tag{8f}$$

$$\eta_{\xi_i} = \eta_i + r_{v_i}(d_i - u_i) - \Delta_i \tag{8g}$$

$$\kappa_i \geq \eta_i + r_{v_i}(d_i - u_i) \tag{8h}$$

$$\eta_i - m_{k_i} \leq T(1 - \phi_i) \tag{8i}$$

$$\eta_i - m_{k_i} < T\phi_i \tag{8j}$$

$$a_i \leq u_i \leq d_i \leq e_i \leq T \tag{8k}$$

Where the valid queue Equation 8a - Equation 8e define the spatial and temporal constraints of the system. These constraint enforce that the buses are placed in such a way that only one bus is allowed at a charger at any given time. Particularly Equation 8a determines if the initial charge time of visit i is after the final charge time of visit j. Similarly, Equation 8b determines if visit i or visit j are scheduled to be on the same queue. Equation 8c describes whether one of the visits come after the other temporally while Equation 8d describes if the chargers are placed in different queues. Equation 8e pulls all the previous constraints together and verifies that at least one of the conditions are true for each visit pair (i, j). The concept of the temporal and spatial constraints can be visualized by Figure 1. The y-axis represents the possible queues for a bus visit to be placed into and the x-axis represents the time that can be reserved for each visit. The shaded rectangles represent time that has been scheduled for each bus visit. The set of constraints Equation 8a - Equation 8e aim to ensure that these shaded rectangles never overlap. Equation 8f calculates the discharge for the route after visit i. Equation 8g calculates the initial charge for the next visit for bus b_i. Equation 8h ensures that the bus is not being over charged. Equation 8i and Equation 8j are used to enable and disable the penalty method in Equation 3. This is done by checking if the initial charge for visit i is greater than or equal to the minimum allowed charge. Equation 8k ensures the continuity of the times (i.e. the arrival time is less than the initial charge which is less than the detach time which is less than the time the bus exits the station and all must be less than the time horizon).

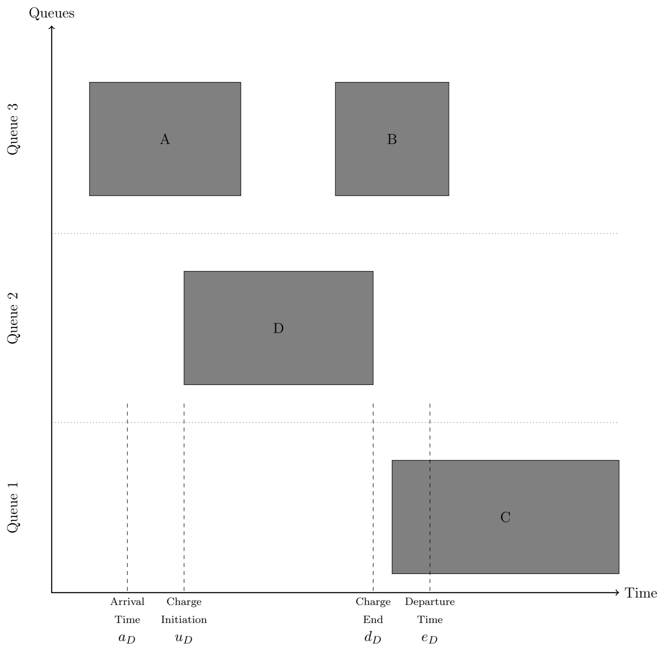

Figure 1: Visualization of the temporal and spatial aspect of the aspect of the schedule.

4 Simulated Annealing

SA is a local search (exploitation oriented) single-solution based (as compared to population based) meta-heuristic approach in which its main advantage is its simplicity both theoretically and in its implementation as well its inherit ability to overcome nonlinearities [1, 8]. This model is named after its analogized process where a crystalline solid is heated then allowed to cool very slowly until it achieves its most regular possible crystal lattice configuration [3]. There are five key components to SA: initial temperature, cooling schedule (temperature function), generation mechanism, acceptance criteria, local search iteration count (temperature change counter) [4].

4.1 Cooling Equation (Experimental)

The initial temperature and cooling schedule are used to regulate the speed at which the solution attempts to converge to the best known solution. When the temperature is high, SA encourages exploration. As it cools down (in accordance to the cooling schedule), it begins to encourage local exploitation of the solution [10, 3]. There are three basic types of cooling equations as shown in Figure 2 [4]. The different types merely dictate the rate at which we begin disallowing exploration. A linear cooling schedule is defined by Equation 9.

$$T[n] = T[n-1] - \Delta_0 \tag{9}$$

with $T[0] = T_0$ and $\Delta_0 = 1/2\ C^\circ$ in Figure 2. A geometric cooling schedule is mostly used in practice [4]. It is defined by Equation 10.

$$T[n] = \alpha T[n-1] \tag{10}$$

where $\alpha = 0.995$ in Figure 2. An Exponential cooling schedule is defined by the difference equation is define as Equation 11.

$$T[n] = e^{\beta} T[n-1] \tag{11}$$

where $\beta = 0.01$ in Figure 2. The initial temperature, T_0, in the case of Figure 2, is set to $500^\circ\ C$ and each schedule's final temperature is $1\ C^\circ$.

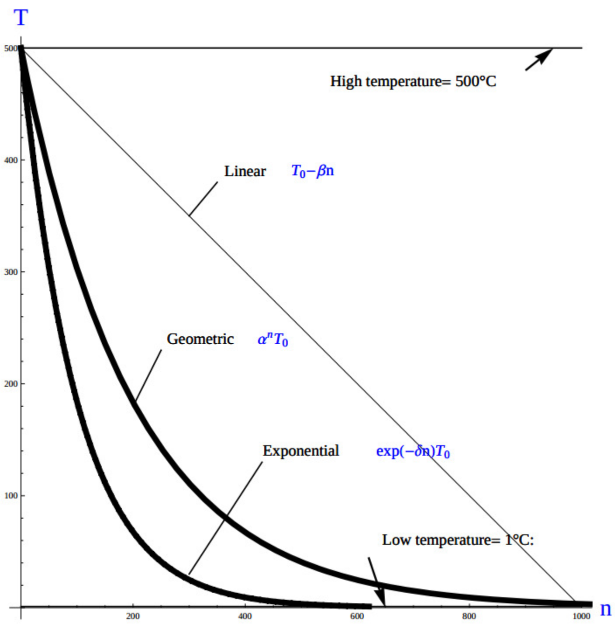

Figure 2: Cooling equations

4.2 Acceptance Criteria

Acceptance criteria describes the method to accept or reject a given candidate solution. In SA, if a new candidate solution is more fit than the currently stored solution it is always accepted as the new solution. However, within SA, worse candidate solutions may be accepted as the new solution. The probability of accepting the candidate solution is described by the function $\exp(\frac{J(x)-J(x')}{T})$ where $J(\cdot)$ is the objective functions described in Subsection 3.2. The probability of acceptance is a function of the cooling equation just described and difference of the current solution and a new candidate solution. Let $\Delta E \equiv J(x) - J(x')$ where x is the current solution and x' is the new candidate solution. The probability of acceptance of x' is defined by Equation 12 [4].

$$f(x, x', T) = \begin{cases} 1 & \Delta E > 0 \\ e^{-\frac{\Delta E}{T}} & \text{otherwise} \end{cases} \tag{12}$$

4.3 Generation Mechanisms

Generation mechanisms in SA are used to generate random solutions to propose to the optimizer, these are known as candidate solutions. In the case of the problem statement made in Section 2, five generation mechanism shall be used: new visit, slide visit, new charger, remove, new window. The purpose of each of these generators is to assign new visits to a charger, adjust a bus visits initial and final charge time within the same time frame/queue, remove a bus from a charger, and place a bus visit into a new time slot/queue. Each generator will be discussed in more detail in Subsubsection 4.3.2.

These generator mechanisms will in turn be utilized by three wrapper functions. The purpose of the route generation to create a set of bus route data to feed to the SA algorithm. Although, strictly speaking, is not a part of the SA algorithm. It is vital in specifying the initial conditions and "setting the stage" for the SA algorithm to solve. The schedule generation is to used create candidate solutions for SA to compare with other solutions, and the perturb schedule generator is used to take a candidate solution and alter it slightly in an attempt to fall into a global/local minimum.

4.3.1 Generator Input/Output

This section discusses in detail the expected inputs and output of each generator. It is important to discuss these parameters in order to have an understanding of the generating algorithms derived. The input consists of the bus visit index of interest, information about the current state of arrivals, $\mathbb{I}$, and the current state of the chargers' availability, $\mathbb{C}$. The output of each generator affects the tuple of decision variables $(v_i, u_i, d_i) \subset \mathbb{I}$.

Generator Input Each generator has the tuple input of $(i, \mathbb{I}, \mathbb{C})$ where i is the visit index, $\mathbb{I}_i$ is the tuple $(b_i, a_i, e_i, u_i, d_i, v_i, \eta_i, \xi_i)$ (Section 2), that describes the set of visits generated by the route generation algorithm (Section 4.3.3), and $\mathbb{C}$ is the set that describes the availability for all chargers $q \in \mathcal{Q}$. In other words, $\mathbb{C}$ defines the set of times when the chargers are not being utilized or are "inactive".

To derive $\mathbb{C}$, consider its inverse, $\mathbb{C}'$, which is the set of "active" time periods for each charger, $\mathbb{C}' = \bigcup\{\mathbb{C}'_q : q \in \mathcal{Q}\}$ where $\mathbb{C}'_q \subset \mathbb{C}'$ describes the active times for charger q. Focusing on an individual charger, consider $\mathbb{C}'_q$ before a schedule has been imposed upon it, $\mathbb{C}'_q \in \varnothing$. In other words, no buses have been assigned to be charged over the time period of $[u_i, d_i]$. After the scheduling process is complete, $\mathbb{C}'_q$ will have a set of active periods of the form $\mathbb{C}'_q \in \{[u_j, d_j] : j \in \mathcal{J}\}$ where $\mathcal{J} \subset \mathcal{I}$. For $\mathbb{C}'_q$ to be of value, its compliment is to be found, $\mathbb{C}_q$.

To determine the inverse of $\mathbb{C}'_q$, begin by noting $\mathbb{C}'_q \bigcap \{[u_j, d_j] : j \in \mathcal{J}\} = \varnothing$, in other words is said to be disjoint [2]. The inverse of a disjoint set can be found by the De Morgan Law as shown in Equation 13. Using Equation 13, the set of inactive periods can be written as $\mathbb{C}_q \equiv \bigcup\{[u_j, d_j]' : j \in \mathcal{J}\}$.

$$(A \cap B)' = A' \cup B' \tag{13}$$

Generator Output The output, $x'_i \equiv (v_i, u_i, d_i) \subset \mathbb{I}_i$ defines tuple of the chosen queue, initial charge time, and detach time from the generator, (v_i, u_i, d_i). The nature of SA implies that the generators have a sense of randomness. Because of that, some of the generators may have multiple choices for what x'_i may be. Let the set of candidates for the output be defined as $x'_i \in X'_i$.

4.3.2 Generators

This section describes and outlines the algorithm pool for the different generator types that are utilized in the wrapper functions. Note that to satisfy constraints, B extra idle queues that provide no power to the bus. Because of this, the set of queues is fully defined as $q \in \{1, ..., Q, Q+1, ..., Q+b\}$ where Q is the total amount of chargers and b is the bus ID. The use case for this is for when a bus is not to be placed on a charger, it will be placed in the queue, $v_i \in \{Q+1, ..., Q+b\}$, which will satisfy the constraints above while allowing the bus to be "set aside" while others charge.

New visit The new visit generator describes the process of moving bus b from the idle queue, $v_i \in \{Q+1, ..., Q+b\}$ to a valid charging queue, $v_i \in \{1, ..., Q\}$. Line 2 initializes the set of solutions to the empty set. Line 3 loops through each charger availability set and line 4 loops thorough

Algorithm: New Visit
Input: $(\mathbb{S})$
Output: x'_i

1 **begin**
2     $X'_i \leftarrow \varnothing$; /* Begin with the empty set */
3   **foreach** $\mathbb{C}_q \in \mathbb{C}$ **do** /* For set of availabile times for charger q */
4   **foreach** $C \in \mathbb{C}_q$ **do** /* For each inactive region in $\mathbb{C}_q$ */
5   **if** `findFreeTime`(C, (a_i, e_i)) $\notin \varnothing$ **then** /* If there is time available in C */
6                 $X'_i \cup x'_i$; /* Add x'_i to the set of candidates */
7   **end**
8   **end**
9   **end**
10   **return** $\mathcal{U}_{X'_i}$ /* Return a random candidate */
11 **end**

Algorithm 2: New visit algorithm

Where $\mathcal{U}_{\{.\}}$ is the discrete uniform distribution of a and b, `route-data` is the data generated in `RouteGeneration` (described in Section 4.3.3), and `charger-data` are the time intervals allocated to buses as described above. The algorithm to find free time is defined in Algorithm 3. `L` and `U` are the lower and upper bound of the time between scheduled times. The possible use cases are depicted in Figure 3.

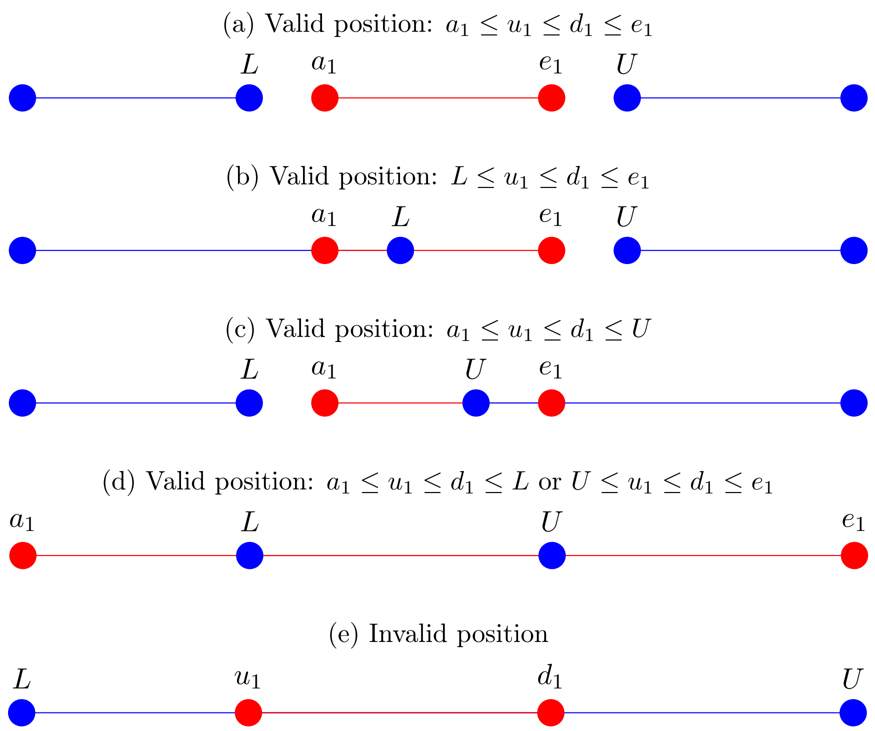

Figure 3: Outlines the different cases that requested time and charger allocated time can overlap

Algorithm: Find Free Time

Input: Lower and upper bound of available time and arrival and departure time for bus: (L, U, a, e)

Output: Tuple of initial and final charge times: (u, d)

```
begin
    if L ≤ a and U ≥ e then
        u ← U_[a,e];
        d ← U_[u,e];
    end
    else if L > a and U ≥ e then
        u ← U_[L,e];
        d ← U_[u,e];
    end
    else if L ≤ a and U < e then
        u ← U_[a,U];
        d ← U_[u,U];
    end
    else L > a and U < e
        u ← ∅;
        d ← U_[u,U];
    end
    return (u,d)
end
```

Algorithm 3: Find free time algorithm searches and returns the available time frames

Slide visit Slide visit is used for buses that have already been scheduled. Because $a_i \leq u_i \leq d_i \leq e_i$ (arrival time is less than initial charge time which is less than the detach time which is less than the time the bus exists the station), there may be some room to move u_i and d_i within the window $[a_i, e_i]$. Two new values, u_i and d_i are selected with a uniform distribution to satisfy $a_i \leq u_i \leq d_i \leq e_i$.

Algorithm: Slide Visit

Input: Visit index, route data, Charger data: (i, route-data, charger-data)

Output: x'_i: (v, u, d)

begin

  $a \leftarrow$ route-data[i].a;

  $e \leftarrow$ route-data[i].e;

  $u \leftarrow \mathcal{U}_{[a,e]}$;

  $d \leftarrow \mathcal{U}_{[u,e]}$;

  return *(v,d)*

end

Algorithm 4: Slide Visit Algorithm

New charger Similar to new visit, this generator moves a bus from one queue to another; however, the new charger generator moves a bus from one charger queue to another, $v_i \in \{0, .., Q\}$. A new charger will be selected at random with a uniform distribution.

Algorithm: New Charger

Input: Visit index, route data, Charger data: (i, route-data, charger-data)

Output: x'_i: (v, u, d)

begin

  $a \leftarrow$ route-data[i].a;

  $e \leftarrow$ route-data[i].e;

  $v \leftarrow$ route-data[i].v;

  valid-visit $\leftarrow \varnothing$;

  for $q \leftarrow 0$ **to** Q *and* $q \neq v$ **do**

    for *region* $\leftarrow$ **to** *q.free* **do**

      `Union(`*valid-visit,* `findFreeTime(`*region, (a,e)*`));`

    end

  end

  return $\mathcal{U}_{[valid-visit[0], valid-visit[length(valid-visit)-1]]}$

end

Algorithm 5: New Charger Algorithm

Remove The remove generator simply removes a bus from a charger queue and places it in its idle queue, $v_i \in \{Q, ..., Q+B\}$.

Algorithm: Remove

Input: Visit index, route data, Charger data: (i, route-data, charger-data)

Output: x'_i: (v, u, d)

begin

  $v \leftarrow Q + b$;

  $u \leftarrow$ route-data[i].u;

  $d \leftarrow$ route-data[i].d;

  return (v, u, d)

end

Algorithm 6: Remove algorithm

New Window New window is a combination of the remove and then new visit generators (Section 12 and Section 6). By this it is meant that current scheduled tuple (v_i, u_i, d_i) is removed and added back in as if it were a new visit.

Algorithm: New Window
Input: Visit index, route data, Charger data: (i, route-data, charger-data)
Output: x'_i: (v, u, d)

```
begin
    v ← route-data[i].v;
    u ← route-data[i].u;
    d ← route-data[i].d;
    (v, u, d) = Remove(v, u, d);
    (v, u, d) = NewVisit(v, u, d);
    return (v, u, d)
end
```

Algorithm 7: New window algorithm

4.3.3 Generator Wrappers

This section covers the algorithms utilized to select and execute different generation processes for the SA process. The generator wrappers are the method immediately called by SA. Each wrapper utilizes the generators previously described and returns either metadata about the bus routes or a new valid charger schedule.

Route Generation The objective of route generation is to create a set of metadata about bus routes given the information in Figure 5. Specifically, the objective is to generate I routes for B buses. Each visit will have an initial charge (specified for first visit only), arrival time, departure time, final charge (minimum allowed charge specified for finial visit only).

This is created by following the "GenerateSchedule" state in the state diagram found in Figure 4. In essence the logic is as follows: Generate B random numbers that add up to I visits (with a minimum amount of visits set for each bus). For each bus and for each visit, set a departure time that is between the range [min_{rest}, max_{rest}] (Figure 5), set the next arrival time to be $j \cdot \frac{T}{\text{J}}$ where j is the j^{th} visit for bus b and J is the total number of visits for bus b. Finally, calculate the amount of discharge from previous arrival to the departure time.

```
Algorithm: RouteGeneration
Input: Route YAML metadata: (mdata)
Output: Array of route events: (route-data)
begin
    while not schedule-created do
        arrival-new ← 0.0;
        arrival-old ← 0.0;
        departure-time ← 0.0;
        schedule-created ← false;
        foreach b ∈ B do
            foreach n ∈ J_b do
                arrival-old ← arrival-new;
                if j = J_b then
                    final-visit = true;
                end
                else
                    final-visit = false;
                end
                departure-time ← DepartureTime(arrival-old, final-visit);
                arrival-new ← current-visit*T/J_b;
                discharge ← discharge-rate*(next-arrival, depart-time) ;
                Union(route-data, (arrival-old, departure-time, discharge));
            end
        end
        schedule-created ← Feasible(route-data);
        SortByArrival(route-data);
    end
end
```

Algorithm 8: Route generation algorithm

Where `discharge-rate` is read from YAML data shown in Figure 5, the `Departure` algorithm is shown in Algorithm 9, and the `Feasible` method is used to determine if the generated schedule is valid (conditions covered in Subsection 3.3). This is done by attempting to generate a schedule that is in the solution space. This is further elaborated on later.

```
Algorithm: DepartureTime
Input: Previous arrival and final visit flag: (arrival-old and final-visit)
Output: Next departure time: (depart)
begin
    if final-visit then
        depart ← T;
    end
    else
        depart ← arrival-old + U[min-rest,max-rest];
    end
    return depart
end
```

Algorithm 9: Departure time algorithm

Schedule Generation The objective of this generator is to generate a candidate solution to the given schedule. To generate a candidate solution the generator is given the route schedule data that was previous generated. A bus is picked at random, $b \in B$, then a random route is picked for bus b. The new arrival generator is then utilized. This process is repeated for each visit. The state diagram is depicted in the state digram in Figure 6 and outlined in Algorithm 10.

Algorithm: ScheduleGeneration
Input: Route data: (route-data)
Output: Candidate charging schedule: (schedule)

```
begin
    schedule ← ∅;
    for i in I do
        bus ← U[0,B];
        visit ← U[0,I];
        Union(schedule,NewVisit((visit.a, visit.e)));
    end
    return schedule
end
```

Algorithm 10: Schedule generation algorithm

Where `schedule` is $\mathbb{I} = \{(b_i, a_i, e_i, u_i, d_i, v_i, \eta_i) : b_i \in B, a_i, e_i, u_i, d_i \in T, v_i \in Q\}$.

Perturb Schedule As described in SA, local searches are also employed to try and exploit a given solution [8]. The method that will be employed to exploit the given solution is as follows: pick a bus, pick a visit, pick a generator. This state diagram is depicted in Figure 7 and outlined in Algorithm 11.

Algorithm: PerturbSchedule
Input: Schedule candidate solution: (schedule)
Output: Perturbed schedule: (schedule)

```
begin
    for i in I do
        visit ← U[0,I];
        generator ← U[0,generator−count];
        schedule ← GeneratorCallback [generator](visit, route-data, charger-data);
    end
    return schedule
end
```

Algorithm 11: Perturb schedule algorithm

5 Optimization Algorithm

This final section combines the generation algorithms and the optimization problem into a single algorithm. The objective is to outline the SA process from start to finish. Algorithm 8 generates a set of bus routes utilizing the route metadata in Figure 5. The initial temperature and cooling schedule will be selected prior to execution and passed into the SA optimization algorithm. A new candidate solution will be generated. The candidate solution will be checked if it is feasible by using the equations from Subsection 3.3. For each step in the cooling schedule will have K iterations to attempt to find a local maxima. Each perturbation to the system is then compared to the current candidate solution. If the new candidate solution is better it is kept; however, if the candidate solution is worse, the solution may still be kept with a calculated probability as described in Subsection 4.2. This process is summarized in Algorithm 12.

Algorithm: SA PAP

Input: Bus route metadata: (file-path)

Output: Optimal charging schedule: (schedule)

```
begin
    T_0 ← InitTemp();
    T_schedule ← GetCoolingEquation();
    route-metadata ← LoadYaml(file-path);
    routes ← RouteGeneration(route-metadata);
    best-solution ← v ∈ ScheduleGeneration(routes);
    foreach T ∈ T_schedule(T_0) do
        candidate-solution ← ScheduleGeneration(routes);
        if InSolutionSpace(candidate-solution) then
            foreach k ∈ K do
                del-sol ← J(candidate-solution) - J(best-solution);
                if del-sol ≤ 0 then
                    best-solution ← candidate-solution;
                end
                else if del-sol ≥ 0 then
                    best-solution ← candidate-solution with probability exp(del-solτ_k);
                end
                schedule ← PerturbSchedule(schedule);
            end
        end
    end
end
```

Algorithm 12: Simulated annealing approach to the position allocation problem

References

[1] Michel Gendreau and Jean-Yves Potvin, editors. *Handbook of Metaheuristics*. Internationalseries in operation research & management science. Springer International Publishing, 3 edition, oct.

[2] Paul R. Halmos. Naive set theory. *Undergraduate Texts in Mathematics*, 1974.

[3] Darrall Henderson, Sheldon H. Jacobson, and Alan W. Johnson. The theory and practice of simulated annealing. In *International Series in Operations Research &amp: Management Science*, pages 287–319. Kluwer Academic Publishers.

[4] Andre A. Keller. *Multi-Objective Optimization In Theory and Practice II: Metaheuristic Algorithms*. BENTHAM SCIENCE PUBLISHERS, mar 2019.

[5] Jing-Quan Li. Battery-electric transit bus developments and operations: A review. *International Journal of Sustainable Transportation*, 10(3):157–169, 2016.

[6] David G. Luenberger and Yinyu Ye. Penalty and barrier methods. *Linear and Nonlinear Programming*, page 401–433, 2008.

[7] Ahad Javandoust Qarebagh, Farnaz Sabahi, and Dariush Nazarpour. Optimized scheduling for solving position allocation problem in electric vehicle charging stations. In *2019 27th Iranian Conference on Electrical Engineering (ICEE)*, pages 593–597, 2019.

[8] Jordan Radosavljevic. *Metaheuristic Optimization in Power Engineering*. Energy Engineering. Institution of Engineering and Technology, Stevenage, England, June 2018.

[9] Maria Analia Rodriguez and Aldo Vecchietti. A comparative assessment of linearization methods for bilinear models. *Computers and Chemical Engineering*, 48:218–233, 2013.

[10] R.A. Rutenbar. Simulated annealing algorithms: an overview. *IEEE Circuits and Devices Magazine*, 5(1):19–26, jan 1989.

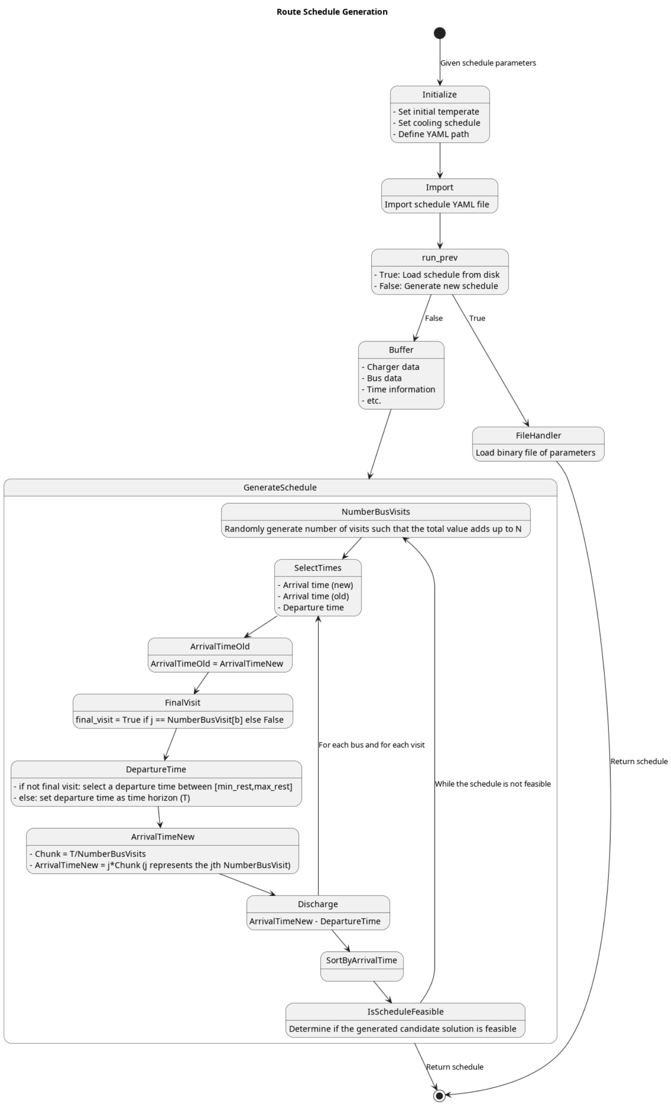

Figure 4: Route generation state diagram

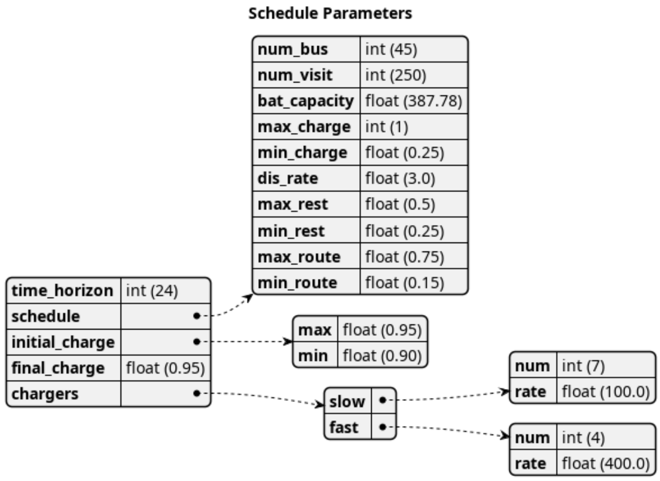

Figure 5: Route YAML file with example data

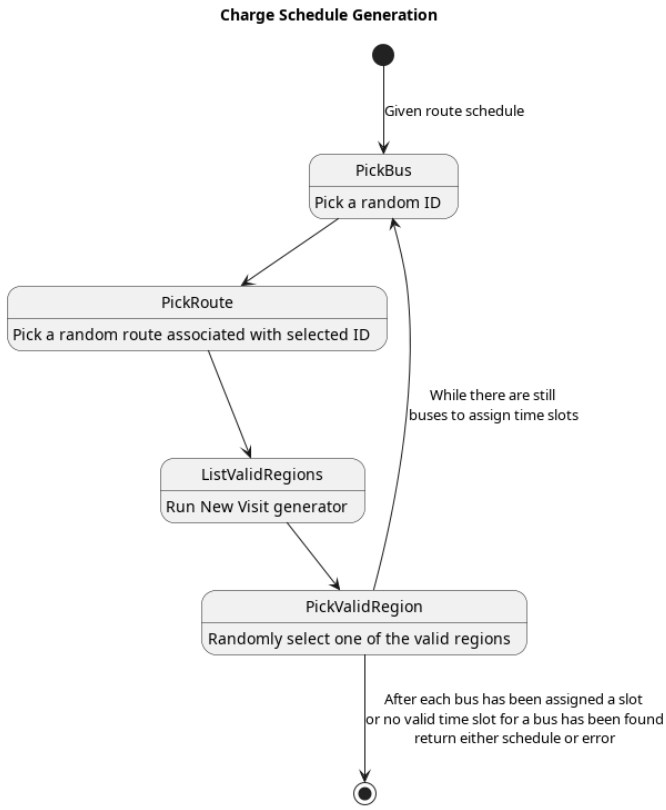

Figure 6: Charge solution state diagram

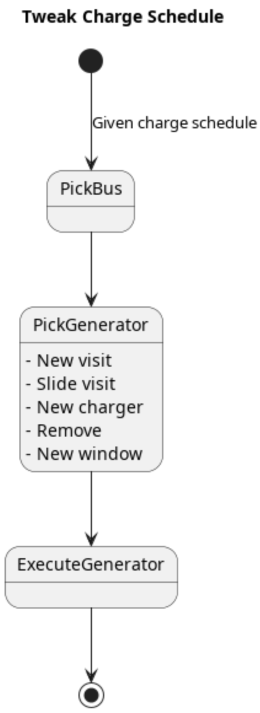

Figure 7: Solution perturb state diagram