Bus Charging Schedule Simulated Annealing with MILP Constraints

Alexander Brown

November 14, 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) and a set of fast and slow chargers, 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 optimizing. 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

It is assumed that there are a total of I visits to the station by B buses. There are a total of Q queues for the bus at the station. 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 $b \in \{1, ..., B\} = \mathcal{B} \subset \mathcal{Z}$. 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, b_i, arrival time to the station, a_i, departure time from the station, e_i, time to start charging, u_i, time to stop charging, d_i, the charger queue for the bus to be placed into, v_i, and the initial State Of Charge (SOC), η_i.

It is assumed that each visit occurs over the time horizon $T = \{t : t_0 \le t \le t_f\}$. The set of all arrivals is represented by the set $\mathbb{I} = \{\forall i \in \mathcal{I}\ (b_i, a_i, e_i, u_i, d_i, v_i, \eta_i, \xi_i) : b_i, \xi_i \in \mathcal{B}, a_i, e_i, u_i, d_i \in T, v_i \in \mathcal{Q}\}$. 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, although a bus may revisit the station multiple times, the model 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 or no charger at all. The bus is only allowed to visit one queue per visit. 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 a bus 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 where an estimated discharge is calculated for each route, Δ_i. The charge supplied while at the station is required 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

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 after visit 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	The next index bus b will arrive
a_i	Arrival time of visit i
b_i	ID for bus visit i
e_i	Time bus 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	Charge for the bus at the beginning of 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	Time to start charging for visit i
v_i	Assigned queue for visit i

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.

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 has no discharge due to a route. Let $\mathcal{J}_b \subset \mathcal{I}$ denote the set of visit indices for bus b, and let $\mathcal{J}_b^f$ denote the final visit index for bus b. Furthermore, let the $\mathcal{I}_f = \{x \in \mathcal{I}_f \subset \mathcal{I} : \forall b \in \mathcal{B}, x \in \mathcal{J}_b^f\}$. Therefore, the last visit for each bus has no discharge due to a bus route and can be written as $\Delta_{\mathcal{I}_f} = 0$. The discharge for visit all visits $i \in \{\mathcal{I} \setminus \mathcal{I}_f\}$, is defined by $\Delta_i = \delta_i \cdot \rho_i$ where δ_i is the amount of energy consumed by the bus per mile and $\rho_{\mathcal{I}_f}$ is the route mileage after visit i. δ_i is calculated using $\delta_i = \text{average mph} \cdot (a_{\xi_i - d_i})$. The average MPH is assumed to be 20 miles per hour.

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

$$\eta_{i_0^b} = \alpha_b \cdot \kappa_b, \tag{1}$$

where κ_b is the battery capacity for bus b, $\eta_{\mathcal{I}_0}$, $\mathcal{I}_0 = \{x \in \mathcal{I}_0 \subset \mathcal{I} : \forall b \in \mathcal{B}, x \in \mathcal{J}_b^0\}$, indicates 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 $\mathcal{J}_b = \{1, 2, 3, 1\}$, 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 run 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 $\forall i \in \mathcal{I}$. 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, $v_i \in \mathcal{Q}$.

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 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 input 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 $J = AC(u, d, v, \eta) + PC(u, d, v)$. $AC(u, d, v, \eta)$ is the assignment cost, and $PC(u, d, v)$ 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 2. $v_i \in q$ is the charger index, u_i is the initial charge time, d_i is the detach time for visit i, ϕ_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 charge for b_i when it arrives for visit i.

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

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. ϕ_i is enabled when the initial charge, η_i, is less than the allowed minimum charge, $m\kappa_{b_i}$. This is further discussed in Subsection 3.3. The form of the penalty function is the most common form that they are found in [6]. Note that the variables ϕ_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 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 consumption cost is merely the summation of all the energy being used over all the active periods for each charger in the time horizon. It is written as shown in Equation 3. r_{v_i} is the active charger for visit i and is multiplied by the time that the charger will be utilized, $d_i - u_i$.

$$\sum_{i=1}^{I} \left(r_{v_i}(d_i - u_i) \right) \tag{3}$$

The demand cost is calculated based on 15 minute increments (900 s). This cost is also referred to as the peak 15. The average power used over an arbitrary 15 minute interval is represented by Equation 4.

$$p_{15}(t) = 1/900 \int_{t-15}^{t} p(\tau) d\tau \tag{4}$$

Worst case must be assumed to always ensure enough power is supplied; therefore, the maximum value found is retained as represented in Equation 5.

$$p_{max}(t) = \max_{\tau \in [0,t]} p_{15}(\tau) \tag{5}$$

A fixed minimum cost is introduced before a specified threshold is exceeded. Let this fixed threshold be defined as p_{fix}. In a similar manner as p_{max}, the maximum value is retained. Furthermore, let s_r define the demand rate which has the units of $\frac{\$}{kW}$.

$$p_{dem}(t) = \max(p_{fix}, p_{max}(t))s_r \tag{6}$$

Equation 6, again, retains the largest p_{dem} value with a starting fixed value of p_{fix}. To write the total power demand at any discrete time, consider Equation 7. Let ω_h be the discrete power demand at time step h where $h \in \{1, 2, ..., H\} \subset \mathcal{Z}$ and $H = \frac{T}{900}$. For conciseness of notation we will abuse t_h to denote the time in discrete form (as opposed to t being continuous), let $dt_h = t_h - t_{h-1}$, and $\mathcal{H} = \{1, 2, ..., H\}$. Let ι_h be a binary decision variable that is enabled if charger $v_i \in \mathcal{Q}$ is enabled in the time frame $[t_{h-1}, t_h]$, where $t_h = 900 \cdot h$. Let the power usage of charger v_i be denoted as r_{v_i}.

$$\omega = \sum_h^H \sum_i^I \iota_h \cdot r_{v_i} \tag{7}$$

The average power can be rewritten as $p_{15} = \omega$, p_{dem} can be rewritten as $p_{dem} = \max_{h \in \mathcal{H}}(p_{15}^h)$, and finally p_d can be written as Equation 8.

$$p_d = \max_{h \in \mathcal{H}}(p_{fix}, p_{max})s_d \tag{8}$$

To write the power cost, Equation 3 and Equation 8 are superimposed to create Equation 9.

$$PC(u, d, v) = p_d + \sum_{i=1}^I \Big(r[v_i](d_i - u_i) \Big) \tag{9}$$

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 10.

$$u_i - d_j - (\sigma_{ij} - 1)T \geq 0 \tag{10a}$$
$$v_i - v_j - (\psi_{ij} - 1)Q \geq 0 \tag{10b}$$
$$\sigma_{ij} + \sigma_{ji} \leq 1 \tag{10c}$$
$$\psi_{ij} + \psi_{ji} \leq 1 \tag{10d}$$
$$\sigma_{ij} + \sigma_{ji} + \psi_{ij} + \psi_{ji} \geq 1 \tag{10e}$$
$$\Delta_i = \delta_i \rho_i \tag{10f}$$
$$\eta_{\xi_i} = \eta_i + r_{v_i}(d_i - u_i) - \Delta_i \tag{10g}$$

$$\kappa_i \geq \eta_i + r_{v_i}(d_i - u_i) \tag{10h}$$

$$\eta_i - m\kappa_{b_i} \leq T(1 - \phi_i) \tag{10i}$$

$$m_{k_i} - m\kappa_{b_i} < T\phi_i \tag{10j}$$

$$a_i \leq u_i \leq d_i \leq e_i \leq T \tag{10k}$$

$$u_i - dt_h \leq T\theta_h \tag{10l}$$

$$dt_h - u_i \leq T(1 - \theta_h) \tag{10m}$$

$$dt_h - d_i \leq T\mu_h \tag{10n}$$

$$d_i - dt_h \leq T(1 - \mu_h) \tag{10o}$$

$$\iota_h = \mu_h\theta_h \tag{10p}$$

Constraints Equation 10a-Equation 10e are the "queuing constraints". They are used to prevent overlapping in both spatially and temporally as shown in 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 and the queue allocated for each bus visit. The set of constraints Equation 10a - Equation 10e aim to ensure that these shaded rectangles never overlap.

Constraint Equation 10e states that the starting service time for bus j, u_j, must be greater than the starting time of bus i, u_i, plus its service time, s_i. The last term utilizes big-M notation to activate or deactivate the constraint. A value of $\sigma_{ij} = 1$ will activate the constraint to ensure that i is complete before j is allowed to begin being serviced. If $\sigma_{ij} = 0$, then the constraint is of the form $T + u_j + s_j > u_i$ rendering the constraint "inactive" because u_i cannot be larger than d_i. This effectively allows the charging windows of the vehicle to overlap.

Similarly, ϕ_{ij} determines whether the vehicles will be charging in the same queue. If $\phi_{ij} = 1$ then (Equation 10b) is rendered active and vehicle i and j must be charging in different queues. If $\phi_{ij} = 0$ then the constraint is deactivated and the vehicle queue assignments may be the same.

Equation 10f calculates the discharge for the route after visit i. Equation 10g calculates the initial charge for the next visit for bus b_i. Equation 10h ensures that the bus is not being over-charged. Equation 10i and Equation 10j are used to enable and disable the penalty method in Equation 2. This is done by checking if the initial charge for visit i is greater than or equal to the minimum allowed charge. Equation 10k 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).

The final set of constraints Equation 10l - Equation 10p set the rules for the decision variable ι_h. Equation 10l and Equation 10m ensure that the initial charge time for visit i, u_i, is greater than the discrete time, $900 \cdot h$, being checked. Similarly, Equation 10n and Equation 10o ensure that the final charge time, d_i, is less than the next discrete time step, dt_h.

4 Simulated Annealing

SA is a local search (exploitation oriented) single-solution based (as compared to population based) metaheuristic 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].

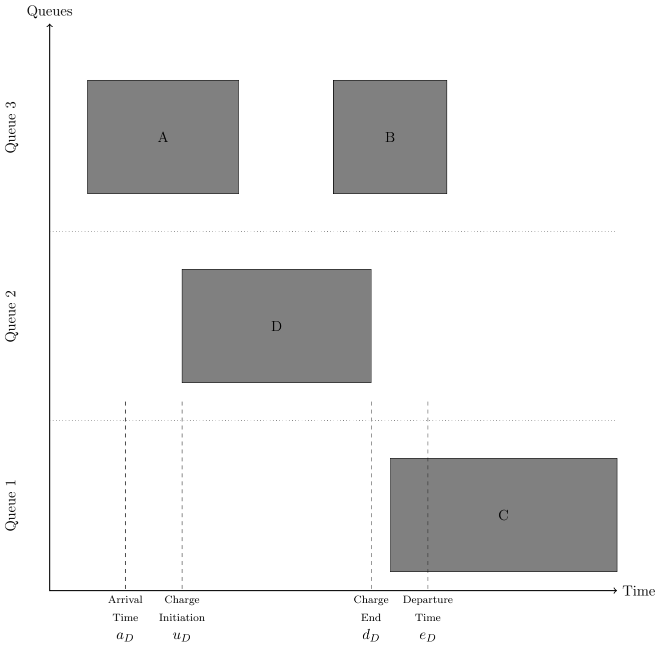

Figure 1: Visualization of the temporal and spatial aspect of the aspect of the schedule.

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 the algorithm progressively disallows exploration. A linear cooling schedule is defined by Equation 11.

$$T[n] = T[n-1] - \Delta_0 \tag{11}$$

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 12.

$$T[n] = \alpha T[n-1] \tag{12}$$

where $\alpha = 0.995$ in Figure 2. An Exponential cooling schedule is defined by the difference equation is define as Equation 13.

$$T[n] = e^{\beta} T[n-1] \tag{13}$$

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 $1\ C^\circ$.

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

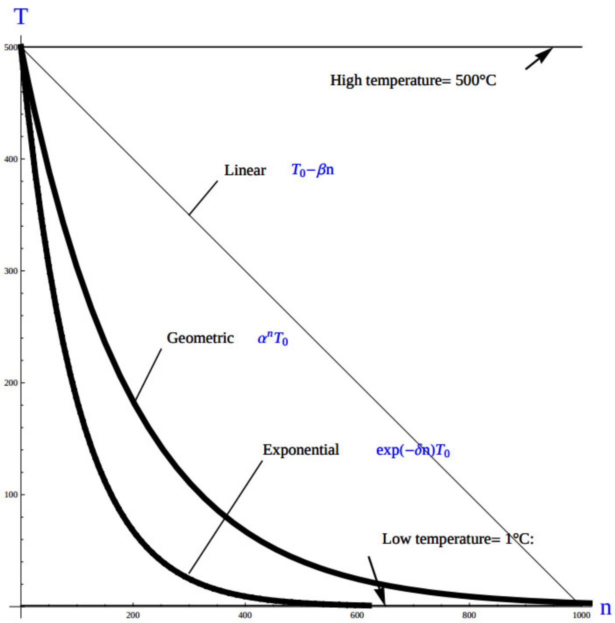

Figure 2: Cooling equations

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 14 [4].

$$f(x, x', T) = \begin{cases} 1 & \Delta E > 0 \\ e^{-\frac{\Delta E}{T}} & \text{otherwise} \end{cases} \tag{14}$$

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 chagrer 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 $\mathbb{C}'_q \in \{[u_j, d_j] : j \in \mathbb{J}\}$ where $\mathbb{J} \subset \mathcal{I}$. 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 $\mathbb{C}'_q \bigcap\{[u_j, d_j] : j \in \mathbb{J}\} = \varnothing$, 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 15. Using Equation 15, the set of inactive periods can be written as $\mathbb{C}_q \equiv \bigcup\{[u_j, d_j]' : j \in \mathbb{J}\}$. Let $\mathbb{S}$ denote the tuple $\mathbb{S} \equiv (i, \mathbb{I}, \mathbb{C})$.

$$(A \cap B)' = A' \cup B' \tag{15}$$

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, $v_i \in \{1, ..., Q\}$. Lines 2 through 4 extract the index, arrival time, and departure time for visit i. Note that in subsequent algorithms, these lines will be omitted. Line 5 initializes the set of solutions to the empty set. Line 6 loops through each charger availability set and line 7 loops thorough each of the available ranges, denoted as L and U for lower and upper free time. Line 8 checks if the range $[a_i, e_i] \subset [L, U]$, and lines 9 and 10 add it to the set of candidates. Line 14 chooses picks a candidate solution, $x'_i \subset X'_i$, with a discrete uniform distribution which is denoted by $\mathcal{U}_{\{.\}}$.

Algorithm: New Visit

Input: ($\mathbb{S}$)

Output: $\mathbb{I}$

begin

  $i \leftarrow \{i : i \in \mathbb{S}\}$; /* The index of the visit i */

  $a_i \leftarrow \{a_i \in \mathbb{I}_i : \mathbb{I}_i \in \mathbb{I} \subset \mathbb{S}\}$; /* Get the arrival time for visit i */

  $e_i \leftarrow \{e_i \in \mathbb{I}_i : \mathbb{I}_i \in \mathbb{I} \subset \mathbb{S}\}$; /* Get the departure time for visit i */

  $X'_i \leftarrow \varnothing$; /* Begin with the empty set */

  foreach $\mathbb{C}_q \in \mathbb{C}$ **do** /* For set of availabile times for charger q */

    foreach $C \in \mathbb{C}_q$ **do** /* For each inactive region in $\mathbb{C}_q$ */

      if `findFreeTime(`C, (a_i, e_i)`)` $\notin \varnothing$ **then** /* If there is time available in C */

        $x'_i \rightarrow$ `findFreeTime(`C, (a_i, e_i)`)` ; /* Assign x'_i to the found reigion */

        $X'_i \cup x'_i$; /* Add x'_i to the set of candidates */

      end

    end

  end

  return $\mathcal{U}_{X'_i}$ /* Return a random candidate */

end

Algorithm 1: New visit algorithm

The algorithm to find free time is defined in Algorithm 2. L and U are the lower and upper bound of the time between scheduled times. The possible use cases are depicted in Figure 3. In each case depicted by Figure 3, the red line shows the arrival and departure time for an arbitrary bus visit, i. The blue lines indicate reigons in which charger q is active. $C \in \mathbb{C}_q \subset \mathbb{C}$ represents one of the regions between the blue lines, $[L, U]$ which stand for the lower and upper portions of the regions, respectively. The output of 2 is a range for which the bus may be charged and the empty set if it cannet. As an example, consider a bus that is in the process of being scheduled and it encounters a situation similar to Figure 3c. That is, the only scheduling constraint is that the arrival time is before charger q is available to charge the bus. Therefore the bus must wait intil L before changer q may charge it. Furthermore, the range that u_i must be selected from is $[L, e]$.

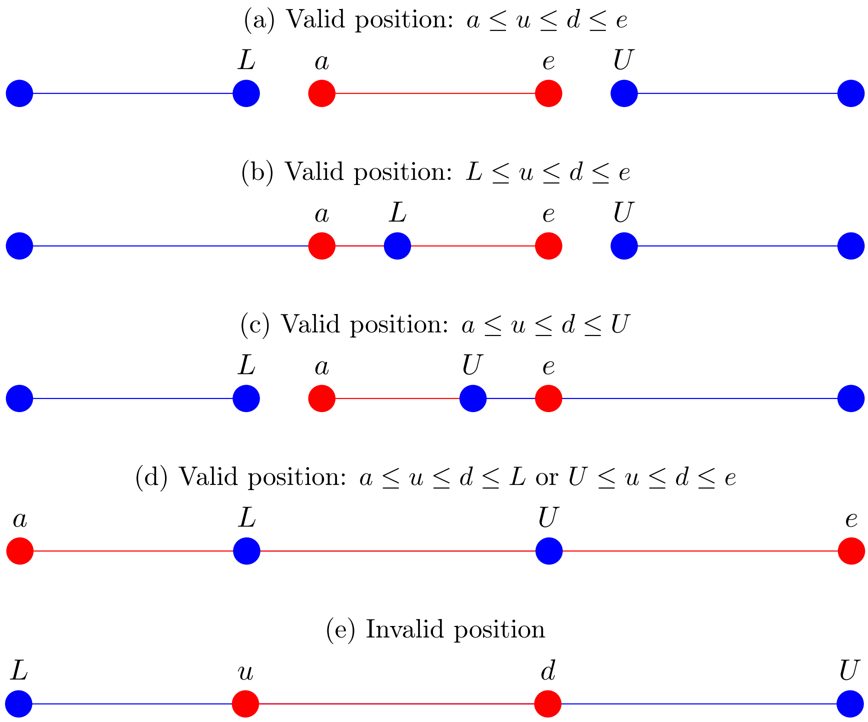

Figure 3: Outlines the different cases that requested time and charger allocated time can overlap

Algorithm: Find Free Time
Input: (L, U, a, e)
Output: (u, d)

begin
    if $L \leq a$ *and* $U \geq e$ **then**   /* If $L < a < e < U]$ (Figure 3a) */
        u $\leftarrow \mathcal{U}_{[a,e]}$;
        d $\leftarrow \mathcal{U}_{[u,e]}$;
    end
    else if $L > a$ *and* $U \geq e$ **then**   /* Else if $a < L < e < U$ (Figure 3b) */
        u $\leftarrow \mathcal{U}_{[L,e]}$;
        d $\leftarrow \mathcal{U}_{[u,e]}$;
    end
    else if $L \leq a$ *and* $U < e$ **then**   /* Else if $L < a < U < e$ (Figure 3c) */
        u $\leftarrow \mathcal{U}_{[a,U]}$;
        d $\leftarrow \mathcal{U}_{[u,U]}$;
    end
    else if $L > a$ *and* $U < e$ **then**   /* Else if $a \leq u \leq d \leq L$ or $U \leq a \leq d \leq e$ (Figure 3d) */
        u $\leftarrow \mathcal{U}_{[a,L],[U,e]}$;
        d $\leftarrow \mathcal{U}_{[u,L],[u,e]}$;
    end
    else   /* Otherwise the bus cannot be scheduled in this time frame (Figure 3e) */
        u $\leftarrow \varnothing$;
        d $\leftarrow \varnothing$;
    end
    return *(u,d)*
end

Algorithm 2: 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 of the constraint Equation 10k 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 the constraint $a_i \leq u_i \leq d_i \leq e_i$. Line 1 initializes the candidate set to be the original charge time frame. Line 3 loops through each opening for charger q, and line 4 checks if a new time frame $[u_i, d_i]$ is able to be scheduled. Lines 5 and 6 add that time frame to the set of candidates. Line 9 returns a candidate with a discrete uniform distribution.

Algorithm: Slide Visit
Input: $\mathbb{S}$
Output: $\mathbb{I}$

1 **begin**
2 $\quad X_i' \leftarrow \varnothing$; /* Begin with the empty set */
3 $\quad$ **foreach** $C \in \mathbb{C}_q$ **do** /* For each inactive region in $\mathbb{C}_q$ where $q = v_i$ */
4 $\quad\quad$ **if** `findFreeTime(`C, (a_i, e_i)`)` $\notin \varnothing$ **then** /* If there is time available in C */
5 $\quad\quad\quad x_i' \rightarrow$ `findFreeTime(`C, (a_i, e_i)`)` ; /* Assign x_i' to the found reigion */
6 $\quad\quad\quad X_i' \cup x_i'$; /* Add x_i' to the set of candidates */
7 $\quad\quad$ **end**
8 $\quad$ **end**
9 $\quad$ **return** $\mathcal{U}_{X_i'}$ /* Return a random candidate */
10 **end**

Algorithm 3: Slide Visit Algorithm

New charger The new charger generator takes a visit $\mathbb{I}_i$ and changes the charger it is on while maintianing the same charge time, $[u_i, d_i]$. Similarly to 1, the new candidate, x_i', must be checked before being added to the set X_i'. Line 2 initializes the candidates to the empty set. Line 3 and 4 loop though each charger availability set and each available range for charger q.

Algorithm: New Charger
Input: $\mathbb{S}$
Output: $\mathbb{I}$

1 **begin**
2 $\quad X_i' \leftarrow \varnothing$; /* Begin with the empty set */
3 $\quad$ **foreach** $\mathbb{C}_q \in \mathbb{C}$ **do** /* For set of availabile times for charger q */
4 $\quad\quad$ **foreach** $C \in \mathbb{C}_q$ **do** /* For each inactive region in $\mathbb{C}_q$ */
5 $\quad\quad\quad$ **if** $L \leq u$ ***and*** $U \geq e$ **then** /* If the charge time is within the region $[L, U]$ */
6 $\quad\quad\quad\quad X_i' \cup (q, u_i, d_i)$; /* Append the candidate to the set */
7 $\quad\quad\quad$ **end**
8 $\quad\quad$ **end**
9 $\quad$ **end**
10 $\quad$ **return** $\mathcal{U}_{X_i'}$ /* Return a random candidate */
11 **end**

Algorithm 4: 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: $\mathbb{S}$

Output: $\mathbb{I}$

```
begin
    return (Q + b, a_i, e_i)
end
```

Algorithm 5: Remove algorithm

New Window New window is a combination of the remove and then new visit generators (Section 11 and Section 3). 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: $\mathbb{S}$

Output: $\mathbb{I}$

```
begin
    x'_i ← Remove(v, u, d) ;        /* Remove visit i from its charger */
    x'_i ← NewVisit(x'_i) ;         /* Add visit i back in randomly */
    return (v, u, d)
end
```

Algorithm 6: 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 4. Specifically, the objective is to generate the input variables in $\mathbb{I}$ for I visits with B buses. Each visit will have an initial charge (specified for first visit only), arrival time, departure and time. In other words, $y_i \equiv (a_i, e_i) \subset \mathbb{I} \forall iin\mathcal{I}$ and each bus will be initialed with the SOC defined by Equation 1.

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 $[\text{min}_{\text{rest}}, \text{max}_{\text{rest}}]$ (Figure 4), set the next arrival time to be $j \cdot \frac{T}{\mathrm{J}}$ 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 the previous arrival to the next departure time as defined by Equation 1.

The metadata in Figure 4 will be denoted by $\mathbb{M}$ visually represents the YAML file used. This data contains all the parameters required to create a set of bus routes. Each of the cells will now be described from left to right, top to bottom. `time_horizon` represents the amount of time that the routes will be running for in hours. `schedule` contains the parameters that directly affect the generated set of routes. Some of the parameters in the YAML have already been defined in this paper, but go under a different name in the file. These parameters are: `num_bus` $\equiv B$, `num_visit` $\equiv I$, and `bat_capacity` $\equiv \kappa$. `max_charge` and `min_charge` represent the maximum and minimum percentages that the buses may be charged to. `max_rest` and `min_rest` represent the maximum and minimum times that buses may remain at the station, and `max_route` and `min_route` represent the maximum and minimum lengths that bus routes may be.

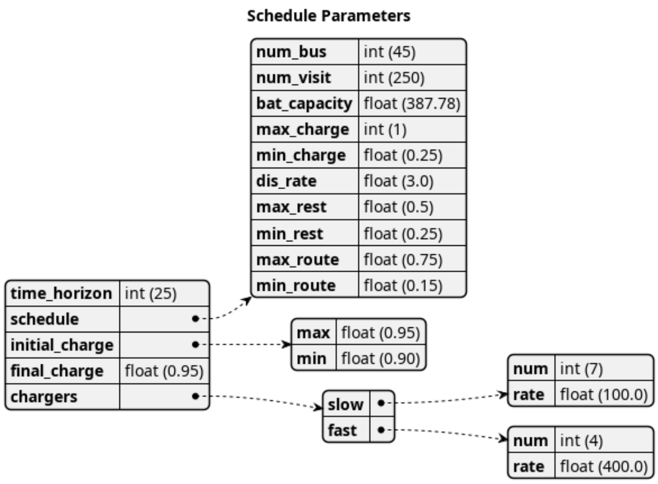

Figure 4: Route YAML file with example data

Algorithm: RouteGeneration

Input: $\mathbb{M}$

Output: $\mathbb{I}$

begin
    while ***not*** *schedule-created* **do**
        arrival-new $\leftarrow$ 0.0;
        arrival-old $\leftarrow$ 0.0;
        departure-time $\leftarrow$ 0.0;
        schedule-created $\leftarrow$ false;
        foreach $b \in B$ **do**
            foreach $n \in J_b$ **do**
                arrival-old $\leftarrow$ arrival-new;
                if $j = J_b$ **then**
                    final-visit $=$ true;
                end
                else
                    final-visit $=$ false;
                end
                departure-time $\leftarrow$ `DepartureTime(`*arrival-old, final-visit*`)`;
                arrival-new $\leftarrow$ current-visit$*\frac{T}{J_b}$;
                discharge $\leftarrow$ discharge-rate*(next-arrival, depart-time) ;
                `Union(`$\mathbb{I}$, *(arrival-old, departure-time, discharge)*`)`;
            end
        end
        schedule-created $\leftarrow$ `Feasible(`$\mathbb{I}$`)`;
        `SortByArrival(`$\mathbb{I}$`)`;
    end
    return $\mathbb{I}$
end

Algorithm 7: Route generation algorithm

The `Departure` algorithm is shown in Algorithm 8, and the `Feasible` method is used to determine if the generated schedule is valid (conditions covered in Subsection 3.3). This is done to generate a schedule that is in the solution space.

Algorithm: DepartureTime
Input: (arrival-old, final-visit)
Output: depart

```
begin
    if final-visit then
        depart ← T;
    end
    else
        depart ← arrival-old + U_[min-rest,max-rest];
    end
    return depart
end
```

Algorithm 8: 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 $\mathbb{I}$, a bus is picked at random, $b \in \mathcal{B}$, then a random visit is picked. The new visit generator (1) is then utilized. This process is repeated for each visit. This algorithm is summarized in 9.

Algorithm: ScheduleGeneration
Input: $\mathbb{I}$
Output: $\mathbb{I}$

```
begin
    𝕀 ← ∅;
    for i in I do
        b ← U_𝓑;
        i ← U_𝓘;
        NewVisit((visit.a, visit.e))
    end
    return 𝕀
end
```

Algorithm 9: Schedule generation algorithm

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. The algorithm is outlined in Algorithm 10.

Algorithm: PerturbSchedule
Input: $\mathbb{I}$
Output: $\mathbb{I}$

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

Algorithm 10: 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 7 generates a set of bus routes utilizing the route metadata in Figure 4. 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 11.

Algorithm: SA PAP

Input: Bus route metadata: (file-path)

Output: Optimal charging $\mathbb{I}$: ($\mathbb{I}$)

```
begin
    T_0 ← InitTemp();
    TI ← GetCoolingEquation();
    route-metadata ← LoadYaml(file-path);
    routes ← RouteGeneration(route-metadata);
    best-solution ← v ∈ ScheduleGeneration(routes);
    foreach T ∈ TI(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-sol τ_k);
                end
                𝕀 ← PerturbSchedule(schedule);
            end
        end
    end
end
```

Algorithm 11: 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.

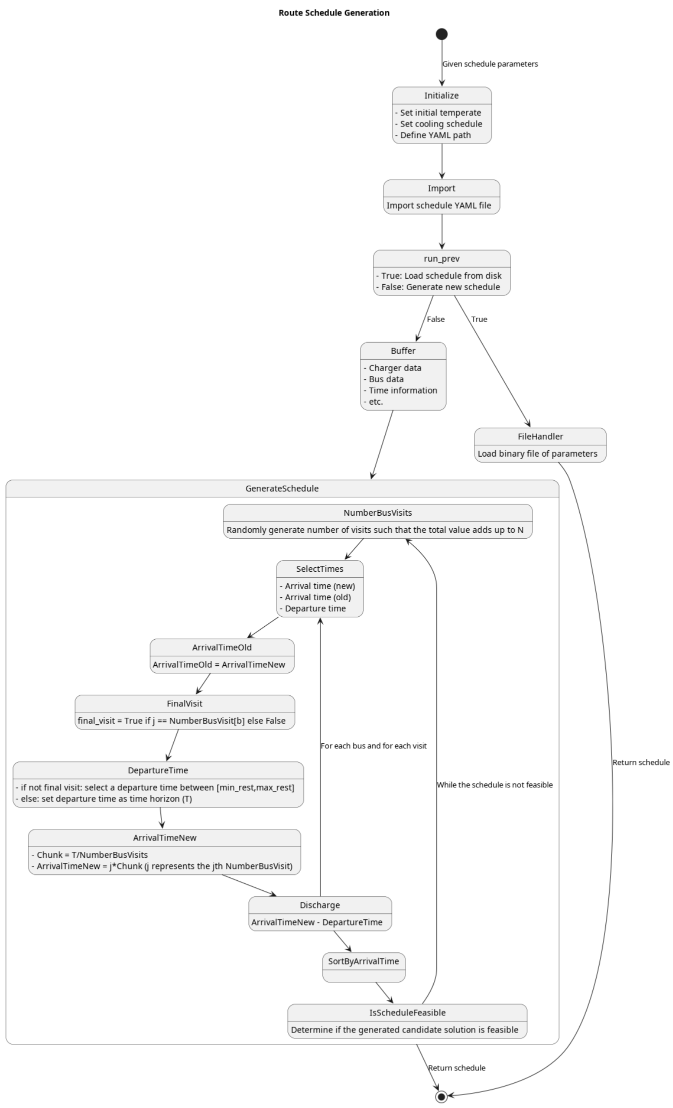

Figure 5: Route generation state diagram

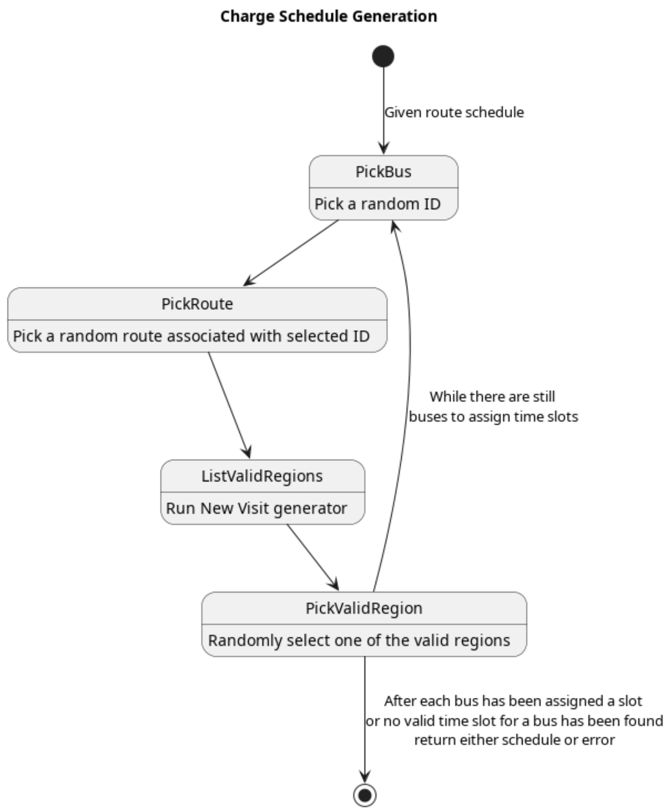

Figure 6: Charge solution state diagram

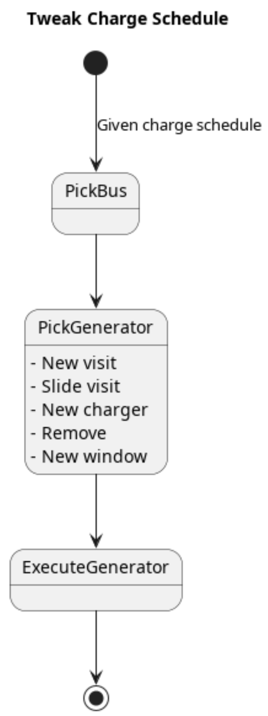

Figure 7: Solution perturb state diagram