CTARA Project Report

Optimization Algorithm for Scheduling Pumping Times for Farmers

Arjunn Pradeep, Devansh Satra

Contents

Executive Summary

This report details the development, implementation, and analysis of the optimization algorithm for scheduling pumping times in a community with a heavily overloaded transformer. The algorithm focuses on balancing load distribution, minimizing transformer failures, and ensuring optimal irrigation schedules. The project integrates real-world constraints, including farmer preferences, crop needs, and transformer capacity, and employs advanced programming techniques to achieve these goals.

1 Introduction

1.1 Context and Challenges

Agricultural communities relying on shared electricity infrastructure, such as transformers, face unique challenges during peak irrigation seasons. Overloaded transformers often result in failures and voltage drops, leading to inefficient pump operation. This project, part of the CTARA initiative, seeks to address these issues by optimizing the scheduling of irrigation pumps.

1.2 Objectives

The key objectives of the project are:

1. To reduce transformer load during peak irrigation times.
2. To prevent undervoltage conditions that can damage irrigation pumps.
3. To create a fair and efficient irrigation schedule that respects individual farmer requirements.

2 Problem Statement

The problem is to develop an algorithm that:

- Reduces load on a heavily overloaded transformer during peak irrigation seasons.
- Minimizes transformer failures caused by excessive simultaneous load.
- Ensures pumps operate efficiently without experiencing undervoltage.
- Considers diverse factors like water needs, crop types, irrigation methods, and transformer capacity.

3 Algorithm Overview

3.1 Overview of the Field Strength Based Algorithm (FSBA)

The FSBA is designed to optimize irrigation schedules for farmers by systematically balancing transformer load. Key steps of the algorithm are:

1. **Determine Irrigation Ranges:** Establish irrigation windows based on farmer-specific inputs, including crop type and soil requirements.
2. **Initial Slot Allocation:** Allocate initial irrigation slots randomly within defined ranges to create starting conditions for optimization.
3. **Net Power Calculation:** Calculate the total power load (field strength) for each day.
4. **Iterative Optimization:** Refine irrigation slots to minimize peak loads while meeting farmer-specific constraints.
5. **Scalability and Future Improvements:** Extend the algorithm with machine learning techniques for enhanced performance.

3.2 Key Components of the Code

1. **Schedule Initialization:** Prepares a dataframe with crop, pump, and irrigation details. Defines irrigation ranges based on crop-specific constraints.
2. **Power Distribution Analysis:** Tracks daily power usage to ensure even distribution. Identifies free slots for rescheduling irrigation events.
3. **Iterative Optimization:** Reschedules slots iteratively to minimize peak power usage. Builds blocks of irrigation events, ensuring gaps between consecutive events adhere to constraints.
4. **Algorithm Scalability:** Handles variable numbers of pumps and irrigation events. Efficiently processes large datasets using matrix operations.

3.3 Performance Metrics

Key metrics to evaluate the algorithm include:

- **Load Balancing:** Standard deviation of daily power usage.
- **Transformer Reliability:** Reduction in transformer failures.
- **Farmer Satisfaction:** Adherence to individual irrigation preferences.

4 Results and Analysis

4.1 Load Balancing

The algorithm successfully balanced the transformer load by distributing irrigation events across different time slots. Simulations show a significant reduction in daily load variance, ensuring transformer safety.

4.2 Pump Efficiency

By prioritizing voltage-stable slots, the algorithm prevented undervoltage conditions, ensuring optimal pump performance.

4.3 Scalability

Tests with larger datasets confirmed the algorithm's ability to handle increased farmer participation without compromising efficiency.

5 Conclusion

The Field Strength Based Algorithm developed in this project provides an effective solution to optimize irrigation schedules while addressing transformer load challenges. By balancing power distribution, preventing undervoltage, and accommodating diverse farmer requirements, the algorithm contributes to sustainable agricultural practices.
The project team remains committed to refining the algorithm, validating its performance, and exploring advanced techniques to enhance its applicability and effectiveness.

Appendix
Technical Documentation for Irrigation Scheduling Code

Arjunn Pradeep, Devansh Satra

Contents

1 Overview

This document provides a detailed explanation of the Python code used to generate an irrigation schedule for farmers. The code optimizes irrigation timings based on constraints like the number of irrigations, daytime or nighttime preference, power usage, and pump sharing between farmers.

2 Code Structure

The code is organized into multiple functions and a main driver function:

- `generate_schedule()`:
  - Processes input JSON files.
  - Initializes global variables.
  - Prepares the dataset for scheduling.
- `create_schedule()`:
  - Implements the main scheduling algorithm.
  - Assigns irrigation events to available time slots.
- Supporting functions:
  - `select_free_days()`: Filters available days for irrigation.
  - `append_days()`: Allocates days for irrigation events.
  - `check_consecutive_free_days()`: Verifies if a sequence of free days is available.
  - `remove_pumps_in_power_df()`: Clears pump usage for rescheduling.
  - `build_blocks()`: Generates possible schedules for crops.
  - `choose_optimal_block()`: Selects the most efficient irrigation block.

3 Global Variables

The following global variables are used throughout the code:

- `data, data2, df`: Store input data and processed farmer details.
- `num_days`: Total number of scheduling days.
- `schedule_data`: Stores scheduling results.
- `power_data, power_df`: Track daily power consumption.

4 Input and Data Preparation

4.1 Input Files

1. **jsonfarmers.json:** Contains farmer, crop, and pump details.
2. **TimeOfSupply.json:** Specifies daytime or nighttime supply availability.

4.2 Data Processing

`generate_schedule()` reads the input JSON files and converts the data into a `pandas` DataFrame, normalizing the crop and farmer details. Key processing steps include:

- Assigning default values for irrigation gaps if not provided.
- Calculating the irrigation width for each crop.
- Sorting crops based on multiple criteria.

5 Main Scheduling Algorithm

The `create_schedule()` function assigns irrigation blocks to crops by iterating through the `df` DataFrame and performing the following:

1. Initializes required variables such as `crop_id`, `days_required`, and `gap`.
2. Filters free days for irrigation using `select_free_days()`.
3. Identifies suitable blocks of consecutive days using `check_consecutive_free_days()`.
4. Allocates power consumption and pump usage for the selected days.
5. Appends results to `schedule_data`.

6 Utility Functions

6.1 `select_free_days()`

Filters available days for irrigation based on pump usage and nighttime preference.

6.2 `append_days()`

Updates power usage and marks the selected days as occupied by the corresponding pump.

6.3 `check_consecutive_free_days()`

Validates the availability of consecutive free days for an irrigation event.

6.4 `build_blocks()`

Constructs potential irrigation schedules based on constraints such as irrigation gap and total days required.

6.5 `choose_optimal_block()`

Evaluates all possible irrigation blocks and selects the one with minimal power usage.

7 Output

The final schedule is output as a JSON object containing:

- `CropID, PumpID`: Identifiers for crops and pumps.
- `BlockDates`: Scheduled irrigation days.
- `HorsePower, NightTime`: Total Power of pumps (in Horsepower) scheduled for each day, and Night Time Irrigation feasability of the crops.

8 Sample Usage

```
# Example JSON files
jsonfarmers = json.load(open('jsonfarmers.json'))
TimeOfSupply = json.load(open('TimeOfSupply.json'))

# Generate irrigation schedule
schedule_json = generate_schedule(jsonfarmers, TimeOfSupply)
print(schedule_json)
```

Listing 1: Sample Code Execution

9 Code Validation

9.1 Error Handling

The code includes safeguards for missing or invalid data:

- Default values for `IrrigationGap`.
- Validation of consecutive free days.

9.2 Optimization

The algorithm minimizes power consumption by dynamically evaluating irrigation blocks and pump schedules.