The Froggiest Frog

A Tomorrowland Line-Up Scheduling Problem.

A person standing in front of a castle

Description automatically generated

Figure 1 - Photo taken by me at Tomorrowland 2024 Ending Show.

Vlerick Business School,

Master in Data Analytics and Artificial Intelligence 2024-2025.

Made by Luna Geens.

For the course Decision Optimization given by Mario Vanhoucke,

as a final assignment due on Friday 20 Dec 2024.

# Contents

[0 Contents 2](#_Toc185869761)

[1 The Problem 3](#_Toc185869762)

[1.1 Contextual Problem Description 3](#_Toc185869763)

[1.2 Mathematical Problem Description 3](#_Toc185869764)

[1.2.1 Input Sets and Parameters 3](#_Toc185869765)

[1.2.2 Decision Variables 4](#_Toc185869766)

[1.2.3 Objective Function 4](#_Toc185869767)

[1.2.4 Constraints 5](#_Toc185869768)

[2 The Data 6](#_Toc185869769)

[2.1 Getting the Data 6](#_Toc185869770)

[2.2 Constructing Data Variables 6](#_Toc185869771)

[2.3 Preparing Data Variables for PuLP 6](#_Toc185869772)

[3 The Model 7](#_Toc185869773)

[3.1 Decision Variable 7](#_Toc185869774)

[3.2 Objective Function 8](#_Toc185869775)

[3.3 Constraints 8](#_Toc185869776)

[4 Appendix 11](#_Toc185869777)

[4.1 A. Full Initial Constraints 11](#_Toc185869778)

# The Problem

## Contextual Problem Description

Tomorrowland, as one of the largest and most prestigious electronic music festivals globally, attracts hundreds of thousands of attendees from over 200 countries, with tickets often selling out within minutes. Since its founding in 2005, Tomorrowland has expanded to host over 15 stages simultaneously, accommodating 800+ artists across multiple weekends, making efficient scheduling a necessity rather than a choice.

Efficient artist scheduling is not only a logistical challenge, but also a critical factor in enhancing the festival experience and business outcomes. Strategic artist placement directly influences festival success:

* Ticket Sales: Scheduling popular artists during prime slots drives higher attendance.
* Reputation: Delivering a seamless experience strengthens Tomorrowland's global standing as a premier festival.
* Cost Optimization: Efficient scheduling minimizes operational costs related to stage management, logistics, and artist coordination.
* Customer Satisfaction: Thoughtful scheduling can maximize audience engagement by aligning artist performances with attendee preferences and peak energy levels.

At Tomorrowland's scale, manual scheduling is impractical. By leveraging optimization-driven approaches, this research provides insights into optimizing festival scheduling, delivering tangible benefits to both attendees and organizers. In recent years, studies on festival optimization have increasingly relied on mathematical models (e.g., linear programming) to address these challenges.

## Mathematical Problem Description

### Input Sets and Parameters

* = Set of **stages**, where
* = Set of **artists**, where
* = Set of **sets performed** by artist , where .
* = Set of **days**, where .
* = Set of **starting points of time slots** available for stage during day , where .

Set-Specific Parameters. For each artist’s set

* : Time point in minutes of the beginning of the set . *Note that this is a decision variable in the problem, not a parameter!*
* : Day of the set . *Note that this is a decision variable in the problem, not a parameter!*
* Duration of set in minutes.

Stage-Specific Parameters: For each stage :

* : Opening time of stage on day .
* : Closing time of stage on day .
* : Clean-up time required after a performance on each stage (e.g., ).

Artist-Specific Parameters: For each artist :

* : Rest time required for each artist between two sets (e.g., ).
* : Maximum number of performances each artist can play per day. (e.g., ).

Popularity Parameters:

* : Popularity score for artist , where .
* : Popularity score for stage , where .
* Popularity score for day , where .
* : Popularity score indicating the popularity of the starting point of the timeslot , where .

### Decision Variables

The decision variable accounts for artists performing multiple sets:

Where:

* : Artist in the set of Artists.
* Set in the set of Sets performed by artist .
* : Stage in the set of Stages.
* Start time in the set of Start times for stage on day .
* : Day in the set of Days.

### Objective Function

The objective is to **maximize customer satisfaction** by considering artist, stage, day, and timeslot popularity. These weights are predefined inputs based on audience preferences or prior analysis.

Where:

* : Artist in the set of Artists.
* Set in the set of Sets performed by artist .
* : Stage in the set of Stages.
* Start time in the set of Start times for stage on day .
* : Day in the set of Days.
* : Popularity score for artist , where .
* : Popularity score for stage , where .
* Popularity score for day , where .
* : Popularity score indicating the popularity of the starting point of the timeslot , where .
* : Binary variable indicating if artist performs set on stage starting at time slot on day .

### Constraints

For the complete formulation of the constraints, I kindly refer to the Appendix A.

(1) Each Set is Scheduled Exactly Once.

Each set of artist must be scheduled at exactly one stage and time.

For :

(2) Each Set is Scheduled within Stage Availability.

The start and end times of each set of artist played on stage on a day must lie within the stage’s open and close times of that day .

For : Both conditions (1) and (2) are required to hold.

Start time of set on stage is after the stage’s opening time :

The set finishes () before the stage’s closing time :

(3) Sets on the Same Stage are Scheduled at a Different Time with a Clean-Up Time in between.

Two sets and on the same stage do not overlap in time. Additionally, a clean-up time is required between the two sets. This ensures that no two performances occur simultaneously on the same stage, and sufficient time is allowed for stage preparation between sets.

For ; ; ; and : At least one of the conditions (1) and (2) are required to hold.

1. Set must finish before starts: .
2. Set must finish before starts: .

(4) Sets of the Same Artist are Scheduled at a Different Time with a Rest Time in between.

Two sets and of the same artist do not overlap in time. Additionally, a rest time is required between the two sets. This ensures that no two performances occur simultaneously of the same artist, and sufficient time is allowed for artist preparation between sets.

For ; ; ; and : At least one of the conditions (1) and (2) are required to hold.

1. Set must finish before starts: .
2. Set must finish before starts: .

(5) No more than 2 Sets of an Artist are Scheduled on the Same Day.

Each artist is restricted to performing at most two sets on any single day. This ensures a manageable workload and a balanced schedule.

For ; :

# The Data

## Getting the Data

First, we need to collect some data on the artists. This process involves scraping Tomorrowland 2024's timetable directly from the web. The schedule is extracted from [an online page providing information on timetables of different festivals](https://festivalviewer.com/tomorrowland/lineup/2024), structured into a table format, and saved as a CSV file. This makes sure that next year, we can reuse the code.

Secondly, I have code that retrieves artist data, including their popularity score, from Spotify Developer Platform. The popularity score is a value between 0 and 100 that reflects an artist's global appeal, based on factors such as streaming activity and audience engagement. This metric provides a quantitative measure of an artist's popularity, which can be used to prioritize performances and optimize scheduling for maximum audience satisfaction.

## Constructing Data Variables

With the line-up information, I constructed several variables to analyze the data effectively. First, I calculated the start and end times, as well as the duration of each set. I assigned a unique ID to each festival day and a unique ID to each set (per artist, meaning most IDs were 1).

Next, I defined stage availability for each combination of day ID and stage name by determining the start time of the earliest set and the end time of the latest set. I also created a list, , containing all unique artists (those with one or more sets).

For these artists, I retrieved their popularity scores from Spotify and assigned a unique ID to each artist. This data, comprising the artist ID and popularity score, was stored in .

By combining the line-up and artist information, I was able to calculate the popularity of each day and stage. Normalized popularity scores were computed for both days and stages based on the aggregate popularity of the artists performing on each day and stage in the given line-up (not the optimized version). The normalization process scaled the scores to a consistent range of 0 to 100, ensuring comparability and clarity.

## Preparing Data Variables for PuLP

Sets. was defined as a list of unique stage names, and as a list of artist IDs. was a dictionary with artist IDs as keys and their corresponding lists of set IDs as values. represented a list of unique day IDs.

Defining (time slots per day and stage) posed some challenges. Initially, I chose a 5-minute granularity for the time slots, but this approach significantly increased runtime. To address this, I opted for hourly granularity instead, though this was a compromise. The time slots for each day and stage started based on the stage availability for that specific day and stage. Additionally, I realized that PuLP couldn’t handle timestamps directly, so I converted all timestamps into the number of minutes since midnight of the festival day. This process was complicated by the fact that some sets extended past midnight but still belonged to the previous festival day.

To assign popularity scores to the time slots, I avoided using because it would be disproportionately affected by the number of artists playing during a given time slot. Instead, I used a linearly increasing function ranging from 0 to 100, based on the total number of time slots for each day and stage.

Set-specific parameters. The start time of each set () and the day of each set () could be derived from the optimized values of the decision variables, so they were not explicitly defined. For the duration of each set (), I used a dictionary where the key was a tuple containing the artist ID and set ID, and the value was the duration in minutes.

Stage-specific parameters. To determine the opening and closing times for each stage on each day, I used the previously defined stage availability data and converted the timestamps into minutes since midnight. The resulting dictionaries, and , used tuples of (stage name, day ID) as keys, with the corresponding number of minutes as values. I set (the minimum gap between sets on the same stage) to 45 minutes, although this value was chosen arbitrarily.

Artist-specific parameters. Regarding artist-specific parameters, I assumed that an artist would perform a maximum of two sets per day, with a minimum gap of 120 minutes (2 hours) between sets.

Popularity parameters. These were all defined earlier on in the notebook.

# The Model

I encountered two major challenges while developing the model:

1. The dataset was too large, causing excessive runtimes even without any constraints.
2. Certain constraints were difficult to formulate.

To address these, I will first outline the initial approach and then explain the adjustments made to improve performance.

## Decision Variable

The decision variables were defined as binary, representing whether a specific artist performed on a particular stage, on a specific day, and at a specific time slot. These variables were created using the LpVariable.dicts function, generating a dictionary indexed by the 5-tuple (𝑎,𝑘,𝑠,𝑑,𝑡). Initially, the model included over 11 million decision variables.

To simplify the problem, I reduced the scope:

* I limited the dataset to only include performances on Weekend 1, focusing on Saturday and Sunday.
* I restricted the stages to four key ones: Mainstage, Library, Core, and Moosebar.

These changes reduced the number of decision variables to 220,320, significantly improving the model's feasibility and runtime.

## Objective Function

Secondly, our decision function that maximizes satisfaction is defined. The code defines the optimization problem "Tomorrowland\_Scheduling" as a linear programming model using the PuLP library, where the goal is to maximize satisfaction.

The objective function sums the weighted satisfaction scores for all possible combinations of artists, their sets, stages, days, and time slots. These values are then multiplied by the binary decision variable which indicates whether artist a performs set k on stage s, on day d, at time t. Finally, the problem is labeled "Maximize\_Satisfaction" to clarify its objective within the model.

## Constraints

For the final formulation of the constraints in Python, I kindly refer to appendix B.

(1) Each Set is Scheduled Exactly Once. This code defines a constraint in a linear programming model to ensure that each set is scheduled exactly once. For each artist and each set (the sets associated with artist , the constraint ensures that the sum of decision variables across all possible stages , days , and time slots equals 1. The constraint is added to the optimization problem with a unique label (f"Unique\_Assignment\_{a}\_{k}") for identification.

(2) Each Set is Scheduled within Stage Availability. To ensure that each set is scheduled within the predefined stage availability, we introduce the variable which represents the start time of a specific set for an artist . This variable is continuous and links to the binary decision variables, since . Additionally, the Big M technique helps link binary decision variables (which show if a set is scheduled) with continuous variables like ​ (the set’s start time). Without Big M, it would not be possible to conditionally enforce constraints based on whether or not a set is scheduled at a specific time. Here's how it works:

**Start Time Constraint:**

This constraint ensures that a set does not start before the stage becomes available. If any , this means that there is a set scheduled on that day on that stage. The term involving M becomes zero. In that case, the constraint enforces that the set’s start time must be greater than or equal to the stage opening time. When , the term involving Big M becomes very large, effectively "disabling" the constraint for that stage, day, and time slot.

**End Time Constraint:**

This constraint ensures that a set does not extend beyond the stage’s closing time. Similar to the start time constraint, when , it enforces the relationship between ​, the set duration, and the stage’s closing time. If , the Big M term ensures the constraint does not apply.

The value of Big M is chosen as an upper bound that is guaranteed to always hold true when the binary variable is 0. For this model, Big M is defined as the maximum possible duration of the entire festival day, calculated as the difference between the earliest stage opening time and the latest stage closing time, plus the longest possible duration of any set.

(3) Sets on the Same Stage are Scheduled at a Different Time with a Clean-Up Time in between.

Initially, I developed code to ensure proper scheduling of sets by precomputing constraints that defined the order in which sets were performed on specific stages and days. This involved identifying all sets scheduled for each stage-day combination and generating binary decision variables (**y**) for every pair of sets. These variables indicated whether one set finished before another, enabling the model to enforce proper sequencing. For each pair of sets, two constraints were created: one to ensure the first set finished before the second (active when y=1) and another to ensure the opposite order (active when y=0). A Big M constant was used to conditionally activate these constraints based on the values of the binary variables.

However, this method had significant limitations. The large number of possible pairwise combinations for sets on a single stage and day resulted in exponential growth in the number of constraints, dramatically increasing computational complexity and memory usage. Additionally, precomputing and storing **y** variables for all stage-day combinations was resource-intensive, especially for larger datasets with many stages, days, and sets. While parallel processing using ProcessPoolExecutor improved efficiency by handling each stage-day combination independently, the method remained constrained by available computing power and became inefficient when the number of tasks was small compared to the processing threads.

To address these challenges, I implemented a simplified alternative. The new constraint ensures that at most one set is assigned to each time slot for a given stage and day, replacing the earlier method of pairwise constraints. This approach eliminated the need to precompute binary variables (**y**) for every pair of sets and significantly reduced the number of constraints by introducing a single constraint per time slot instead of exponential combinations. However, this simplified method assumes that each set occupies exactly one time slot (1 hour), disregarding clean-up times and variations in set durations. While less precise, this approach is far more computationally efficient and practical for large-scale scheduling.

(4) Sets of the Same Artist are Scheduled at a Different Time with a Rest Time in between.

I had the same problem as with constraint (3). In the end, I ended up just ensuring that no artist is scheduled to perform more than one set at the same time on the same day. It aggregates all possible time slots across all stages for a given day and enforces that the sum of the binary decision variables for an artist's sets at any given time slot does not exceed 1, preventing overlapping performances by the same artist. However, this approach does not account for variations in set durations or the required rest time between an artist's performances.

(5) No more than 2 Sets of an Artist are Scheduled on the Same Day.

This constraint ensures that artists with more than two sets are limited to a maximum of two performances per day. For each day, it sums the decision variables for all the artist's sets across all stages and time slots and enforces that the total does not exceed 2.

# The Results

To improve the performance, I tested both Highs and Pulp as the default solvers, ultimately producing the time schedule outlined in Appendix C. While the current approach yielded a functional schedule, there are notable limitations that should be addressed in future iterations.

Firstly, the absence of genre classification per artist limits the system’s ability to optimize the alignment between artist performances and stage themes. Additionally, the current model does not account for artist collaboration preferences, which could help enhance the overall festival experience by scheduling complementary acts sequentially or on nearby stages. Technical requirements for performances, such as specific audio-visual setups, are also omitted, which may lead to logistical inefficiencies. Another limitation is the lack of audience flow management—ensuring that popular artists are not scheduled on opposite sides of the venue simultaneously or in a way that causes excessive crowding.

Moreover, artist exclusivity has not been considered; assigning unique slots to high-demand artists could elevate the festival’s appeal and drive audience interest. Lastly, the constraints in the current model were simplified for computational feasibility, which may not fully capture the nuances of operational and logistical realities, such as dynamic rest periods or variable stage preparation times.

In the future, I plan to address these limitations by incorporating genre-based assignments for artists and ensuring better thematic matches with stages. I also aim to refine the model by including collaboration preferences, technical requirements, audience flow optimization, and artist exclusivity as key factors. These improvements will not only make the scheduling process more efficient but also elevate the overall festival experience for attendees and performers alike.

# Appendix

## A. Full Initial Constraints

(1) Each Set is Scheduled Exactly Once.

Each set of artist must be scheduled at exactly one stage and time.

For :

Where:

* : Artist in the set of Artists.
* Set in the set of Sets performed by artist .
* : Stage in the set of Stages.
* Start time in the set of Start times for stage on day .
* : Day in the set of Days.
* : Binary variable indicating if artist performs set on stage starting at time slot on day .

(2) Each Set is Scheduled within Stage Availability.

The start and end times of each set of artist played on stage on a day must lie within the stage’s open and close times of that day .

For : Both conditions (1) and (2) are required to hold.

Start time of set on stage is after the stage’s opening time :

The set finishes () before the stage’s closing time :

Where:

* : Artist in the set of Artists.
* Set in the set of Sets performed by artist .
* : Stage in the set of Stages.
* : Day in the set of Days.
* : Time point in minutes of the beginning of the set .
* : Duration of set in minutes.
* : Opening time of stage on day .
* : Closing time of stage on day .

(3) Sets on the Same Stage are Scheduled at a Different Time with a Clean-Up Time in between.

Two sets and on the same stage do not overlap in time. Additionally, a clean-up time is required between the two sets. This ensures that no two performances occur simultaneously on the same stage, and sufficient time is allowed for stage preparation between sets.

For ; ; ; and : At least one of the conditions (1) and (2) are required to hold.

1. Set must finish before starts: .
2. Set must finish before starts: .

Where:

* : Artist in the set of Artists.
* Set in the set of Sets performed by artist .
* : Stage in the set of Stages.
* : Day in the set of Days.
* : Time point in minutes of the beginning of the set .
* : Duration of set $ k $ in minutes.
* : Clean-up Time .

(4) Sets of the Same Artist are Scheduled at a Different Time with a Rest Time in between.

Two sets and of the same artist do not overlap in time. Additionally, a rest time is required between the two sets. This ensures that no two performances occur simultaneously of the same artist, and sufficient time is allowed for artist preparation between sets.

For ; ; ; and : At least one of the conditions (1) and (2) are required to hold.

1. Set must finish before starts: .
2. Set must finish before starts: .

Where:

* : Artist in the set of Artists.
* Set in the set of Sets performed by artist .
* : Stage in the set of Stages.
* : Day in the set of Days.
* : Time point in minutes of the beginning of the set .
* : Duration of set $ k $ in minutes.
* : Rest Time .

(5) No more than 2 Sets of an Artist are Scheduled on the Same Day.

Each artist is restricted to performing at most two sets on any single day. This ensures a manageable workload and a balanced schedule.

For ; :

Where:

* : Artist in the set of Artists.
* Set in the set of Sets performed by artist .
* : Stage in the set of Stages.
* Start time in the set of Start times for stage on day .
* : Day in the set of Days.
* : Binary variable indicating if artist performs set on stage starting at time slot on day .

## B. Final Constraints in Python

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

## C. Results

A screenshot of a computer

Description automatically generated