Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Posgrado en Optimización

Energy-Efficient Train Timetables

Discrete Optimization Challenge

Rodrigo Alexander Castro Campos* Sergio Luis Pérez Pérez†
Gualberto Vazquez Casas‡ Francisco Javier Zaragoza Martínez§

July 2015

Team Optimixtli

*Doctoral Student, Graduate Program in Optimization, UAM Azcapotzalco.
†Doctoral Student, Graduate Program in Optimization, UAM Azcapotzalco.
‡Masters Student, Graduate Program in Optimization, UAM Azcapotzalco.
§Full Professor, Graduate Advisor, Systems Department, UAM Azcapotzalco.

This is a report of *Team Optimixtli*'s participation in the Discrete Optimization Challenge organized by the Friedrich-Alexander-Universität Erlangen-Nürnberg as part of their Open Research Challenge.

We describe a mixed integer program formulation corresponding to the optimization challenge, the optimal results obtained for some instances using an MIP solver, and the main idea we used to give good solutions (and improve them) for the rest of the instances.

All members of *Team Optimixtli* participate in the Graduate Program in Optimization at Universidad Autónoma Metropolitana Azcapotzalco, a public university in Mexico.

Postscript: If you want a short version of the report, see Sections 2.2, 2.3, 2.4, 4.1, and the first page of Chapter 5.

Contents

Overview

This is a report of *Team Optimixtli*'s participation in the *Discrete Optimization Challenge: Energy-Efficient Train Timetables*. This competition is part of the Open Research Challenge organized by the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) in Germany.

Chapter 1 is a verbatim copy of the challenge as found on FAU Open Research Challenge's website, it is included here only for completeness and therefore it can be skipped. Chapter 2 describes an exact mixed integer programming formulation for the optimization problem, the process of transforming the given data into a suitable format for a MIP solver, the instances, and the results (either optimal or failures) obtained. Chapter 3 discusses several proposals of how to obtain good solutions in those cases where the MIP solver failed. Chapter 4 describes our main proposal (which we call the *constant granularity* mixed integer programming formulation), the obtained solutions, and the further improvements. Chapter 5 displays our best obtained results.

The Appendices contain all the code written to process the data, generate mixed integer programs, format solutions, and interface with the MIP solver.

1. The Challenge

Railway traffic is the biggest individual electricity consumer in Germany. It amounts to 2% of the country's total electricity usage, which equals the consumption of the city of Berlin. However, the annual electricity cost (1 billion Euros per year) is not only determined by the total amount of energy used. Electricity contracts of big customers normally also incorporate a price component that is dependent on the highest power value drawn within the billing period, i.e. it depends on the timely distribution of the energy usage. This power component accounts for up to 20% of the energy bill.

In the FAU Open Research Challenge "Energy-Efficient Train Timetables", the aim is to optimize the timetables of railway traffic in order to avoid high peaks in power consumption as they lead to high electricity costs. This can be done by desynchronizing too many simultaneous departures as well as synchronizing accelerating trains with braking trains to make better use of the recuperated braking energy. Starting with a raw timetable (before its publication), the task is to shift the departures of the trains from the stations within small time intervals (several minutes) to come to the final optimized timetable.

1.1. The Task

Your goal is to compute optimal timetables with respect to the above goal. To this end, you are given problem instances featuring the following data:

1. A railway network in the form of a directed multi-graph.
2. The planning horizon under consideration.
3. A set of trains to schedule, and for each such train:
   a) the intermediate stations and tracks traveled,
   b) feasible departure intervals for each departure from a station,
   c) minimum stopping times in the stations (for getting on and off),
   d) travel times for the tracks,
   e) power profiles for each track traveled , i.e. a time-power-curve.
4. For each track in the network, the order in which it is passed by which trains.

5. The necessary safety distances.

The task is to ensure a favorable system-wide power load over the given planning horizon to minimize the power component of the electricity bill. The system-wide power load is determined by summing up all the individual power profiles of the trains under consideration. Obviously, this total power profile changes when shifting the trains in time. To minimize the power cost of the railway system, you have to minimize the average power drawn by all trains together over any consecutive 15-minute interval within the planning horizon.

To achieve this, your task is to come up with models and algorithms to optimize the departure times of the trains from the stations, keeping to the following requirements:

1. Each train travels the stations and tracks in the order given in data.
2. Each train departure is within the feasible interval for the corresponding station.
3. The minimum stopping time at each station is respected.
4. The trains pass the tracks in the given order and respect the safety distances.
5. Passenger connections at the stations are respected.

The winning team for a given problem instance is the one with the lowest 15-minute average of power consumption over the planning horizon. This value is summed up over the 10 instances given in the contest. To win the whole contest, you have to be the team with the lowest overall value over these 10 instances.

1.2. Description of the Optimization Problem

The degree of freedom in this optimization problem are the departure times of the trains from the stations. As a part of each problem instance (format description see readme file), you are given a series of train runs. Each of them consists of a series of legs, i.e. journeys between two consecutive stations at which the train stops. You are to adapt the current departure times to form a new timetable which has to respect several side constraints:

Maximal deviation from the current timetable: Each leg has an earliest and a latest possible departure time. The new departure has to lie in this interval.

Minimal stopping time: The description of each leg contains a minimum stopping time. This is the minimal time (in minutes) which the train has to wait at the destination of the leg before it can depart for the next leg. The corresponding value for the final leg can be ignored.

Safety distances: Each leg possesses a safety distance value (in minutes). This is the minimal time which the subsequent train passing the same track has to wait before it can depart for the corresponding leg. The order of the trains passing a given track has to be retained.

Passenger connections: At each station, we have to ensure that the passenger connections established in the original timetable are retained. If the arrival time of one train and the departure time of another lie in an interval of 5 to 15 minutes, their new arrival and departure times have to lie in this interval, too.

Under these timetable constraints, you are to find a new timetable that minimizes the system-wide power cost. The description of each leg contains (in a separate file) the time-power curve induced by the corresponding train. We assume that this curve as well as the travel time of the leg are fixed. Summation of the power curves for all trains over the complete planning horizon (starting at minute 0 and ending with the latest possible arrival time) leads to the system-wide power curve. Its power cost is determined by averaging its values over 15-minute intervals. The first such interval starts at minute 0, the next one at minute 15, etc. The power cost is now given by the value of the highest such 15-minute average multiplied by a constant cost factor. That means your objective is to minimize the highest arising 15-minute average of the system-wide power curve.

Your solutions have to fulfill the following requirements in addition to the above rules:

1. Trains may only depart at full minutes in the planning horizon (which starts at minute zero, second zero). In the timetable data, you find an earliest and a latest departure time for each leg (given in full minutes). The departure time for this leg has to be a full minute from this interval.

2. When computing the 15-minute averages , you have to set negative values of the summed, system-wide power curve to zero! If braking trains provide more energy than accelerating trains can use at the same time, this energy is lost. That means: You calculate the system-wide curve by summing over the individual curves of the trains, depending on their departure times. Then you compute the maximum of this curve and zero. And then you average over 15-minute intervals – the first of them starting at minute zero, second zero, and ending at minute 15, second zero, interval ends including (the next would start at minute 15, second zero and end at minute 30, second zero).

3. The calculation of the 15-minute average power values follows the trapezoidal rule for the approximate calculation of integrals as described in the following. Each 15 minute-interval consists of 901 seconds (as the last second is also part of the next interval). Thus, to compute the average of the system-wide power curve, we sum over the 899 *inner* seconds of the interval (with weight 1) and add the value for the

first and the last second with a weight of $\frac{1}{2}$. This value is then divided by 900 – the length of the interval. The resulting value is the 15-minute average which is to be optimized (Explanation: without the division by 900, this value represent the total energy consumed by the trains over that interval – under the assumption that the system-wide power curve is a piecewise-linear function. Division by the length of the time interval yields the average power drawn from the system).

4. To enable you to check the feasibility of your solutions and the correctness of your objective values, we provide you with a solution checker (download link below). It is written for Python 2.7 and is executed as follows: `python solution_checker.py instance-data-file power-data-file solution-file`

   Together with the solution checker, we also provide two sample solutions for instances 1 and 10 (which correspond to the initial timetable, i.e. to the `CurrentDepartureTimes`). Please submit all your solutions in the json-format as in these two examples. Basically, you have to give one value (in minutes) for the departure time of each leg. Please pack your 10 solutions to one zip file before uploading.

2. Mixed Integer Program Formulation

We describe the instances of the challenge, an exact mixed integer programming formulation for the optimization problem, the process of transforming the given data into a suitable format for a MIP solver, and the results (either optimal or failures) obtained.

2.1. The Instances

The challenge contains ten instances with the following statistics:

Instance	Trains	Legs	Tracks	Stations
1	13	206	58	30
2	26	241	78	43
3	37	352	139	76
4	48	676	150	84
5	71	863	274	131
6	73	712	221	109
7	126	1524	308	147
8	182	3709	100	51
9	237	1808	605	265
10	277	2641	666	306

Each instance consists of two JSON files with the same number:

1. `instance_data_number.json.txt` contains the timetable information and
2. `power_data_number.json.txt` contains the power profiles for each leg.

The information on file `instance_data_number.json.txt` is organized with the following fields:

1. `int TrainID` uniquely identifies each train.
2. `int TrackID` uniquely identifies each track.
3. `int LegID` uniquely identifies each train run on a given track.
4. `int StartStationID` uniquely identifies the origin of the leg.

5. `int EndStationID` uniquely identifies the destination of the leg.
6. `int EarliestDepartureTime` is the earliest departure time of the leg.
7. `int LatestDepartureTime` is the latest departure time of the leg.
8. `int CurrentDepartureTime` is the current departure time.
9. `int TravelTime` is the travel time to the next station in minutes.
10. `int MinimumStoppingTime` is the minimum time the trains needs to stay in the following station (if one exists) in minutes.
11. `int MinimumHeadwayTime` is the minimum time the next train on the same route has to wait until it can depart.

This is a minimal example taken from `instance_data_1.json.txt`:


```
{
  "Trains": [
    {
      "TrainID": 31698,
      "Legs": [
        {
          "StartStationID": 2716,
          "LatestDepartureTime": 4,
          "CurrentDepartureTime": 1,
          "EndStationID": 6499,
          "MinimumHeadwayTime": 2,
          "TrackID": 415361,
          "EarliestDepartureTime": 0,
          "MinimumStoppingTime": 1,
          "TravelTime": 3,
          "LegID": 426408
        },
      ]
    }
  ]
}
```


These data are subject to the following constraints:

1. The trains can only depart every full minute from the `EarliestDepartureTime` until `LatestDepartureTime`.
2. Two consecutive trains using the same track have to satisfy a safety constraint. The departure of the later train must be greater or equal to the `DepartureTime` of the earlier train plus the minimum headway time specified for the earlier train.

3. This constraint implicitly enforces the same order of the trains on each track as established in the original timetable. That means the order of the trains is fixed and is not part of the optimization.

4. At each station, the new timetable has to respect the passenger connections established in the original timetable. If the arrival of one train at a given station and the departure of another train take place within an interval of 5 to 15 minutes in the old timetable, this relation has to be preserved in the new timetable.

The information on file `power_data_number.json.txt` is organized with the following fields:

1. `int LegID` correspond to the same `LegID` as in the `instance_data_number.json.txt` and is unique.

2. `float Powerprofile[]` contains exactly `TravelTime*60+1` time steps (seconds), where the power is measured (in MW).

This is a minimal example taken from `power_data_1.json.txt`:

```
{
  "Powerprofiles": [
    {
      "LegID": 426408,
      "Powerprofile": [
        0.0,
        0.029,
        ...
        -0.027,
        0.0
      ]
    }
  ]
}
```

2.2. Constants and Decision Variables

The general parameters for a given instance $\mathcal{I}$ are:

1. Let T be the number of *trains* and $\mathcal{T}$ be the set of trains.

2. Let L be the number of *legs* and $\mathcal{L}$ be the set of legs.

3. Let K be the number of *tracks* and $\mathcal{K}$ be the set of tracks.

4. Let S be the number of *stations* and $\mathcal{S}$ be the set of stations.

For a given leg $\ell \in \mathcal{L}$, we define some constants:

1. Let $s_\ell \in \mathcal{S}$ be its *source* (origin) station.
2. Let $t_\ell \in \mathcal{S}$ be its *target* (end, destination) station.
3. Let $e_\ell \geq 0$ be its *earliest departure* time.
4. Let $l_\ell \geq 0$ be its *latest departure* time.
5. Let $c_\ell \geq 0$ be its *current departure* time.
6. Let $r_\ell \geq 0$ be its *running* (travel) time.
7. Let $w_\ell \geq 0$ be its *minimum waiting* (stopping) time at the target destination.
8. Let $h_\ell \geq 0$ be the *minimum headway* time for the next train on the same track.
9. For $0 \leq i \leq 60r_\ell$, let $p_{\ell,i}$ be the power consumption measured at second i from departure.

It is given that $e_\ell \leq c_\ell \leq l_\ell$ for all $\ell \in \mathcal{L}$. Finally, we define the decision variable $x_\ell \in \mathcal{Z}_+$ as the *actual departure* time for leg $\ell \in \mathcal{L}$.

2.3. Constraints

2.3.1. Departure Constraints

The following *departure constraints* must hold:

$$e_\ell \leq x_\ell \leq l_\ell \text{ for all } \ell \in \mathcal{L}. \tag{2.1}$$

2.3.2. Safety Constraints

For each track $k \in \mathcal{K}$, let $\ell_{k,1}, \ell_{k,2}, \ldots, \ell_{k,p}$ be the p legs using track k sorted by increasing departure time, that is, $c_{\ell_{k,1}} < c_{\ell_{k,2}} < \cdots < c_{\ell_{k,p}}$. Then the following *safety constraints* must hold:

$$x_{\ell_{k,i}} - x_{\ell_{k,i-1}} \geq h_{\ell_{k,i-1}} \text{ for all } 1 < i \leq p. \tag{2.2}$$

2.3.3. Waiting Constraints

For each train $t \in \mathcal{T}$, let $\ell_{t,1}, \ell_{t,2}, \ldots, \ell_{t,q}$ be the q legs using train t sorted by increasing departure time, that is, $c_{\ell_{t,1}} < c_{\ell_{t,2}} < \cdots < c_{\ell_{t,q}}$. Then the following *waiting constraints* must hold:

$$x_{\ell_{t,i}} - x_{\ell_{t,i-1}} \geq r_{\ell_{t,i-1}} + w_{\ell_{t,i-1}} \text{ for all } 1 < i \leq q. \tag{2.3}$$

2.3.4. Connection Constraints

For each station $s \in \mathcal{S}$, let $\ell^-_{s,1}, \ell^-_{s,2}, \ldots, \ell^-_{s,p}$ be the p legs arriving to station s and let $\ell^+_{s,1}, \ell^+_{s,2}, \ldots, \ell^+_{s,q}$ be the q legs departing from station s. Then the following *connection constraints* must hold:

$$5 \leq x_{\ell^+_{s,j}} - (x_{\ell^-_{s,i}} + r_{\ell^-_{s,i}}) \leq 15 \text{ for all } 1 \leq i \leq p \text{ and } 1 \leq j \leq q, \tag{2.4}$$

provided that legs $\ell^-_{s,i}$ and $\ell^+_{s,j}$ correspond with different trains and their departure times satisfy $5 \leq c_{\ell^+_{s,j}} - (c_{\ell^-_{s,i}} + r_{\ell^-_{s,i}}) \leq 15$.

2.4. Objective Function

All given instances have a planning horizon of 17 fifteen-minutes intervals. For a given feasible solution x, let $f(x,i)$ be its average power consumption on interval $0 \leq i \leq 16$. The value of solution x is then $f(x) = \max\{f(x,i) : 0 \leq i \leq 16\}$. Finally, the objective value is $z^* = \min\{f(x) : x \text{ is feasible}\}$. It is not completely obvious that z^* is a linear function of x.

Assume first that we could write $f(x,i)$ as a linear function of x, then it would be easy to obtain z^* in the following way:

$$\begin{aligned} z^* \quad = \quad & \min z & (2.5)\\ & \text{subject to} \\ & f(x,i) \leq z \text{ for all } 0 \leq i \leq 16. & (2.6) \end{aligned}$$

In order to write each $f(x,i)$ as a linear function of x, we proceed in three steps. First, for each $0 \leq s \leq 17 \times 15 \times 60$, let $\pi_s \geq 0$ be the total power consumption on second s (multiplied by $\frac{1}{2}$ if s is a multiple of 900). Therefore:

$$f(x,i) = \frac{1}{900} \sum_{s=900i}^{900(i+1)} \pi_s. \tag{2.7}$$

Second, for each leg $\ell \in \mathcal{L}$, and for each $e_\ell \leq m \leq l_\ell$, let $y_{\ell,m} \in \{0,1\}$ be a binary variable indicating whether leg ℓ departed on minute m. This can be achieved as follows:

$$\sum_{m=e_\ell}^{l_\ell} y_{\ell,m} \quad = \quad 1 \tag{2.8}$$

$$\sum_{m=e_\ell}^{l_\ell} m y_{\ell,m} \quad = \quad x_\ell. \tag{2.9}$$

Third, we need to relate the variables π_s with the variables $y_{\ell,m}$. In particular, $y_{\ell,m} = 1$ contributes to the value of π_s if $60m \leq s \leq 60(m + r_\ell)$. If s is not a multiple of 900 we have:

$$\pi_s \geq \sum_{\ell \in \mathcal{L}} \left(\sum_{m=\lceil s/60 - r_\ell \rceil}^{\lfloor s/60 \rfloor} p_{\ell,s-60m} y_{\ell,m} \right). \tag{2.10}$$

Otherwise, if s is a multiple of 900 we have:

$$\pi_s \geq \frac{1}{2} \sum_{\ell \in \mathcal{L}} \left(\sum_{m=\lceil s/60 - r_\ell \rceil}^{\lfloor s/60 \rfloor} p_{\ell,s-60m} y_{\ell,m} \right). \tag{2.11}$$

Note that in the inner sum, if $m < e_\ell$ or $m > l_\ell$ we can assume $y_{\ell,m} = 0$. Also note that these inequalities allow their right-hand sides to be negative, but $\pi_s \geq 0$.

2.5. Mixed Integer Model Generator

We will briefly explain the PHP program used to generate the LP files read by Gurobi in order to find the solutions. The explanation does not contain code since it can be found in the appendices. On the other hand, the following text explains the program in the order in which the code appears and indicates the lines of code inside parentheses.

Programs written in PHP can be executed directly from the console, in a similar way to Python or Java. For example, to generate the model for instance 1, we will execute the program like this:

```
php model.php 1
```

The program automatically reads the instance and power data files. This is convenient most of the time, but expects the files to be present in the current directory with the expected names. If the data files could not be read, the execution terminates (2 – 12).

PHP can read objects described in JSON into PHP's `stdClass` objects or optionally translate them to a PHP array representation. We prefer to use the latter if possible. Unfortunately, the instances are not described in a format we found useful, as most of the time we want fast access to the information of a leg (using its ID as a key). Because of this, we reconstruct the data internally and also associate the power wave for each leg (16 – 29).

We proceed to generate the LP model. We will store its contents in memory and then we will write it to disk in a single operation; this may improve the performance of the model generator. In order to do so, we start the output buffering using the `ob_start` function: the buffer will capture any printed or `echo`'ed string instead of sending it to the standard output (31 – 32).

In the generated model, the z^* variable of the objective function (described in chapter 2) is denoted as `promedio_intervalo` (35 – 36).

Some auxiliary functions that will be used in several places through the script are:

- The `mapea` function. It splits a set of legs based on their value for a given attribute (for example, `EndStationID`); such value is then used as an index (42 – 51).
- The `compara_3vias`. It implements a simple three-way comparison (returning a negative value, zero or a positive value depending on `a < b`, `a = b` or `a > b` respectively (53 – 56).
- The `compara_llegadas`. It numerically compares arrival times between two different legs irrespective of their destinations (58 – 61).
- The `compara_salidas`. It numerically compares departure times between two different legs irrespective of their starting stations (63 – 66).

We denote the variables x_ℓ (described in chapter 2) as `salida_l`. To generate constraints that ensure that safety distances are not violated, we map every leg by `TrackID` and then sort each group by departure time. Consecutive trains that use the same track must depart with at least `MinimumHeadwayTime` minutes of difference (70 – 81).

To generate constraints that ensure that minimum stopping times at the destinations are respected, we do something similar. We map every leg by `TrainID` and then sort each group by departure time. The next departure of a train must happen at least `TravelTime + MinimumStoppingTime` minutes after the previous departure (83 – 97).

To generate constraints that ensure that connection relationships are respected, we map a copy of the legs by `EndStationID` and another copy by `StartStationID`. Then we sort the first one by arrivals and the second one by departures. After that, we check for each station and for each arrival, which departures are within the interval of what is considered a connection. PHP does not provide a binary search function and we didn't consider important to implement one, but we do prematurely end the search when we notice we can do so (101 – 140).

Generating the linearization constraints of the objective function is a bit more complicated. Before generating any constraint and in an attempt to decrease the file size of the generated model, we calculate for each interval the set of departure times per leg that may affect its power wave. We also define an auxiliary function `accede` that returns the power level of a power wave in a given second; it returns zero if the offset is out of bounds (144 – 163).

We denote the variables $y_{\ell,m}$ (described in chapter 2) as `salida_l_m`. With this information, we calculate the sum of all power contributions per second per interval. All (non-zero) power contributions are added to the corresponding seconds' power level when a binary variable `salida_l_m` has a value of 1. To implement the trapezoidal rule, we simply divide by 2 the power contributions at seconds that are multiples of 900 (that

is, the first and last second of each interval). The sum of power contributions at second `i` is represented as the model variable `segundo_i`. The total power of an interval is the sum of the variables `segundo_i` included in such interval. We then specify that the objective function variable `promedio_intervalo` must be greater than or equal to the sum of each interval. We do not divide this value by 900 as this can be done after solving the model (167 – 200).

The constraints that force the binary variables `salida_l_m` to 1 if `salida_l` = `m` (and to zero otherwise) as described in chapter 2 can be generated without difficulties. We finish the generation of the LP file by writing the types and bounds for each variable. The bounds for the `salida_l` variables are simply those of the earliest and latest departure times. It is important not to forget to set the bounds for the `segundo_i` variables, since the power level of a second could be negative otherwise (204 – 234).

To retrieve everything stored in the output buffer and write it to disk at once, we use the PHP functions `ob_get_clean` and `file_put_contents`. We also generate the MST file, which will contain an initial feasible solution that Gurobi can read. This file is generated using the current departure times (260 – 267).

2.6. Starting the Solver

The Gurobi Optimizer can be started in multiple ways. For example, the program `gurobi_cl` offers a command line interface to Gurobi. In fact, all Gurobi parameters can be set using the command line interface.

In our experiments, we used non-default values for the following parameters (sorted by importance and frequency of use):

- The `InputFile` parameter. A path to an MST file with an initial feasible solution.
- The `ResultFile` parameter. The path where to store the optimal solution if found.
- The `MIPFocus` parameter. An integer between 0 (the default) and 3, where 1 tells Gurobi to focus on finding good solutions, 2 to focus on proving the current solution is optimal, and 3 to focus on raising the lower bound.
- The `MIPGap`. If Gurobi proves that the current solution is less than `MIPGap` percent away from the optimal, the optimization terminates.
- The `Threads`. It limits the number of concurrent threads Gurobi uses during the MIP optimization. Reducing the number of threads probably reduces the performance but also lowers the memory requirements, as every thread needs a copy of the model.

- The `ImproveStartGap`. Gurobi will dinamically give up on proving optimality and will change its focus on improving solutions after a certain MIP Gap is reached.
- The `Heuristics`. An integer between 0 and 1 that indicates the percentage spent on heuristic algorithms.
- The `Cuts`. An integer between -1 (the default) and 3 that indicates the level of aggressiveness of the cut generator.

We also tried to use the `NodefileStart` parameter to allow Gurobi to dump inactive data to disk in order to free some RAM, but it triggered runtime errors. Most of the time we used `MIPFocus=2`.

The path to the model file is the last (nameless) parameter specified in the command line. For example, we can solve instance 1 with the following command:

```
gurobi_cl InputFile=model1.mst ResultFile=result1.mst MIPFocus=2
    MIPGap=0 model1.lp
```

Using the command line interface, only the last solution found will be written to disk and this is done after the optimization ends. If the model is taking too long to reach optimality, terminating the process is necessary in order to recover the best current solution.

2.7. Recovering Intermediate Solutions

We will briefly explain the C++ program used to write to disk the sequence of solutions found by Gurobi. The explanation does not contain code since it can be found in the appendices. On the other hand, the following text explains the program in the order in which the code appears and indicates the lines of code inside parentheses.

Programs written in C++ can be executed directly from the console and they can use Gurobi as a library. We have compiled it under Windows and Linux. Under Windows, we used Visual Studio 2013 with the following commands:

```
"C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\vcvarsall.
    bat" x64
cl gurobi_cpp.cpp /EHsc gurobi60.lib gurobi_c++mt2013.lib
```

The LIB files may be found in the Gurobi installation folder; we simply copied them to the current working directory. Under Linux, the following command may be used (assuming the installation is at the specified folders):

```
g++ gurobi_cpp.cpp -std=c++11 -I"/opt/gurobi604/linux64/include"
    "/opt/gurobi604/linux64/lib/libgurobi_c++.a" "/opt/gurobi604/
    linux64/lib/libgurobi.so.6.0.4" -O3 -o gurobi_cpp
```

Gurobi has many parameters but our program does not support them all. This is important to note, since every non-default parameter must be set using the Gurobi API. In most of our experiments, we only specified the `InputFile`, `ResultFile`, `MIPFocus` and `MIPGap` parameters. Thus, our C++ program needs (and actually requires) the first three parameters plus the LP model; the MIP Gap will always be set to 0, which is a non-default value. For example, we can solve instance 1 with initial solution at `initial1.mst`, optimal solution to be stored in `optimal1.mst` and a MIP focus of 2 with the following command:

```
./gurobi_cpp initial1.mst optimal1.mst 2 model1.lp
```

The C++ program sets the parameters via the Gurobi API and reads the model. The Gurobi API throws an exception when an error is found (for example, unable to read file), but we do not catch it. This may cause the program to abort (36 – 48).

After reading the input, we install a callback that Gurobi will invoke each time an event is recorded during the MIP optimization. The event `GRB_CB_MIPSOL` is signaled when a new feasible solution is found, so we load the current values of the model variables and write them in an MST file named with the model name, the string `"_temp"` and the objective value of the current solution (48 – 49, 6 – 32).

If the model is feasible and after the optimization ends, we write the optimal solution to disk (50 – 54).

2.8. Transforming MST into JSON

We will briefly explain the PHP program used to generate the JSON files from the MST files that Gurobi writes as solutions. The explanation does not contain code since it can be found in the appendices. On the other hand, the following text explains the program in the order in which the code appears and indicates the lines of code inside parentheses.

The program needs the instance number and the MST file. For example, to the process `solution1.mst` for instance 1, we will execute the program like this:

```
php mst_handler.php 1 solution1.mst
```

The program does not validate feasibility, but does evaluate the objective function. Because of this, it automatically reads the corresponding power data file. This is convenient most of the time, but expects the file to be present in the current directory with the expected name. If the data files could not be read, the execution terminates (2 – 12).

PHP can create `stdClass` objects and write them into JSON. From the MST file we extract the information of the variables x_ℓ denoted as `salida_l`, constructing an `stdClass` object in the process. Since Gurobi is a branch-and-cut solver that uses floating point arithmetic, it may happen that some values are non-integers: Gurobi's default integer

feasibility tolerance is 1E-05. We have noticed this does not happen for the `salida_l` variables but sometimes does for the `salida_l_m` variables, which we ignore anyway. Nevertheless, we round the wanted values and cast them to integers. (17 – 33).

Some MST files may contain information about unwanted variables (for example, `segundo_i`) so we also purge the original MST file. We then write the resulting JSON code to disk in a single operation (36 – 37).

To compute the objective function, we simply calculate the sum of power contributions for each second and then the sum of power contributions for each interval, using the trapezoidal rule where needed. Negative power levels are zeroed when added to the power consumption of an interval (40 – 68).

2.9. Results

The models for the ten instances have the following number of constraints:

Instance	Departure	Safety	Waiting	Connection
1	206	148	193	21
2	241	163	215	31
3	352	213	615	68
4	676	526	628	274
5	863	589	792	210
6	712	491	639	184
7	1524	1216	1398	1035
8	3709	3609	3527	17261
9	1808	1203	1571	985
10	2641	1975	2364	1773

After running the solver, five instances were solved to optimality, while the other five instances ran out of memory:

Instance	Current value	Optimal value	Improvement
2	4.620948333	2.939638333	36.38%
3	20.635706111	15.179507777	26.44%
4	19.983620000	14.966736111	25.10%
5	27.551612222	19.416015000	29.53%
6	26.349433888	20.431125556	22.46%

3. Ideas for Finding Good Solutions

We sketch two main ideas of how to obtain good solutions and good lower bounds in those cases where the MIP solver failed. They are simplifications of the objetive function and of the data. We also sketch a dynamic programming approach to solve the problem.

3.1. Simplification of Objective Function

Our first simplification is straightforward: instead of computing the objetive function with the trapezoidal rule, simply sum the power consumption over each 900 seconds interval. In other words, do not use Equation 2.7 while looking for solutions.

3.2. Simplification of Power Consumption

Our second simplification is also simple: instead of using the data for each second, group the data with certain granularity and average it. Hence, instead of having a variable for each second's power consumption, we would only have a variable for each subinterval. In all figures, a red dot means power consumption, a blue dot means power generation.

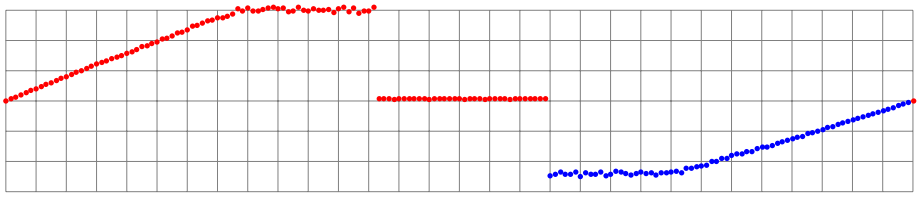

Figure 3.1.: A typical power consumption wave.

An interesting property of granularity and averaging is as follows: Consider a fixed fifteen-minutes interval i and a fixed schedule x^i for the legs running during that interval. Partition interval i into a set A of subintervals and further refine this partition into a set B of subintervals. Let $f_A(x^i, i)$ be the power consumption evaluated using power averages over each subinterval in A and let $f_B(x^i, i)$ be the power consumption evaluated using power averages over each subinterval in B. Then $f_A(x^i, i) \leq f_B(x^i, i)$. Furthermore, if x_A^* is a minimizer of f_A and x_B^* is a minimizer of f_B, then $f_A(x_A^*, i) \leq f_A(x_B^*, i) \leq f_B(x_B^*, i)$. In particular, for an arbitrary partition A we have the lower bound $f_A(x_A^*, i) \leq f(x^*, i)$.

3.2.1. Constant Granularity

Since each interval consists of 900 seconds, it is natural to group the power consumption data in $\frac{900}{d}$ intervals of d seconds. We discuss this idea further in the next chapter. Note that if $d_1|d_2$ and $d_2|900$, then $f_{d_2}(x^i, i) \leq f_{d_1}(x^i, i)$. In particular, given a $d|900$ we have $f_d(x_d^*, i) \leq f_1(x^*, i) = f(x^*, i)$. Hence, we have a lower bound to the original problem.

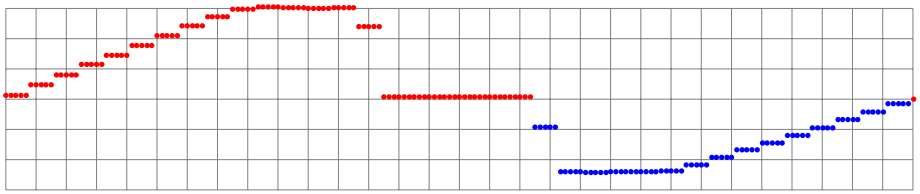

Figure 3.2.: A power consumption wave averaged over 5 seconds subintervals.

3.2.2. Variable Granularity

On further inspection of the power consumption data, we noted that each leg goes through five stages (or less) of power consumption:

1. Almost linearly increasing power consumption.
2. Almost constant maximum power consumption.
3. Almost constant cruise speed power consumption.
4. Almost constant maximum power generation.
5. Almost linearly decreasing power generation.

Therefore, it would also be natural to split the power consumption data over this five stages and average in each of them. We noted that sometimes only the first, third, and fifth stages occur. We also noted that sometimes the third stage generates power.

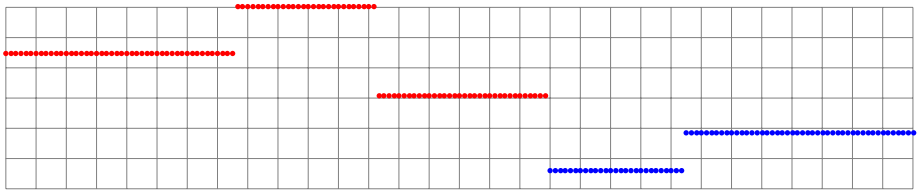

Figure 3.3.: A power consumption wave averaged over each stage.

3.2.3. Piecewise Linear Functions

Given the behavior described above, one could also model the power consumption as a piecewise linear function with five (or three) pieces. Each of these linear functions could be computed using, for instance, linear regression.

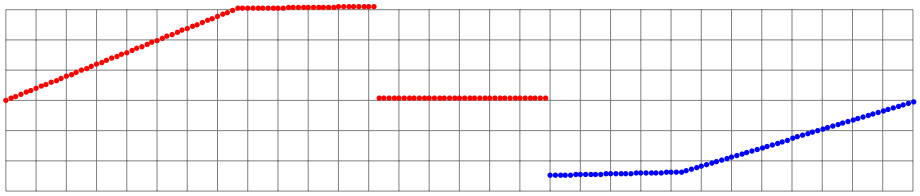

Figure 3.4.: A power consumption wave approximated by a piecewise linear function.

3.3. Dynamic Programming

For each $0 \leq i \leq 16$, let $\mathcal{L}_i$ be the set of legs that may be running during the i-th fifteen-minutes interval in the planning horizon and let $\mathcal{X}_i$ be the set of *feasible* assignments to the variables corresponding to legs in $\mathcal{L}_i$. Note that a leg may belong to one or more consecutive intervals. Also note that the power consumption during the i-th fifteen-minutes interval only depends on the assignment $x^i \in \mathcal{X}_i$. We say that $x^{i-1} \in \mathcal{X}_{i-1}$ and $x^i \in \mathcal{X}_i$ are *compatible* if they coincide in $\mathcal{L}_{i-1} \cap \mathcal{L}_i$, and satisfy all required constraints.

For $x^i \in \mathcal{X}_i$, let $g(x^i, i)$ be the minimum of the maximum power consumption over the fifteen-minutes intervals $0, \ldots, i$ given that interval i is scheduled with x^i. Then

$$g(x^0, 0) = f(x^0, 0) \text{ for all } x^0 \in \mathcal{X}_0 \tag{3.1}$$

and, for all $1 \leq i \leq 16$ and all $x^i \in \mathcal{X}_i$,

$$g(x^i, i) = \min_{x^{i-1} \in \mathcal{X}_{i-1}} (\max(g(x^{i-1}, i-1), f(x^i, i)) : x^{i-1} \text{ and } x^i \text{ are compatible}). \tag{3.2}$$

Finally, let $g(i) = \min(g(x^i, i) : x^i \in \mathcal{X}_i)$. It follows that $g(0) \leq g(1) \leq \cdots \leq g(16) = z^*$.

3.3.1. Dynamic Programming with Boundaries

It turns out that the sets $\mathcal{X}_i$ are too large and, therefore, the computation of all $g(x^i, i)$ would be too slow. We obtain an improvement if, instead of considering all legs in $\mathcal{L}_i$, we only consider the assignments to *boundary* legs, that is, those legs in $\mathcal{L}_i$ that constrain the assignments of legs in either $\mathcal{L}_{i-1}$ or $\mathcal{L}_{i+1}$. In this way, there would be fewer $g(x^i, i)$ computed, but each of these computations would be more expensive. In either case, the computation could be accelerated by implementing it in parallel for each stage i.

4. Modified Mixed Integer Program

We describe an approximate mixed integer programming formulation for the optimization problem, based on the idea of constant granularity, and the results obtained.

4.1. Constant Granularity Mixed Integer Program

The mixed integer program that we propose is a modification of our previous exact model. All variables x_ℓ and $y_{\ell,m}$ retain their meaning, but variables π_s change their meaning into the total power consumption over a subinterval of a few seconds. Therefore, we only explain how do we compute a modified objective function.

Let d be a divisor of 900. For a given feasible solution x, let $f_d(x,i)$ be its average power consumption on interval $0 \leq i \leq 16$, computed using the *average* power consumption in subintervals of d seconds. The value of solution x is then $f_d(x) = \max\{f_d(x,i) : 0 \leq i \leq 16\}$. Finally, the objective value is $z_d^* = \min\{f_d(x) : x \text{ is feasible}\}$.

If we write $f_d(x,i)$ as a linear function of x, then it is easy to obtain z_d^* as follows:

$$\begin{aligned} z_d^* &= \min z && (4.1)\\ &\quad \text{subject to} \\ &\quad f_d(x,i) \leq z \text{ for all } 0 \leq i \leq 16. && (4.2) \end{aligned}$$

In order to write each $f_d(x,i)$ as a linear function of x, we proceed in three steps. First, for each $0 \leq s \leq \frac{1}{d}17 \times 15 \times 60$, let $\pi_s \geq 0$ be the total power consumption on the time interval $[ds, d(s+1)]$ in seconds. Therefore:

$$f_d(x,i) = \frac{1}{900} \sum_{s=900i/d}^{900(i+1)/d-1} \pi_s. \tag{4.3}$$

Second, for each leg $\ell \in \mathcal{L}$, and for each $e_\ell \leq m \leq l_\ell$, let $y_{\ell,m} \in \{0,1\}$ be a binary variable indicating whether leg ℓ departed on minute m. This can be achieved as follows:

$$\sum_{m=e_\ell}^{l_\ell} y_{\ell,m} = 1 \tag{4.4}$$

$$\sum_{m=e_\ell}^{l_\ell} m y_{\ell,m} = x_\ell. \tag{4.5}$$

Third, we need to relate the variables π_s with the variables $y_{\ell,m}$. In particular, $y_{\ell,m} = 1$ contributes to the value of π_s if $60m \leq ds \leq 60(m + r_\ell)$.

$$\pi_s \geq \sum_{\ell \in \mathcal{L}} \left(\sum_{m=\lceil ds/60 - r_\ell \rceil}^{\lfloor ds/60 \rfloor} \left(\sum_{k=0}^{d-1} p_{\ell,ds-60m+k} \right) y_{\ell,m} \right). \tag{4.6}$$

Note that in the middle sum, if $m < e_\ell$ or $m > l_\ell$ we can assume $y_{\ell,m} = 0$. Also note that these inequalities allow their right-hand sides to be negative, but $\pi_s \geq 0$.

4.2. Modified Mixed Integer Model Generator

There are only minor differences between the exact model generator and the constant granularity model generator. The first difference is that the granularity `g` must be provided as an extra command argument and it must be a divisor of 900 (2 – 17).

The next difference is that the sum of power contributions at second `i` (where `i` is a multiple of `g`) is itself added to those for seconds `i + 1`, `i + 2`, ... `i + g - 1`. To implement something compatible with the trapezoidal rule, the second 0 relative to each interval is half-added while the second 900 is ignored. The variables `segundo_i` no longer exist in the model; the variables that contain the sum of all power contributions in a group of `g` seconds are denoted as `segmento_i` (179 – 206).

Finally, the file name of the generated LP file has the suffix `g` added (269).

4.3. Solving the Modified Model

An optimal solution found for granularity greater than 1 is not necessarily optimal for granularity equal to 1. In fact, an optimal solution for a coarse granularity is often worse than good but not optimal solution for a fine granularity, as the former is too specialized for the coarse model. However, when a mediocre solution did not improve quickly with the exact model, it often happened that the first improved solutions for coarse granularities were much better than it in the exact model. Furthermore, those better solutions were found in a matter of seconds.

Using a good starting solution affects Gurobi in strange and noticeable ways. The search tree prune and also the Gurobi heuristics become more effective, decreasing the overall runtime and more importantly, the memory consumption. Many were the times when Gurobi ran out of memory while solving the exact model, and a better upper bound found using granularities allowed the exact model to delay such situation in later executions, increasing the chance of finding even better solutions.

4.4. Results

Using constant granularity, we were able to obtain good solutions for all instances that were not solved before. The following table summarizes these results. The results in **bold** were used to continue our work.

Inst.	Current value	g	$f(x_g)$	Impr.	$f_g(x_g)$	Gap
1	1.778052222	20	**1.09065**	38.66%	1.04635	4.06%
7	21.700938333	60	**16.33821**	24.71%	16.32999	0.05%
7	21.700938333	50	16.67676	23.15%	16.29276	2.30%
7	21.700938333	45	16.54990	23.74%	16.29961	1.51%
7	21.700938333	9	16.38669	24.49%	16.30985	0.47%
8	2.772963333	60	2.56884	7.36%	2.23526	12.99%
8	2.772963333	50	2.55792	7.76%	2.19013	14.38%
8	2.772963333	45	2.51708	9.23%	2.19154	12.93%
8	2.772963333	9	2.44258	11.91%	2.28935	6.27%
8	2.772963333	2	**2.42464**	12.56%	2.42195	0.11%
9	113.391771111	60	98.40077	13.22%	98.10779	0.30%
9	113.391771111	45	98.17457	13.42%	98.09943	0.08%
9	113.391771111	9	98.16771	13.43%	98.09419	0.07%
9	113.391771111	6	98.16771	13.43%	98.09419	0.07%
9	113.391771111	2	**98.13391**	13.46%	98.09419	0.04%
10	79.585383888	60	68.16482	14.35%	68.10658	0.09%
10	79.585383888	45	68.14369	14.38%	68.08756	0.08%
10	79.585383888	9	**68.14079**	14.38%	68.10320	0.06%

5. Our Results

For each instance, we collect the original objective value, our best objective value, the improvement (as a percentage), our best lower bound, and the optimality gap (as a percentage). The last line is the total over all instances.

Inst.	Current value	Our best value	Impr.	Lower bound	Gap
1	1.778052222	1.071813889	39.72%	0.994686978	7.20%
2	4.620948333	2.939638333	36.38%	2.939638333	0.00%
3	20.635706111	15.179507777	26.44%	15.179507777	0.00%
4	19.983620000	14.966736111	25.10%	14.966736111	0.00%
5	27.551612222	19.416015000	29.53%	19.416015000	0.00%
6	26.349433888	20.431125556	22.46%	20.431125556	0.00%
7	21.700938333	16.321493889	24.79%	16.308135556	0.08%
8	2.772963333	2.424517778	12.57%	2.424517778	0.00%
9	113.391771111	98.133795000	13.46%	98.133674222	0.00%
10	79.585383888	68.121432222	14.40%	68.115126000	0.01%
T	318.370429443	259.006075555	18.65%	258.909163089	0.04%

In what follows, we display power consumption in two figures for each instance: the first for the given schedule, the second for our best result.

As before, a red dot means power consumption, a blue dot means power generation. However, to avoid clutter in this chapter, each dot represents an average power consumption over one minute. The vertical scale accommodates the maximum instantaneous power consumption as given in the current schedule[1], while the horizontal scale accommodates the 17 fifteen-minutes intervals. This scale holds for both figures.

We also display in light red the average power consumption (and in light blue the average *unused* power generation) on each fifteen-minutes interval. In this case, the vertical scale accommodates the maximum average power generation on a fifteen-minutes interval as given in the current schedule. This scale also holds for both figures.

[1] A black dot in the second figure means that our solution exceeds this instantaneous maximum.

5.1. Instance 1

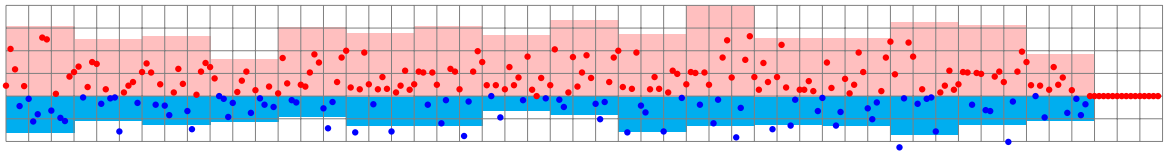

Figure 5.1.: The original input for instance 1.

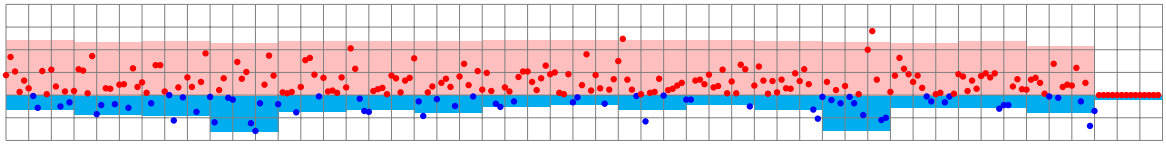

Figure 5.2.: The best output for instance 1.

5.2. Instance 2

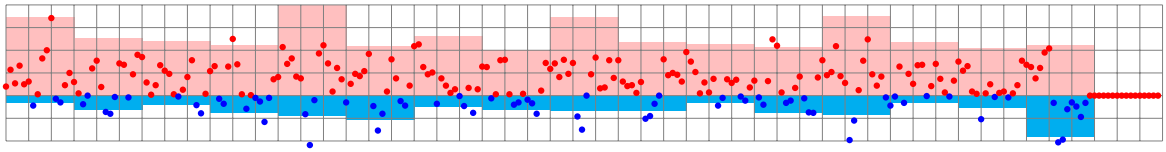

Figure 5.3.: The original input for instance 2.

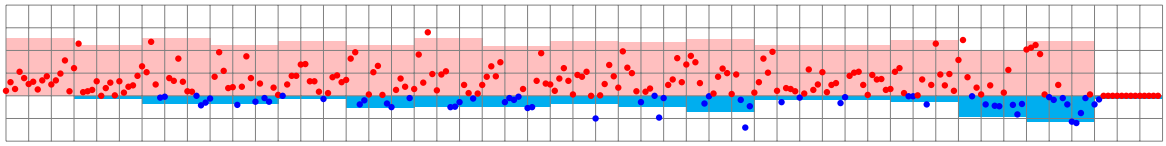

Figure 5.4.: The optimal output for instance 2.

5.3. Instance 3

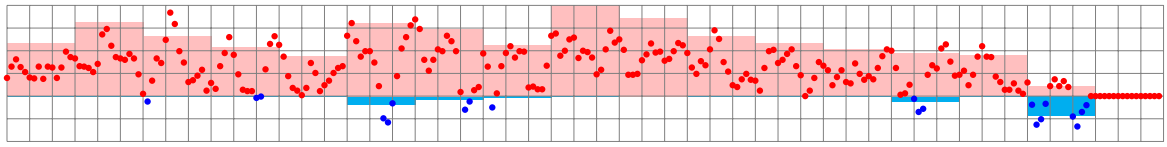

Figure 5.5.: The original input for instance 3.

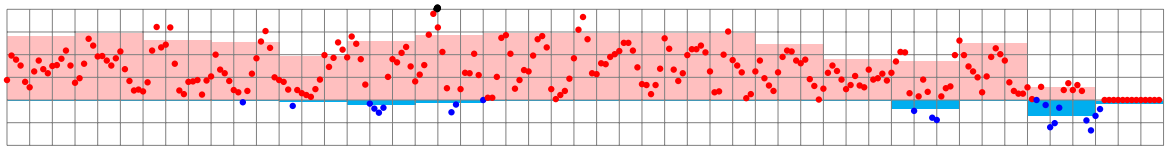

Figure 5.6.: The optimal output for instance 3.

5.4. Instance 4

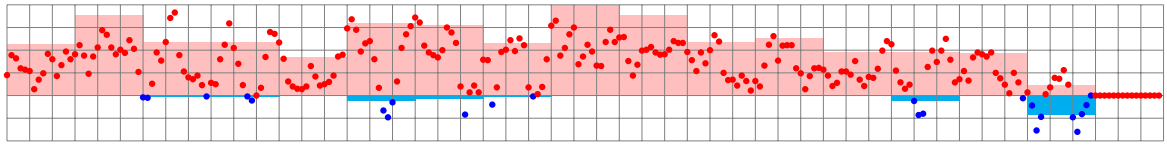

Figure 5.7.: The original input for instance 4.

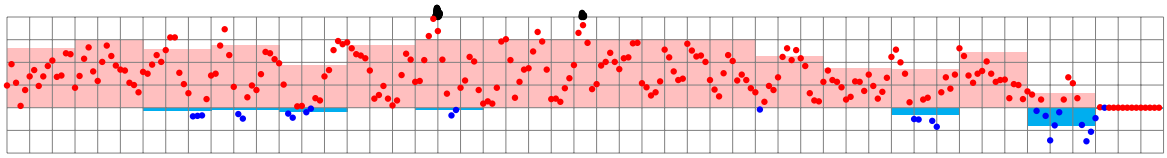

Figure 5.8.: The optimal output for instance 4.

5.5. Instance 5

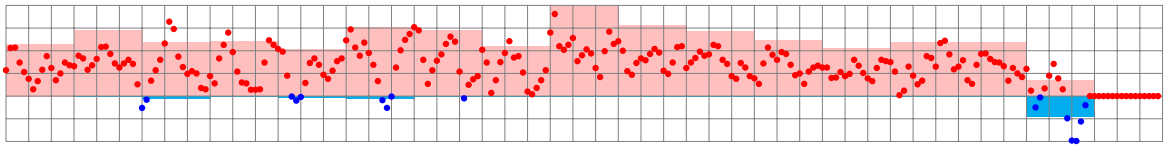

Figure 5.9.: The original input for instance 5.

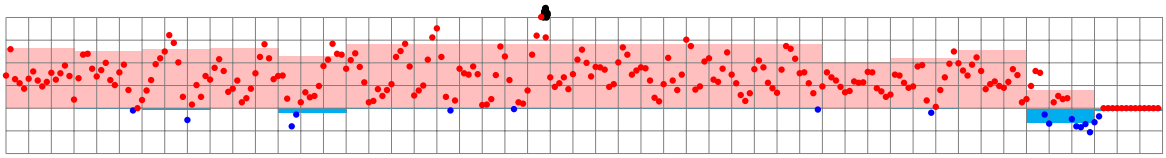

Figure 5.10.: The optimal output for instance 5.

5.6. Instance 6

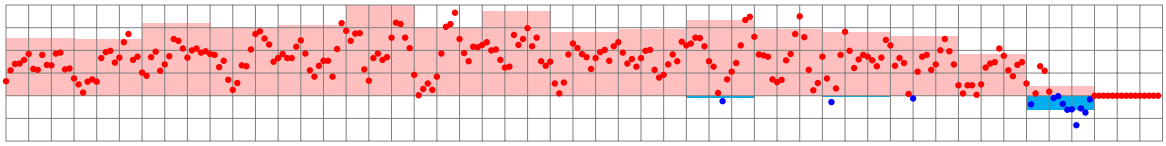

Figure 5.11.: The original input for instance 6.

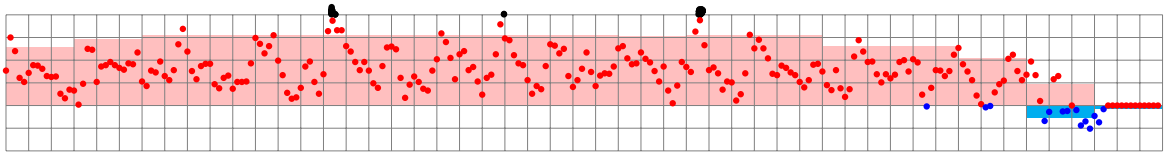

Figure 5.12.: The optimal output for instance 6.

5.7. Instance 7

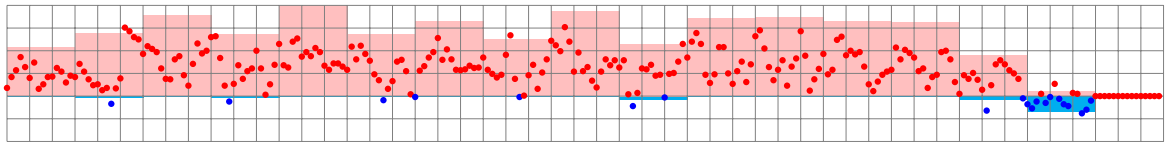

Figure 5.13.: The original input for instance 7.

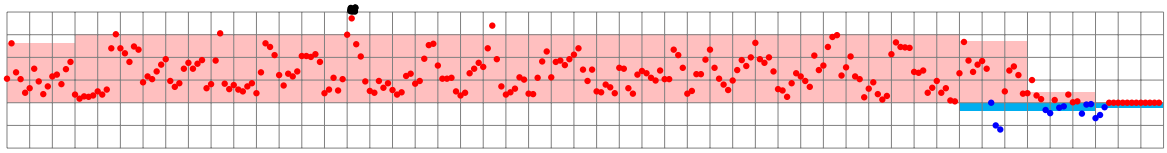

Figure 5.14.: The best output for instance 7.

5.8. Instance 8

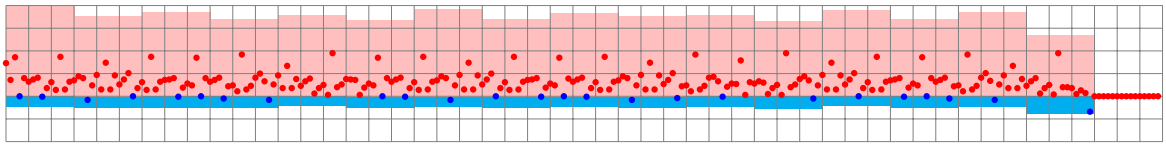

Figure 5.15.: The original input for instance 8.

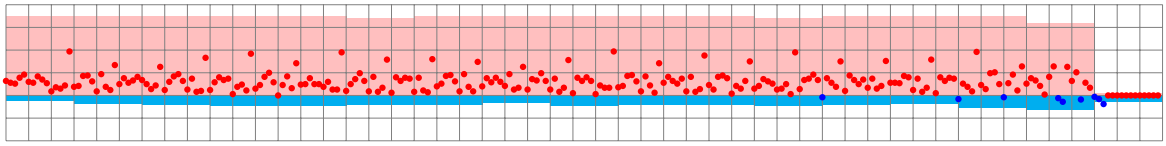

Figure 5.16.: The optimal output for instance 8.

5.9. Instance 9

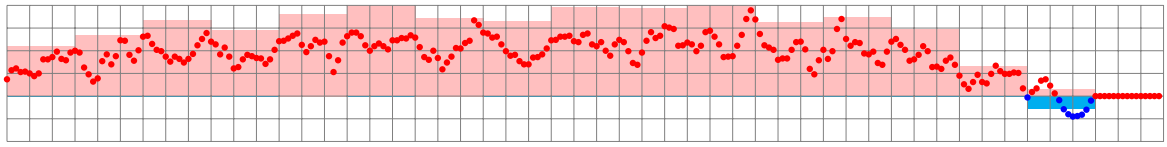

Figure 5.17.: The original input for instance 9.

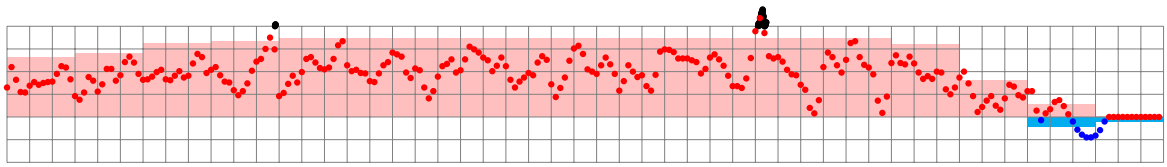

Figure 5.18.: The best output for instance 9.

5.10. Instance 10

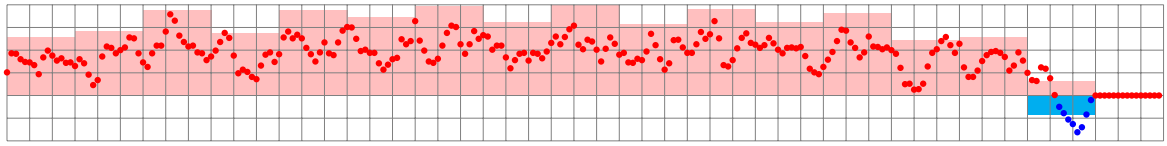

Figure 5.19.: The original input for instance 10.

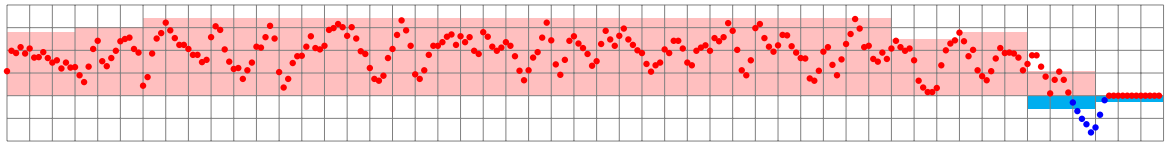

Figure 5.20.: The best output for instance 10.

6. Conclusions

We approached FAU's Optimization Challenge from a mathematical programming perspective, that is, building a mixed integer programming model and solving it via a MIP solver. This proved to be easier said than done: being our computational resources very limited, the MIP solver hit rapidly the maximum memory that we had available.

Therefore, we tried several other algorithmic approaches, but again our limited computational resources disallowed nice techniques such as dynamic programming. It was then that we decided for a heuristic approach, but one that was fit for the purpose.

The idea of simplifying the power consumption waves via averaging with a fixed granularity was very successful. First, within our limited resources, it allowed the MIP solver to find near-optimal solutions to this modified problem that were already big improvements over the given schedule. Second, it came as a bonus that the modified problem gave us simultaneously good upper and lower bounds to the optimal solution. Third, it also allowed us to restart the MIP solver using these solutions in order to obtain near-optimal solutions to the original optimization problem.

An unexpected deficiency of the near-optimal solutions obtained is related to peak power consumption. This happens in our solutions to instances 3–7 and 9, where the new peak power consumption is greater than the initial peak power consumption. In a more realistic setting, there should be a constraint related to peak power consumption.

Another deficiency of our near-optimal solutions is that sometimes our schedule wastes more generated power than the original schedule. This usually happens at the end of the planning horizon: the last fifteen-minutes interval (and often the previous one too) does not reach the maximum average power consumption (in fact, its consumption is almost negligible). We could think of a secondary objective: to minimize the total power consumption (or, equivalently, to minimize the total unused power generation).

Acknowledgments

> Along this particular stretch of
> line no express had ever passed.
> All the trains – the few that there
> were – stopped at all the stations.
>
> *(Aldous Huxley, Crome Yellow)*

We thank the Mexican National Council on Science and Technology (CONACyT) for providing scholarships to most students in our graduate program (including two team members) and also for providing a scholarship to another team member through the National System of Researchers (SNI). We thank the Mexiquense Council on Science and Technology (COMECyT) for providing a scholarship to our last team member.

We thank Universidad Autónoma Metropolitana Azcapotzalco (UAM A) for allowing us the use of some computing facilities through the Systems Department.

We thank Gurobi Optimization for giving us free licenses to their software.

Last, but not least, we thank honorary team member Gabrijela Zaragoza for proposing a nice team name, a portmanteau of *optimization* and the nahuatl word *mixtli* (cloud).

A. Mixed Integer Model Generator

```
<?php
  if (!isset($argv[1])) {
    die("Usage: {$argv[0]} #instance");
  }

  $numero = (int)$argv[1];
  $instancia = json_decode(@file_get_contents("instance_data_{
     $numero}.json.txt"), true);
  $consumo = json_decode(@file_get_contents("power_data_{$numero}.
     json.txt"), true);

  if (!isset($instancia, $consumo)) {
    die("{$argv[0]} {$argv[1]}: unable to read the input files");
  }

  // reconstruction of the input

  $corridas = [ ];

  foreach ($instancia['Trains'] as $trenes) {
    foreach ($trenes['Legs'] as $corrida) {
      $corridas[$corrida['LegID']] = $corrida + [ 'TrainID' =>
         $trenes['TrainID'] ];
    }
  }

  foreach ($consumo['Powerprofiles'] as $onda) {
    $corridas[$onda['LegID']]['Powerprofiles'] = $onda['Powerprofile
       '];
  }

  unset($instancia);
  unset($consumo);

  // LP file
  ob_start( ); {
    // objective function definition
```

```
35      echo "minimize\n";
        echo "\tpromedio_intervalo\n";

        // constraints

40      echo "subject␣to\n";

        function mapea($corridas, $atributo)
        {
          $arr = [ ];
45
          foreach ($corridas as $corrida) {
            $arr[$corrida[$atributo]][] = $corrida;
          }

50        return $arr;
        }

        function compara_3vias($a, $b)
        {
55        return 2 * ($a > $b) - ($a != $b);
        }

        function compara_llegadas($corrida1, $corrida2)
        {
60        return compara_3vias($corrida1['CurrentDepartureTime'] +
             $corrida1['TravelTime'], $corrida2['CurrentDepartureTime']
             + $corrida2['TravelTime']);
        }

        function compara_salidas($corrida1, $corrida2)
        {
65        return compara_3vias($corrida1['CurrentDepartureTime'],
             $corrida2['CurrentDepartureTime']);
        }

        // constraints - headways

70      echo "\\\theadways\n";

        foreach (mapea($corridas, 'TrackID') as $corridas_via) {
          usort($corridas_via, 'compara_salidas');

75        for ($i = 1; $i < count($corridas_via); ++$i) {
```

```
      $corrida = $corridas_via[$i];
      $anterior = $corridas_via[$i - 1];

      echo "\tsalida_{$corrida['LegID']}␣-␣salida_{$anterior['
         LegID']}␣>=␣{$anterior['MinimumHeadwayTime']}\n";
    }
  }

  // constraints - stops at stations

  echo "\\\tstops␣at␣stations\n";

  foreach (mapea($corridas, 'TrainID') as $corridas_tren) {
    usort($corridas_tren, 'compara_salidas');

    for ($i = 1; $i < count($corridas_tren); ++$i) {
      $corrida = $corridas_tren[$i];
      $anterior = $corridas_tren[$i - 1];
      $viaje_con_parada = $anterior['MinimumStoppingTime'] +
         $anterior['TravelTime'];

      echo "\tsalida_{$corrida['LegID']}␣-␣salida_{$anterior['
         LegID']}␣>=␣{$viaje_con_parada}\n";
    }
  }

  // constraints - connections

  echo "\\\tconnections\n";

  $llegadas = mapea($corridas, 'EndStationID');
  $salidas = mapea($corridas, 'StartStationID');
  $estaciones = array_unique(array_merge(array_keys($llegadas),
     array_keys($salidas)));

  foreach ($estaciones as $estacion) {
    if (!isset($llegadas[$estacion])) {
      $llegadas[$estacion] = [ ];
    }

    if (!isset($salidas[$estacion])) {
      $salidas[$estacion] = [ ];
    }

    usort($llegadas[$estacion], 'compara_llegadas');
```

```
    usort($salidas[$estacion], 'compara_salidas');

    foreach ($salidas[$estacion] as $salida) {
      foreach ($llegadas[$estacion] as $llegada) {
        $diferencia = $salida['CurrentDepartureTime'] - ($llegada[
            'CurrentDepartureTime'] + $llegada['TravelTime']);

        if ($diferencia < 5) {
          break;
        }

        if ($diferencia > 15) {
          continue;
        }

        if ($salida['TrainID'] != $llegada['TrainID']) {
          $inferior = $llegada['TravelTime'] + 5;
          $superior = $llegada['TravelTime'] + 15;

          echo "\tsalida_{$salida['LegID']}␣-␣salida_{$llegada['
              LegID']}␣>=␣{$inferior}\n";
          echo "\tsalida_{$salida['LegID']}␣-␣salida_{$llegada['
              LegID']}␣<=␣{$superior}\n";
        }
      }
    }
  }

  // linearization constraints - preprocessing

  $intervalos = [ ];

  function accede($onda, $offset)
  {
    return ($offset < 0 || $offset >= count($onda) ? 0 : $onda[
        $offset]);
  }

  foreach ($corridas as $corrida) {
    $pri = $corrida['EarliestDepartureTime'];
    $ult = $corrida['LatestDepartureTime'];

    for ($i = $pri; $i <= $ult; ++$i) {
      $bloque_ini = (int)($i / 15);
      $bloque_ult = (int)(($i + $corrida['TravelTime']) / 15);
```

```
      for ($j = $bloque_ini; $j <= $bloque_ult; ++$j) {
        $intervalos[$j][$corrida['LegID']][] = $i;
      }
    }
  }

  // linearization constraints - power consumption

  echo "\\\tlinearization constraints - power consumption\n";

  foreach ($intervalos as $bloque => $datos) {
    $segundo_ini = 15 * 60 * $bloque;
    $segundo_fin = 15 * 60 * ($bloque + 1);
    $sumandos = [ 'promedio_intervalo' ];

    for ($i = $segundo_ini; $i <= $segundo_fin; ++$i) {
      $contribuciones = [ ];

      foreach ($datos as $indice_corrida => $salidas) {
        $corrida = $corridas[$indice_corrida];

        foreach ($salidas as $salida) {
          $contribucion = accede($corrida['Powerprofiles'], $i -
              60 * $salida) / (1 + (int)($i % 900 == 0));

          if ($contribucion != 0) {
            $contribuciones[] = "{$contribucion} salida_{$corrida
                ['LegID']}_{$salida}";
          }
        }
      }

      if (empty($contribuciones)) {
        echo "\tsegundo_{$i} = 0\n";
      }
      else {
        echo "\t", implode(' + ', $contribuciones), " - segundo_{
            $i} <= 0\n";
      }

      $sumandos[] = "segundo_{$i}";
    }

    echo "\t", implode(' - ', $sumandos), " >= 0\n";
```

```
200     }

        // linearization constraints - integer to binary variables

        echo "\\\tlinearization␣constraints␣-␣integer␣to␣binary␣
            variables\n";
205
        foreach ($corridas as $corrida) {
          $suma1 = [ ];
          $sumap = [ ];

210       $pri = $corrida['EarliestDepartureTime'];
          $ult = $corrida['LatestDepartureTime'];

          for ($i = $pri; $i <= $ult; ++$i) {
            $suma1[] = "salida_{$corrida['LegID']}_{$i}";
215         $sumap[] = "{$i}␣salida_{$corrida['LegID']}_{$i}";
          }

          echo "\t", implode('␣+␣', $suma1), "␣=␣1\n";
          echo "\t", implode('␣+␣', $sumap), "␣-␣salida_{$corrida['LegID
              ']}␣=␣0\n";
220     }

        // bounds

        echo "bounds\n";
225
        foreach ($corridas as $corrida) {
          echo "\t{$corrida['EarliestDepartureTime']}␣<=␣salida_{
              $corrida['LegID']}␣<=␣{$corrida['LatestDepartureTime']}\n";
        }

230     $ultimo_segundo = (max(array_keys($intervalos)) + 1) * 15 * 60;

        for ($i = 0; $i <= $ultimo_segundo; ++$i) {
          echo "\tsegundo_{$i}␣>=␣0\n";
        }
235
        // integer variables

        echo "general\n";

240     foreach ($corridas as $corrida) {
          echo "\tsalida_{$corrida['LegID']}\n";
```

```
    }

    // binary variables

    echo "binary\n";

    foreach ($corridas as $corrida) {
      $pri = $corrida['EarliestDepartureTime'];
      $ult = $corrida['LatestDepartureTime'];

      for ($i = $pri; $i <= $ult; ++$i) {
        echo "\tsalida_{$corrida['LegID']}_{$i}\n";
      }
    }

    // end

    echo "end\n";
  } file_put_contents("model{$numero}.lp", ob_get_clean( ));

  // MST file
  ob_start( ); {
    foreach ($corridas as $corrida) {
      echo "salida_{$corrida['LegID']}␣{$corrida['
          CurrentDepartureTime']}\n";
    }
  } file_put_contents("model{$numero}.mst", ob_get_clean( ));
?>
```

B. Modified Mixed Integer Model Generator

```
<?php
  if (!isset($argv[1], $argv[2])) {
    die("Usage: {$argv[0]} #instance #granularity");
  }

  $numero = (int)$argv[1];
  $precision = (int)$argv[2];
  $instancia = json_decode(@file_get_contents("instance_data_{
     $numero}.json.txt"), true);
  $consumo = json_decode(@file_get_contents("power_data_{$numero}.
     json.txt"), true);

  if (!isset($instancia, $consumo)) {
    die("{$argv[0]} {$argv[1]} {$argv[2]}: unable to read the input
       files");
  }

  if ($precision < 1 || $precision > 900 || 900 % $precision != 0) {
    die("{$argv[0]} {$argv[1]} {$argv[2]}: invalid granularity");
  }

  // reconstruction of the input

  $corridas = [ ];

  foreach ($instancia['Trains'] as $trenes) {
    foreach ($trenes['Legs'] as $corrida) {
      $corridas[$corrida['LegID']] = $corrida + [ 'TrainID' =>
         $trenes['TrainID'] ];
    }
  }

  foreach ($consumo['Powerprofiles'] as $onda) {
    $corridas[$onda['LegID']]['Powerprofiles'] = $onda['Powerprofile
       '];
  }
```

```
unset($instancia);
unset($consumo);

// LP file
ob_start( ); {
  // objective function definition

  echo "minimize\n";
  echo "\tpromedio_intervalo\n";

  // constraints

  echo "subject␣to\n";

  function mapea($corridas, $atributo)
  {
    $arr = [ ];

    foreach ($corridas as $corrida) {
      $arr[$corrida[$atributo]][] = $corrida;
    }

    return $arr;
  }

  function compara_3vias($a, $b)
  {
    return 2 * ($a > $b) - ($a != $b);
  }

  function compara_llegadas($corrida1, $corrida2)
  {
    return compara_3vias($corrida1['CurrentDepartureTime'] +
       $corrida1['TravelTime'], $corrida2['CurrentDepartureTime']
       + $corrida2['TravelTime']);
  }

  function compara_salidas($corrida1, $corrida2)
  {
    return compara_3vias($corrida1['CurrentDepartureTime'],
       $corrida2['CurrentDepartureTime']);
  }

  // constraints - headways
```

```
75      echo "\\\theadways\n";

        foreach (mapea($corridas, 'TrackID') as $corridas_via) {
          usort($corridas_via, 'compara_salidas');

80        for ($i = 1; $i < count($corridas_via); ++$i) {
            $corrida = $corridas_via[$i];
            $anterior = $corridas_via[$i - 1];

            echo "\tsalida_{$corrida['LegID']}␣-␣salida_{$anterior['
               LegID']}␣>=␣{$anterior['MinimumHeadwayTime']}\n";
85        }
        }

        // constraints - stops at stations

90      echo "\\\tstops␣at␣stations\n";

        foreach (mapea($corridas, 'TrainID') as $corridas_tren) {
          usort($corridas_tren, 'compara_salidas');

95        for ($i = 1; $i < count($corridas_tren); ++$i) {
            $corrida = $corridas_tren[$i];
            $anterior = $corridas_tren[$i - 1];
            $viaje_con_parada = $anterior['MinimumStoppingTime'] +
               $anterior['TravelTime'];

100         echo "\tsalida_{$corrida['LegID']}␣-␣salida_{$anterior['
               LegID']}␣>=␣{$viaje_con_parada}\n";
          }
        }

        // constraints - connections
105
        echo "\\\tconnections\n";

        $llegadas = mapea($corridas, 'EndStationID');
        $salidas = mapea($corridas, 'StartStationID');
110     $estaciones = array_unique(array_merge(array_keys($llegadas),
           array_keys($salidas)));

        foreach ($estaciones as $estacion) {
          if (!isset($llegadas[$estacion])) {
            $llegadas[$estacion] = [ ];
115       }
```

```
    if (!isset($salidas[$estacion])) {
      $salidas[$estacion] = [ ];
    }

    usort($llegadas[$estacion], 'compara_llegadas');
    usort($salidas[$estacion], 'compara_salidas');

    foreach ($salidas[$estacion] as $salida) {
      foreach ($llegadas[$estacion] as $llegada) {
        $diferencia = $salida['CurrentDepartureTime'] - ($llegada[
            'CurrentDepartureTime'] + $llegada['TravelTime']);

        if ($diferencia < 5) {
          break;
        }

        if ($diferencia > 15) {
          continue;
        }

        if ($salida['TrainID'] != $llegada['TrainID']) {
          $inferior = $llegada['TravelTime'] + 5;
          $superior = $llegada['TravelTime'] + 15;

          echo "\tsalida_{$salida['LegID']}␣-␣salida_{$llegada['
              LegID']}␣>=␣{$inferior}\n";
          echo "\tsalida_{$salida['LegID']}␣-␣salida_{$llegada['
              LegID']}␣<=␣{$superior}\n";
        }
      }
    }
  }

  // linearization constraints - preprocessing

  $intervalos = [ ];

  function accede($onda, $offset)
  {
    return ($offset < 0 || $offset >= count($onda) ? 0 : $onda[
        $offset]);
  }

  foreach ($corridas as $corrida) {
```

```
    $pri = $corrida['EarliestDepartureTime'];
    $ult = $corrida['LatestDepartureTime'];

    for ($i = $pri; $i <= $ult; ++$i) {
      $bloque_ini = (int)($i / 15);
      $bloque_ult = (int)(($i + $corrida['TravelTime']) / 15);

      for ($j = $bloque_ini; $j <= $bloque_ult; ++$j) {
        $intervalos[$j][$corrida['LegID']][] = $i;
      }
    }
  }

  // linearization constraints - power consumption

  echo "\\\tlinearization␣constraints␣-␣power␣consumption\n";

  foreach ($intervalos as $bloque => $datos) {
    $segundo_ini = 15 * 60 * $bloque;
    $segundo_fin = 15 * 60 * ($bloque + 1);
    $sumandos = [ 'promedio_intervalo' ];

    for ($i = $segundo_ini; $i < $segundo_fin; $i += $precision) {
      $contribuciones = [ ];

      foreach ($datos as $indice_corrida => $salidas) {
        $corrida = $corridas[$indice_corrida];

        foreach ($salidas as $salida) {
          $contribucion = 0;

          for ($j = 0; $j < $precision; ++$j) {
            $contribucion += accede($corrida['Powerprofiles'], $i
                + $j - 60 * $salida) / (1 + (int)(($i + $j) % 900
                == 0));
          }

          if ($contribucion != 0) {
            $contribuciones[] = "{$contribucion}␣salida_{$corrida
                ['LegID']}_{$salida}";
          }
        }
      }

      if (empty($contribuciones)) {
```

```
            echo "\tsegmento_{$i} = 0\n";
        }
        else {
          echo "\t", implode(' + ', $contribuciones), " - segmento_{
              $i} <= 0\n";
        }

        $sumandos[] = "segmento_{$i}";
      }

      echo "\t", implode(' - ', $sumandos), " >= 0\n";
    }

    // linearization constraints - integer to binary variables

    echo "\\\tlinearization constraints - integer to binary
        variables\n";

    foreach ($corridas as $corrida) {
      $suma1 = [ ];
      $sumap = [ ];

      $pri = $corrida['EarliestDepartureTime'];
      $ult = $corrida['LatestDepartureTime'];

      for ($i = $pri; $i <= $ult; ++$i) {
        $suma1[] = "salida_{$corrida['LegID']}_{$i}";
        $sumap[] = "{$i} salida_{$corrida['LegID']}_{$i}";
      }

      echo "\t", implode(' + ', $suma1), " = 1\n";
      echo "\t", implode(' + ', $sumap), " - salida_{$corrida['LegID
          ']} = 0\n";
    }

    // bounds

    echo "bounds\n";

    foreach ($corridas as $corrida) {
      echo "\t{$corrida['EarliestDepartureTime']} <= salida_{
          $corrida['LegID']} <= {$corrida['LatestDepartureTime']}\n";
    }

    $ultimo_segundo = (max(array_keys($intervalos)) + 1) * 15 * 60;
```

```
240
        for ($i = 0; $i < $ultimo_segundo; $i += $precision) {
          echo "\tsegmento_{$i} >= 0\n";
        }

245     // integer variables

        echo "general\n";

        foreach ($corridas as $corrida) {
250       echo "\tsalida_{$corrida['LegID']}\n";
        }

        // binary variables

255     echo "binary\n";

        foreach ($corridas as $corrida) {
          $pri = $corrida['EarliestDepartureTime'];
          $ult = $corrida['LatestDepartureTime'];
260
          for ($i = $pri; $i <= $ult; ++$i) {
            echo "\tsalida_{$corrida['LegID']}_{$i}\n";
          }
        }
265
        // end

        echo "end\n";
      } file_put_contents("model{$numero}_g{$precision}.lp",
         ob_get_clean( ));
270
      // MST file
      ob_start( ); {
        foreach ($corridas as $corrida) {
          echo "salida_{$corrida['LegID']} {$corrida['
             CurrentDepartureTime']}\n";
275     }
      } file_put_contents("model{$numero}.mst", ob_get_clean( ));
    ?>
```

C. Solution Formatter

```php
<?php
  if (!isset($argv[1], $argv[2])) {
    die("Usage: {$argv[0]} #instance mst_file");
  }

  $numero = (int)$argv[1];
  $archivo = $argv[2];
  $consumo = json_decode(@file_get_contents("power_data_{$numero}.
     json.txt"), true);
  $respuesta = @file($archivo, FILE_IGNORE_NEW_LINES|
     FILE_SKIP_EMPTY_LINES);

  if (!isset($respuesta, $consumo) || empty($respuesta)) {
    die("{$argv[0]} {$argv[1]} {$argv[2]}: unable to read the input
       files");
  }

  // MST cleaning, JSON file generation

  $res = new stdClass( );
  $res->Legs = new stdClass( );
  $corridas = [ ];
  $mst = [ ];

  foreach ($respuesta as $fila) {
    $capturas = [ ];

    if (preg_match('/salida_(\d+) ([-+]?\d+(\.\d+([eE][-+]?\d+)?)?)/
       ', $fila, $capturas)) {
      $id_corrida = $capturas[1];
      $salida_corrida = (int)round($capturas[2]);

      $res->Legs->{$id_corrida} = $salida_corrida;
      $corridas[$id_corrida]['DepartureTime'] = $salida_corrida;
      $mst[] = "salida_{$id_corrida} {$salida_corrida}";
    }
  }
```

```
35    file_put_contents($archivo, implode("\n", $mst));
      file_put_contents(preg_replace('/\.mst$/i', '', $archivo).'.json.
         txt', json_encode($res, JSON_PRETTY_PRINT));

      // objective function computation

40    foreach ($consumo['Powerprofiles'] as $onda) {
        $corridas[$onda['LegID']]['Powerprofiles'] = $onda['Powerprofile
           '];
      }

      $segundos = array_fill(0, 17 * 15 * 60 + 1, 0);
45
      foreach ($corridas as $corrida) {
        $inicial = 60 * $corrida['DepartureTime'];

        foreach ($corrida['Powerprofiles'] as $segundo => $consumo) {
50        $segundos[$inicial + $segundo] += $consumo;
        }
      }

      $intervalos = [ ];
55
      foreach (range(0, 16) as $bloque) {
        $segundo_ini = 15 * 60 * $bloque;
        $segundo_fin = 15 * 60 * ($bloque + 1);
        $sumando = 0;
60
        for ($i = $segundo_ini; $i <= $segundo_fin; ++$i) {
          $sumando += max($segundos[$i], 0) / (1 + (int)($i % 900 == 0))
             ;
        }

65      $intervalos[] = $sumando;
      }

      echo "objective␣function␣value:␣", max($intervalos) / 900, "\n";
    ?>
```

D. Gurobi Interface

```
#include "gurobi_c++.h"
#include <fstream>
#include <iostream>
#include <string>

class callback_reto : public GRBCallback {
public:
   callback_reto(GRBVar* v, int n, const char* i)
   : vars_(v), nvars_(n), instancia_(i)
   {
   }

   void callback( )
   {
      if (where == GRB_CB_MIPSOL) {
         std::ofstream ofs(instancia_ + std::string("_temp") + std::
            to_string(getDoubleInfo(GRB_CB_MIPSOL_OBJ)) + std::
            string(".mst"));
         auto val = getSolution(vars_, nvars_);

         for (int i = 0; i < nvars_; ++i) {
            ofs << vars_[i].get(GRB_StringAttr_VarName) << ' ' <<
               val[i] << '\n';
         }

         delete[] val;
      }
   }

private:
   GRBVar* vars_;
   int nvars_;

   const char* instancia_;
};

int main(int argc, const char** argv)
```

```
{
    if (argc < 5) {
       std::cout << "Usage: " << argv[0] << ' ' << "InputFile
          ResultFile MIPFocus file\n";
       return 0;
    }

    auto env = GRBEnv( );
    env.set(GRB_IntParam_MIPFocus, std::atoi(argv[3]));
    env.set(GRB_DoubleParam_MIPGap, 0);

    auto modelo = GRBModel(env, argv[4]);
    modelo.read(argv[1]);

    auto callback = callback_reto(modelo.getVars( ), modelo.get(
       GRB_IntAttr_NumVars), argv[4]);
    modelo.setCallback(&callback);
    modelo.optimize( );

    if (modelo.get(GRB_IntAttr_Status) == GRB_OPTIMAL) {
       modelo.write(argv[2]);
    }
}
```