New Normal – Cinema Seating Planning

Edo Mangelaars
Utrecht University, The Netherlands
e.c.mangelaars@students.uu.nl

Tijmen van den Pol
Utrecht University, The Netherlands
t.vandenpol@students.uu.nl

Matthijs Wolters
Utrecht University, The Netherlands
m.s.wolters@students.uu.nl

Daniele Di Grandi
Utrecht University, The Netherlands
d.digrandi@students.uu.nl

Bernd van den Hoek
Utrecht University, The Netherlands
b.vandenhoek@uu.nl

Abstract

Due to COVID-19 and the requirement for cinemas to seat groups of customers at least 1.5m apart, cinemas are unable to operate at full capacity. We propose and compare several algorithms to maximize the number of people seated in a cinema while maintaining 1.5m distance.

For the "offline" version of the cinema seating problem, where the sizes of all groups of customers are known in advance, we compare an Integer Linear Programming method, which guarantees optimality but can have prohibitively long runtimes for very large cinemas, with two graph-based approximations which we call FirstFit and LeastBlockFit, which are faster but may not be optimal.

For the "online" version of the problem, where groups of customers are seated as they come in, we prove that our online FirstFit algorithm guarantees a competitive ratio of $\frac{1}{8}$ compared to an optimal solution with the assumption of a simplified cinema with no gaps or corridors.

We show that our algorithms can compute an optimal seating arrangement within a reasonable timeframe for cinemas up to about 1000 seats, and a seating arrangement within 25% of optimal in most cases for cinemas of any reasonable size. We also show that our online algorithm can compute a seating arrangement within 30% of optimal for cinemas of any reasonable size.

2012 ACM Subject Classification Computing methodologies → Planning and scheduling

Keywords and phrases Cinema seating, Optimization, Independent set

Supplementary Material Source code of the programs described in this paper can be found at `https://gitlab.com/edo-codes/ads-project-2020`. Test instances and results, as also mentioned in the result section of this paper, can be found at `https://webspace.science.uu.nl/~3898733/ads-2020/`

1 Introduction

Due to the ramifications of the COVID-19 virus, businesses have to find new ways of housing customers safely. This is particularly relevant to cinemas, where one would usually sit next to or near strangers, who at this time could possibly carry a significant viral load. As such cinemas need ways to house groups of people in a manner that allows the groups of people to sit together but keeps other groups of people at least 1.5 meters away from each other. Versions of this problem have been discussed before such as the unfriendly theater seating problem [2], which is a maximum independent set problem. Maximum independent set problems have been proven to be NP-complete [3]. This means that they are optimally solvable in exponential time. The goal of this paper is to discuss different approaches to solve the cinema seating problem, with varying levels of accuracy and time complexity.

There are two versions of the cinema seating problem, the offline and the online versions. The offline problem is when the complete number and sizes of all groups are known beforehand. This allows for some reasoning about how to fit the maximum number of groups in. In the online problem, groups arrive one by one and there is no way to reason about the best way to fit as many people in the cinema, other than the strategy defined beforehand. It is also not possible to move an already seated group.

We tried a number of strategies for the offline version of the problem. The Integer Linear Programming solved small instances of the offline problem optimally, however it was very slow. The FirstFit algorithms performed much faster and were thus able to solve much larger problems with relative ease, but lacked optimality.

We approached the online version of this problem in a similar manner to the FirstFit algorithms. We will discuss the competitive ratio of this approach and compare the results from the online and offline versions.

2 Definitions

2.1 Graph

A graph is an ordered pair of pair of disjoint sets (V, E), such that E is a subset of the set $V^{(2)}$ of unordered pairs of V. [1]

Using this general definition for a graph it is possible to model a large number of problems. In this paper graphs will be used for a number of different applications, from representing the cinema and therefore the search space to expanding the cinema according to every possible combination of groups that could be seated. The set V is known as the set of vertices or nodes – these are the objects to be modeled. The set E is known as the set of edges – an edge represents a relation between modeled objects (vertices).

2.2 Independent sets

Given a graph $G = (V, E)$, a set $S \subseteq E$ is an independent set if for all $v, w \in S, (v, w) \notin E$. In other words, it is a subset of vertices where none of the vertices are connected by an edge in G.

Finding the largest possible independent set, called a maximum independent set, is NP-complete [3]. However, a lot of research has been done towards finding efficient (though still not polynomial) algorithms for the maximum independent set problem.

3 Offline problem

3.1 Integer Linear Programming

As a first attempt, we attacked the offline cinema problem by designing an integer linear programming model.

The advantages to using this method are predominantly concerning the optimal solution: by using well-known solving techniques like branch and bound and cutting planes, it is certain that the solution of this model is optimal because of the optimality proofs behind these algorithms.

Moreover, we implemented the model in a well-known commercial solver, in order to be assured that the problem would be solved in the most efficient way, because of the best practice developed by the software owners. For this reason, the commercial solver gurobi (and especially the library "gurobipy" for Python) was chosen for the implementation, using the academic license that Utrecht University provided.

However, suspecting that this problem is NP-hard, the modeling structure will directly determine the runtime necessary to solve this problem, especially for large inputs. The next paragraphs will explain in detail the modeling structure we decided to adopt.

3.1.1 Parameters

We define the following parameters:

$n =$ number of rows in the cinema	$n \in [1, 1000]$,	n int
$m =$ number of columns in the cinema	$m \in [1, 1000]$,	m int
$k =$ size of the group of people	$k \in [1, 8]$,	k int
$s_k =$ number of groups of size k that have to be placed		

3.1.2 Variables

$$y_{ijk} = \begin{cases} 1 & \text{if a } (i, j) \text{ seat is assigned to a group of size } k \\ 0 & \text{otherwise} \end{cases}$$

$$x_{ijk} = \begin{cases} 1 & \text{if a group of size } k \text{ starts seating from position } (i, j) \\ 0 & \text{otherwise} \end{cases}$$

In this way, it is possible to model the cinema like a simple grid, where each square (i, j) contains 16 variables:

x_{111} y_{111} x_{112} y_{112} $\dots$ x_{118} y_{118}	x_{121} y_{121} x_{122} y_{122} $\dots$ x_{128} y_{128}	$\dots$	x_{1m1} y_{1m1} x_{1m2} y_{1m2} $\dots$ x_{1m8} y_{1m8}
x_{211} y_{211} x_{212} y_{212} $\dots$ x_{218} y_{218}	x_{221} y_{221} x_{222} y_{222} $\dots$ x_{228} y_{228}	$\dots$	x_{2m1} y_{2m1} x_{2m2} y_{2m2} $\dots$ x_{2m8} y_{2m8}
$\dots$	$\dots$	$\dots$	$\dots$
x_{n11} y_{n11} x_{n12} y_{n12} $\dots$ x_{n18} y_{n18}	x_{n21} y_{n21} x_{n22} y_{n22} $\dots$ x_{n28} y_{n28}	$\dots$	x_{nm1} y_{nm1} x_{nm2} y_{nm2} $\dots$ x_{nm8} y_{nm8}

Thus, for example, if $y_{343} = 1$ it means that in line $n = 3$ and column $m = 8$ there is seated one person that is part of a group of $k = 3$ people. And also, if $x_{343} = 1$ it means that that specific group of size $k = 3$ starts precisely from line $n = 3$ and column $m = 8$, thus it is expected that both y_{353} and y_{363} would be equal to 1.

In the image it can be seen also that the notation used is: $i = 1, \dots, n$ and $j = 1, \dots, m$.

Note that it is impossible to write the model with only one of the two sets of variables just defined, and the reason is because of the constraint F, discussed later.

3.1.3 Objective function

As the objective function, we try to maximize the total amount of people in the cinema. That is equal to maximizing the place where each group starts sitting times the size of that group:

$$\max z = \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{8} k x_{ijk}$$

Note that we can't simply maximize y_{ijk} because of how the constraints of this model are defined.

3.1.4 Constraints

The constraints are divided by alphabet letters. In this way, we can easily match the constraint with the equivalent coded constraint in gurobi.

A) Preprocessing constraints: for (i, j) positions without a chair (0 in the given input), we have to enforce the corresponding y_{ijk} variables to be equal to 0, because no one can be seated there:

$$\sum_{k=1}^{8} y_{ijk} = 0 \quad \forall i = 1, \dots, n;\ \forall j = 1,\ \dots, m \text{ where } (i, j) = 0 \text{ in the given input}$$

Note that enforcing also $x_{ijk} = 0$ for (i, j) positions without a chair would be redundant: constraint B ensures that if $y_{ijk} = 0$ then also $x_{ijk} = 0$.

B) If a group of size k is placed in a position (i, j), (namely, if $x_{ijk} = 1$), then y_{ijk} is forced to be equal to 1, but if $y_{ijk} = 1$ then x_{ijk} has to be free to change:

$$x_{ijk} \leq y_{ijk} \quad \forall i = 1, \dots, n;\ \forall j = 1, \dots,\ m;\ \forall k = 1, \dots, 8$$

This set of constraints is necessary because we are enforcing the correspondent y_{ijk} variable to 1 if a group of size k starts in the square (i, j) but it's not true the other way around: if $y_{ijk} = 1$ for some reason, it's not true that this specific group starts in the correspondent (i, j) square, therefore, the correspondent x_{ijk} could be either 0 or 1. But it's true that if $y_{ijk} = 0$, then the square (i, j) is off-limits, meaning that no groups of any size should start in that square, therefore, also $x_{ijk} = 0$.

C) The amount of people seated have to be less or equal than the total people given in the input:

$$\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ijk} \leq k s_k \quad \forall k = 1, \dots, 8$$

This set of constraints precludes the possibility to have more people seated, for each group size, with respect to the people given in input.

D) No more than one group should be placed in the same seat. We have split this constraint in two sets of constraints:

D1) $$\sum_{k=1}^{8} x_{ijk} \leq 1 \quad \forall i = 1, \dots, n;\ \forall j = 1, \dots, m$$

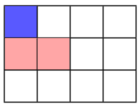
D2) $$\sum_{k=1}^{8} y_{ijk} \leq 1 \quad \forall i = 1, \dots, n;\ \forall j = 1, \dots, m$$

The D1 set of constraints ensures that only one group per size could start to occupy a certain seat. The D2 set of constraints ensures the same thing but they are necessary in order to find a feasible solution.

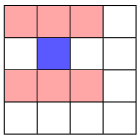
E) Constraints regarding the 1.5m distance rule. We have also split these constraints in two: the first part ensures that if some $y_{ijk} = 1$ it means that a person is seated in that chair, therefore, every chair in front, under and diagonally should be enforced to be equal to 0. The second part ensures that if a chair is selected to let seat-down a group of size k, the 2 chairs after that group (namely, the 2 chairs after k people) have to be enforced equal to 0.

First part

This first part also has been split in 9 sets of constraints. The problem with writing them in only 1 set of constraints was the dependence of these constraints on the position of where they have to be written. For instance, consider the following examples:

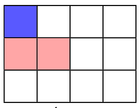


If a person is seated in the blue square, then the first part of this constraint should enforce only the variables in the red squares to be equal to 0.

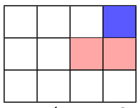


Here is the same as above, but for this blue square, there is a different pattern of red squares to be enforced equal to 0.

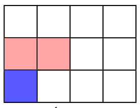
Because of this problem, we have split these constraints according to the following patterns:

E1)  Pattern: $i = 1, j = 1$

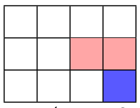
$$M\left(1 - \sum_{k=1}^{8} y_{ijk}\right) \geq \sum_{k=1}^{8} y_{i+1,j,k} + \sum_{k=1}^{8} y_{i+1,j+1,k} \quad for\ i = 1;\ j = 1$$

E2)  Pattern: $i = 1, j = m$

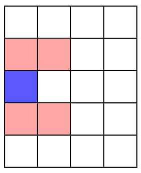
$$M\left(1 - \sum_{k=1}^{8} y_{ijk}\right) \geq \sum_{k=1}^{8} y_{i+1,j,k} + \sum_{k=1}^{8} y_{i+1,j-1,k} \quad for\ i = 1;\ j = m$$

E3)  Pattern: $i = n, j = 1$

$$M\left(1 - \sum_{k=1}^{8} y_{ijk}\right) \geq \sum_{k=1}^{8} y_{i-1,j,k} + \sum_{k=1}^{8} y_{i-1,j+1,k} \quad for\ i = n;\ j = 1$$

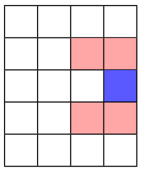
E4)  Pattern: $i = n, j = m$

$$M\left(1 - \sum_{k=1}^{8} y_{ijk}\right) \geq \sum_{k=1}^{8} y_{i-1,j,k} + \sum_{k=1}^{8} y_{i-1,j-1,k} \quad for\ i = n;\ j = m$$

E5)  Pattern: $j = 1$, for each $i \neq 1, n$

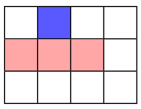
$$M\left(1 - \sum_{k=1}^{8} y_{ijk}\right) \geq \sum_{k=1}^{8} y_{i+1,j,k} + \sum_{k=1}^{8} y_{i-1,j,k} + \sum_{k=1}^{8} y_{i+1,j+1,k} + \sum_{k=1}^{8} y_{i-1,j+1,k}$$

$$for\ j = 1;\ \forall i = 2, \dots, n-1$$

E6)  Pattern: $j = m$, for each $i \neq 1, n$

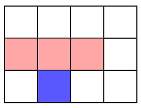
$$M\left(1 - \sum_{k=1}^{8} y_{ijk}\right) \geq \sum_{k=1}^{8} y_{i+1,j,k} + \sum_{k=1}^{8} y_{i-1,j,k} + \sum_{k=1}^{8} y_{i-1,j-1,k} + \sum_{k=1}^{8} y_{i+1,j-1,k}$$

$$for\ j = m;\ \forall i = 2, \dots, n-1$$

E7)  Pattern: $i = 1$, for each $j \neq 1, m$

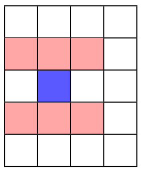
$$M\left(1 - \sum_{k=1}^{8} y_{ijk}\right) \geq \sum_{k=1}^{8} y_{i+1,j,k} + \sum_{k=1}^{8} y_{i+1,j+1,k} + \sum_{k=1}^{8} y_{i+1,j-1,k}$$

$$for\ i = 1;\ \forall j = 2, \dots, m-1$$

E8)  Pattern: $i = n$, for each $j \neq 1, m$

$$M\left(1 - \sum_{k=1}^{8} y_{ijk}\right) \geq \sum_{k=1}^{8} y_{i-1,j,k} + \sum_{k=1}^{8} y_{i-1,j-1,k} + \sum_{k=1}^{8} y_{i-1,j+1,k}$$

$$for\ i = n;\ \forall j = 2, \dots, m-1$$

E9)  Pattern: generic middle square (not on first/last column/row)

$$M\left(1 - \sum_{k=1}^{8} y_{ijk}\right) \geq \sum_{k=1}^{8} y_{i+1,j,k} + \sum_{k=1}^{8} y_{i-1,j,k} + \sum_{k=1}^{8} y_{i-1,j-1,k} + \sum_{k=1}^{8} y_{i+1,j+1,k}$$
$$+ \sum_{k=1}^{8} y_{i-1,j+1,k} + \sum_{k=1}^{8} y_{i+1,j-1,k}$$

$$\forall i = 2, \dots, n-1;\ \forall j = 2, \dots, m-1$$

In all of these 9 constraints, M is a value big enough to ensure correctness of a constraint if $y_{ijk} = 0$ on the left-hand side. In this case, we could have set a different M for each constraint, but for simplicity, we decided to set $M = 48$ (upper bound of all the M possible, given by the E9 constraint) for each constraint, since it is the maximum value reachable on the right-hand side.

Second part

E10) This second part of the 1.5m distance constraints deals with enforcing the 2 seats (on the same row) to be 0 after a group of k size is placed on a certain seat:

$$N\,(1 - x_{ijk}) \geq \sum_{s=1}^{8} \sum_{L=0}^{r} y_{i,j+k+L,s} \quad \forall i = 1,\ldots,n;\ \forall j = 1,\ldots,m-k;\ \forall k = 1,\ldots,8$$

where

$$r = \begin{cases} 1 & \text{if } m - j - k > 0 \\ 0 & \text{otherwise} \end{cases}$$

and where N means the same thing as M before, but in this case, setting $N = 16$ is sufficient.

These constraints are written as $\forall j = 1,\ldots,m-k$ in order to not exceed the length of the cinema, with a special attention to the indexes even in the second summation.

F) Person of the same group have to be placed near each other:

$$(k-1)x_{ijk} \leq \sum_{L=1}^{k-1} y_{i,j+L,k}$$
$$\forall i = 1,\ldots,n;\ \forall j = 1,\ldots,m-1;\ \forall k = 2,\ldots,\min(m+1-j,8)$$

Even here, we have written the constraints such that the length of the cinema is not exceeded.

However, in this set of constraints, only one set of variables (x_{ijk} or y_{ijk}) is not enough to describe the problem for the following reason:

Suppose these constraints were written with only one set of variables:

$$(k-1)y_{ijk} \leq \sum_{L=1}^{k-1} y_{i,j+L,k}$$
$$\forall i = 1,\ldots,n;\ \forall j = 1,\ldots,m-1;\ \forall k = 2,\ldots,\min(m+1-j,8)$$

This will clearly cause a recursion problem: since these constraints are written for each square, if for example a group of 3 people start seating from square (3,1) (so: $y_{313} = 1$) this implies that the two next seats are enforced to be equal to 1, that is, both y_{323} and y_{333} are enforced to be equal to 1.

But since that these constraints are also written for both y_{323} and y_{333} on the left-hand side, when they will assume the value of 1 this wrongly implies that y_{343} and y_{353} should also be equal to 1, but this in turn implies that also y_{363} and y_{373} would be equal to 1, and so on with all the variables in the same row and with the same k: they will wrongly assume the value of 1.

Instead, by introducing a new set of variables x_{ijk} that will flag only the first place where a group of size k starts seating, this recursion problem is solved.

G) Enforce $x_{ijk} = 0$ if the solver is nearly at the end of the cinema and is trying to seat a group of k people with k greater than the number of seats left:

$$\sum_{k=m-j+2}^{8} x_{ijk} = 0 \quad \forall i = 1,\ldots,n;\ \forall j = 1,\ldots,m$$

H) We also need constraints that ensure, if some $x_{ijk} = 1$ with a group of size k is in the next $(k-1)$ seats, all the x_{ijk} should be equal to 0. That is, no other groups are allowed to start sitting there:

$$(k-1)(1-x_{ijk}) \geq \sum_{L=1}^{k-1} x_{i,j+L,k}$$
$$\forall i = 1,\ldots,n;\ \forall j = 1,\ldots,m-1;\ \forall k = 2,\ldots,\min(m+1-j,\ 8)$$

It turns out that this set of constraints will speed up the computation a lot, because when a decision of some $x_{ijk} = 1$ is made, this implies that a whole set of other variables will assume a value of 0, making it easier to evaluate the situation.

I) Lastly, all the variables used are binary:

$$x_{ijk}, y_{ijk} \in \{0,1\} \quad \forall i = 1,\ldots,n;\ \forall j = 1,\ldots,m;\ \forall k = 1,\ldots,8$$

3.1.5 ILP model

Thus, the whole model will be as follows:

$$\max z = \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{8} k x_{ijk}$$

Subject to

$$\sum_{k=1}^{8} y_{ijk} = 0 \quad \forall i = 1,\ldots,n;\ \forall j = 1,\ \ldots,m \text{ where } (i,j) = 0 \text{ in the given input} \tag{A}$$

$$x_{ijk} \leq y_{ijk} \quad \forall i = 1,\ldots,n;\ \forall j = 1,\ldots,\ m;\ \forall k = 1,\ldots,8 \tag{B}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ijk} \leq k s_k \quad \forall k = 1,\ldots,8 \tag{C}$$

$$\sum_{k=1}^{8} x_{ijk} \leq 1 \quad \forall i = 1,\ldots,n;\ \forall j = 1,\ldots,m \tag{D1}$$

$$\sum_{k=1}^{8} y_{ijk} \leq 1 \quad \forall i = 1,\ldots,n;\ \forall j = 1,\ldots,m \tag{D2}$$

$$M\left(1-\sum_{k=1}^{8} y_{ijk}\right) \geq \sum_{k=1}^{8} y_{i+1,j,k} + \sum_{k=1}^{8} y_{i+1,j+1,k} \quad for\ i = 1;\ j = 1 \tag{E1}$$

$$M\left(1-\sum_{k=1}^{8} y_{ijk}\right) \geq \sum_{k=1}^{8} y_{i+1,j,k} + \sum_{k=1}^{8} y_{i+1,j-1,k} \quad for\ i = 1;\ j = m \tag{E2}$$

$$M\left(1-\sum_{k=1}^{8} y_{ijk}\right) \geq \sum_{k=1}^{8} y_{i-1,j,k} + \sum_{k=1}^{8} y_{i-1,j+1,k} \quad for\ i = n;\ j = 1 \tag{E3}$$

$$M\left(1-\sum_{k=1}^{8} y_{ijk}\right) \geq \sum_{k=1}^{8} y_{i-1,j,k} + \sum_{k=1}^{8} y_{i-1,j-1,k} \quad for\ i = n;\ j = m \tag{E4}$$

$$M\left(1-\sum_{k=1}^{8} y_{ijk}\right) \geq \sum_{k=1}^{8} y_{i+1,j,k} + \sum_{k=1}^{8} y_{i-1,j,k} + \sum_{k=1}^{8} y_{i+1,j+1,k} + \sum_{k=1}^{8} y_{i-1,j+1,k}$$
$$for\ j=1;\ \forall i=2,\dots,n-1 \quad \text{(E5)}$$

$$M\left(1-\sum_{k=1}^{8} y_{ijk}\right) \geq \sum_{k=1}^{8} y_{i+1,j,k} + \sum_{k=1}^{8} y_{i-1,j,k} + \sum_{k=1}^{8} y_{i-1,j-1,k} + \sum_{k=1}^{8} y_{i+1,j-1,k}$$
$$for\ j=m;\ \forall i=2,\dots,n-1 \quad \text{(E6)}$$

$$M\left(1-\sum_{k=1}^{8} y_{ijk}\right) \geq \sum_{k=1}^{8} y_{i+1,j,k} + \sum_{k=1}^{8} y_{i+1,j+1,k} + \sum_{k=1}^{8} y_{i+1,j-1,k}$$
$$for\ i=1;\ \forall j=2,\dots,m-1 \quad \text{(E7)}$$

$$M\left(1-\sum_{k=1}^{8} y_{ijk}\right) \geq \sum_{k=1}^{8} y_{i-1,j,k} + \sum_{k=1}^{8} y_{i-1,j-1,k} + \sum_{k=1}^{8} y_{i-1,j+1,k}$$
$$for\ i=n;\ \forall j=2,\dots,m-1 \quad \text{(E8)}$$

$$M\left(1-\sum_{k=1}^{8} y_{ijk}\right) \geq \sum_{k=1}^{8} y_{i+1,j,k} + \sum_{k=1}^{8} y_{i-1,j,k} + \sum_{k=1}^{8} y_{i-1,j-1,k} + \sum_{k=1}^{8} y_{i+1,j+1,k}$$
$$+\sum_{k=1}^{8} y_{i-1,j+1,k} + \sum_{k=1}^{8} y_{i+1,j-1,k}$$
$$\forall i=2,\dots,n-1;\ \forall j=2,\dots,m-1 \quad \text{(E9)}$$

$$N\left(1-x_{ijk}\right) \geq \sum_{s=1}^{8}\sum_{L=0}^{r} y_{i,j+k+L,s} \quad \forall i=1,\dots,n;\ \forall j=1,\dots,m-k;\ \forall k=1,\dots,8$$
$$\text{where } r = \begin{cases} 1 & \text{if } m-j-k>0 \\ 0 & \text{otherwise} \end{cases} \quad \text{(E10)}$$

$$(k-1)x_{ijk} \leq \sum_{L=1}^{k-1} y_{i,j+L,k}$$
$$\forall i=1,\dots,n;\ \forall j=1,\dots,m-1;\ \forall k=2,\dots,\min(m+1-j,8) \quad \text{(F)}$$

$$\sum_{k=m-j+2}^{8} x_{ijk} = 0 \quad \forall i=1,\dots,n;\ \forall j=1,\dots,m \quad \text{(G)}$$

$$(k-1)(1-x_{ijk}) \geq \sum_{L=1}^{k-1} x_{i,j+L,k}$$

$$\forall i=1,\ldots,n;\ \forall j=1,\ldots,m-1;\ \forall k=2,\ldots,\min(m+1-j,\ 8) \quad \text{(H)}$$

$$x_{ijk}, y_{ijk} \in \{0,1\} \quad \forall i=1,\ldots,n;\ \forall j=1,\ldots,m;\ \forall k=1,\ldots,8 \quad \text{(I)}$$

We can easily achieve an interesting extension of this model: the constraint to accept group sizes of at most 8 can be relaxed. This is simple to change. Call g the maximum size permitted (so $k = 1,\ldots,g$) and by only replacing every 8 that shows up in the model with g, it will still be completely solvable without changing anything in the logic of the constraints, except the values of N and M that will become: $N = 2g$ and $M = 6g$.

After having implemented the model on gurobi, the expected outcome was confirmed: the model can find an optimal solution in a reasonable time, but only for inputs that are not too large (results are discussed in the appropriate chapter).

Due to having implemented this on gurobi, the model can also handle larger inputs by simply stopping the computation before it is finished: this way, it will output the best solution found up to that moment, but for very large cinemas ($n, m > 250$) it doesn't find a first solution at all. Nonetheless, considering the sizes of real cinemas, this ILP model is fully functional and will always find the optimal solution in reasonable time. However, in order to satisfy all the input sizes permitted, we need to find something that can handle inputs up to $n = 1000$ and $m = 1000$.

3.2 FirstFit Algorithm

The next attempt we made to solve this problem was to make a greedy graph based algorithm, which was named FirstFit (FF). The decision to design such an algorithm was based on the problem of the ILP solver, that while optimal (i.e. fills the cinema with the most seats possible), it is not very efficient and thus very slow. The greedy algorithm is not optimal, but is very fast.

The FF approach means that groups are placed on the first possible position found, this is the greedy aspect of the algorithm. The cinema is searched from top left to bottom right. For this approach we turned the cinema into a graph-like structure. The nodes became the seats and the edges are to the surrounding open seats, which are blocked should someone sit in that particular seat. Using this structure means finding a correct solution is trivial, as it is made impossible to place someone in a blocked or taken seat.

The algorithm searches the rows of the cinema for adjacent, open and not blocked seats. It starts the search with the biggest available group to ensure that no smaller group blocks a group of adjacent seats in which a larger group could sit. Once a large enough group of adjacent seats is found, the graph is immediately updated to place the group of people and block the correct seats. The algorithm does not consider if there is a location which would be more optimal for the group of people to be placed in. This algorithm also does not consider whether a combination of smaller groups could place more people than a number of bigger groups. These are the reasons this algorithm is not optimal.

The time complexity of this algorithm is $\mathcal{O}(n)$ because the cinema is searched at most once for every group in the input. This number is actually less than one as once a group of a particular size cannot be placed anymore, it is unnecessary to continue searching for groups of that size.

3.2.1 Heuristics

There are heuristics which could move the FirstFit algorithm towards an optimal solution. One has actually already been discussed; starting the search with the largest group possible. Due to the greedy nature of the algorithm and the fact we wanted to avoid exponential time solutions means that finding an optimal solution is next to impossible. The reason a exponential time solution was to be avoided is that the ILP solver already solves in exponential time, while the model might be able to be adjusted so that it is faster than it is currently, a different, much faster, approach was preferred.

As mentioned in the previous section, the FF algorithm is not optimal for two reasons. The algorithm does not consider the surrounding area when placing a group and it does not consider whether different combinations of groups could fit more people. The first problem is the one tackled by the heuristic described in the following paragraph. To tackle the second problem would require us to design very complex heuristics or a more brute force approach, which would mean the time complexity of the algorithm would increase significantly, especially for large instances. Furthermore, the ILP solver tackles these problems and it was the our intention to use the solutions given by the fast algorithms as a first solution for the ILP solver. As such, the solution did not have to be optimal and the focus lay on efficiency over optimality. Due to time constraints it was not possible to implement this combination of approaches.

The heuristic to tackle the first problem, where the algorithm does not consider the surrounding area should be clear. The group of people should be placed in the position which blocks the least amount of open seats. However, there are many ways of implementing this heuristic and we suspected that they would increase the runtime by a significant amount. Therefore we tried two different approaches.

The first method of implementing this heuristic is sorting the cinema by the rows that have the lowest sum of edges per seat to a row above or below them. It was assumed that this would be a more general bound on an optimal result. Rather than searching the whole cinema the algorithm searches the rows that blocks the least amount of seats. Once a group is placed the list of rows has to be updated to reflect how many seats have been blocked. Sorting this list however increased runtime by a significant amount and it was found that this heuristic overfit slightly. There is no guarantee that the row which can block the least amount of seats actually contains the best position for the group currently being placed. Due to this, the results for this algorithm were omitted from this paper.

Another way to implement this heuristic is to abandon the greedy aspect of the algorithm. In this case the algorithm searches the whole cinema once for each input and evaluates the best position possible for the group (i.e. the position in which the group blocks the least amount of open seats). This algorithm will be referred to as Least Block Fit (LBF). Rather than the greedy FF search, every time a number of adjacent open seats of the same size as the group is found, the algorithm checks the amount of seats that would be blocked by placing the group in this position. The location for which the group blocks the least amount of seats is recorded as the best position and the group is placed once the whole cinema has been searched. The best position was only updated if a position was found that was strictly better than the previous best. This means that positions to the top left will be preferred to a degree, as the basis for this algorithm is still the same as FF.

3.3 Independent set on graph

We have also started a third attempt at solving the problem, but were not able to finish it in time for the deadlines, so we unfortunately don't know how this idea would perform. Nonetheless, we will describe it in broad strokes here for completeness' sake.

The idea is to represent the cinema as a graph, and transform that graph in such a way that solving the problem of finding the most optimal seating arrangement reduces to a modified version of the maximum independent set problem.

3.3.1 Constructing the graph

First, we represent the cinema as an undirected graph with a vertex for every seat, and for every seat s, we add an edge (s, t) to every neighboring seat t that would be 'blocked' if s were occupied in order to maintain the 1.5m distance, which are at most 10 seats. The vertices of this graph can be interpreted as the set of all possible seatings of a group of 1, where no two seatings that share an edge can both be occupied.

To account for groups of size two, a vertex is added for every seat where it is possible to seat the leftmost person of a group of two, i.e. every seat that also has a seat one place to the right. For every vertex s of this new set of vertices, an edge (s, t) is added for both the vertex representing a group of 1 and, if present, the vertex representing a group of 2 for every neighboring seat, plus an edge to the vertex representing a group of 1 for that same seat.

Nodes and edges for groups of size three up to eight are added analogously.

3.3.2 Finding the seating arrangement

Though we have not been able to implement this part before the deadline, we can provide a sketch of a possible solution.

The graph we have constructed now contains a vertex for every possible seating of a group of any possible size (up to 8), and an edge between every vertex that would block another seating if a group is seated there. Thus, the problem of finding a seating arrangement for multiple groups is now conceptually (but not computationally) quite simple. A valid seating arrangement corresponds to an independent set of our graph that contains a vertex for every group of people, where that vertex corresponds to the seat of the leftmost person of that group and represents the right group size.

Finding the optimal seating of multiple groups of people is slightly more complicated: it is an independent set that corresponds to the right group sizes with the greatest sum of the group sizes of the groups that are seated.

With enough time, our approach would have been either to adapt an maximum independent set algorithm to this more complicated problem, or to create an integer linear programming model with the graph exported as a set of constraints plus constraints for the restrictions mentioned above.

4 Online problem

For the online problem we will use the FirstFit algorithm. In most ways the FirstFit algorithm is similar to the one proposed to the algorithm proposed in 3.2, only with the online version of the problem the groups will come one by one. In this section we will analyze the competitive ratio of the algorithm for any cinema. After that we will show that the competitive ratio in (part of) a cinema without gaps is better than the overall competitive ratio.

4.1 Competitive ratio

In this section we will proof the competitive ratio of the FirstFit algorithm in any cinema. In order to do this we will first construct a worst case scenario (which does not necessarily apply to the FirstFit algorithm). After that we will show that in fact FirstFit always performs better than this worst case scenario.

▶ **Lemma 1.** *The worst case scenario for an input I has a ratio $\frac{ALG(I)}{OPT(I)}$ of $\frac{1}{25}$*

Proof. Let's first take a look at the situation in figure 1. When placing a single person in a cinema it affects the neighbouring seats. We could in fact have blocked 3 groups of 8 people as shown in figure 2. The potential number of seated people was 25, but instead we placed a single person therefore the ratio of this situation is $\frac{1}{25}$. We can, however, have bigger groups. Let's consider a group of 2 people. Instead of 3 blocked groups of 8, we could now potentially have blocked 4 groups of 8 (one group of 8 in each corner of the blocked seats). For groups bigger than 2 we have the same situation except for the fact that we can still place $n - 2$ people extra in the middle as shown in figure 3. The ratio for a group of 2 people is therefore $\frac{1}{16}$ and it becomes better for larger groups. The worst case ratio $\frac{ALG(I)}{OPT(I)}$ is therefore $\frac{1}{25}$.

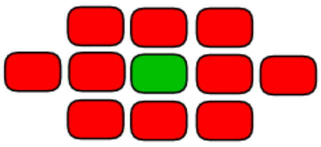


Figure 1 A single person seated with the blocked neighbouring seats.

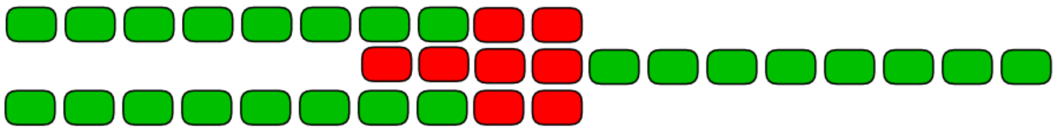


Figure 2 The potential situation if we don't place a person on the seat in the middle. We can now place 3 groups of 8 in using the otherwise blocked seats.

◀

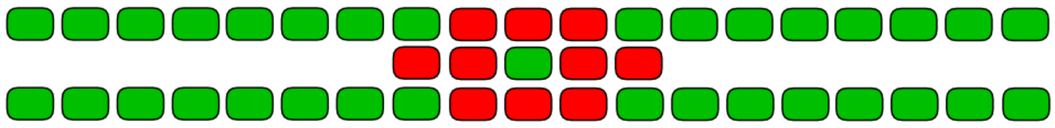


Figure 3 For a group of size $n > 2$ we can, instead of placing the group itself, place a group of $n - 2$ in the middle and 4 groups of 8 around it.

▶ **Theorem 2.** *There does not exist a c-competitive algorithm for the cinema problem with* $c < \frac{1}{25}$.

Proof. This easily follows from lemma 1. Since there is no situation in which 1 person could be replaced by more than 25 people, it cannot occur that the optimal solution OPT places more than $25 \cdot ALG(I)$ people. ◀

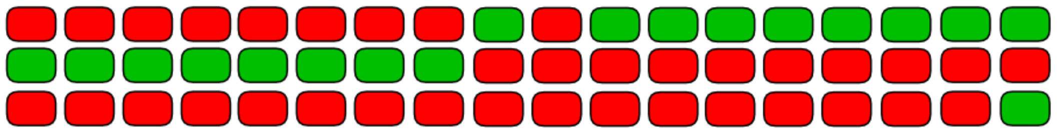


Figure 4 A cinema where FirstFit has a worst case scenario given the input 1 8 8. Seats are marked by green nodes. Empty spots are marked by red nodes.

Now let's take a look at the FirstFit algorithm. The situation as shown in lemma 1 cannot happen with FirstFit since it will never place a person on the seat in the middle. It will instead place the person in the first possible seat. This will in fact improve the competitive ratio of the algorithm.

▶ **Lemma 3.** *The lower bound for the competitive ratio of the FirstFit algorithm is* $\frac{1}{17}$*.*

Proof. If we again consider the situation in figure 1, we can analyze the situation for FirstFit. FirstFit can only block groups which come after the seated person. If it were to block a group before the seated person FirstFit would have place the person there instead. Therefore we can ignore anything before the seated person. Now if we take a look at the situation in figure 2 we can see that FirstFit blocks 2 groups of 8 instead of 3 groups of 8. It therefore places 1 person instead of the potential 17 people who could have been seated. The ratio for larger groups becomes better just like the way it did in lemma 1. This gives us a ratio of $\frac{ALG(I)}{OPT(I)} = \frac{1}{17}$. ◀

▶ **Lemma 4.** *The upper bound for the competitive ratio of the FirstFit algorithm is* $\frac{1}{17}$*.*

Proof. To proof the upperbound of the FirstFit algorithm we will give an input I for which $\frac{ALG(I)}{OPT(I)} = \frac{1}{17}$. Given the cinema in figure 4 and the input 1 8 8, we can see that FirstFit will seat a single person while optimally we could have placed all 17 people. ◀

▶ **Theorem 5.** *FirstFit is a* $\frac{1}{17}$*-competitive algorithm.*

Proof. This follows from lemma 3 and 4. ◀

4.2 Improved competitive ratio for a dense cinema

4.2.1 Worst Case Ratio

Though we can sketch a very bad input. We look at a cinema of one row, since this is representative for the cinema as a whole when there are no gaps Imagine an input I as follows:

```
1
10
1111111111
1 8
```

The optimal solutions would be:

```
xxxxxxxx11
```

The solution by FirstFit would be:

```
x111111111
```

Thus the optimal solution would be to place group with eight persons. Though FirstFit will fit the group of 1 person, after which the group of eight cannot be placed. This gives a ratio of:

$$\frac{OPT(I)}{ALG(I)} = \frac{8}{1} = 8$$

In the next two sections we will show that this ratio can in fact not get any worse.

4.2.2 For a single row

We can show that for a single row the competitive ratio can never exceed 8. The intuition for the proof is as follows. Suppose the optimal solution placed n people. If we need at least $s(n)$ seats for n people and FirstFit has used at most k seats so far, then there are $s(n) - k$ seats left for FirstFit to use. If $k \geq 8$ we know that we can place the next group no matter what size. In the following lemma's we will show that k is always greater than or equal to 8 when $\frac{OPT}{ALG} > 8$. Therefore that ratio cannot occur for any input I.

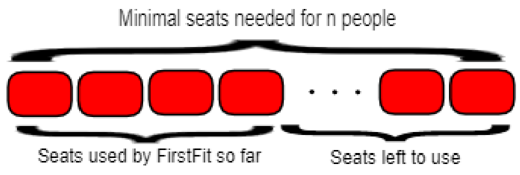

Minimal seats needed for n people
Seats used by FirstFit so far
Seats left to use

Figure 5 The intuition of the proof for a single row. If we need at least $s(n)$ seats to seat n people and FirstFit has used k seats so far it still has $s(n) - k$ seats left.

▶ **Lemma 6.** *The minimum number of seats needed on a single row for n people $s(n)$, is*

$$s(n) = n + \lfloor\frac{n-1}{8}\rfloor \cdot 2$$

Proof. We use the least amount of space when we place groups as large as possible. To place n people we fit as many groups of 8 as possible and then a group the size of the remaining seats. We then use n seats, one for each person, and we block 2 seats after every group of 8 (unless it is at the end of the row). This happens $\lfloor\frac{n-1}{8}\rfloor$ times, so the total number of seats needed is $n + \lfloor\frac{n-1}{8}\rfloor \cdot 2$ ◀

▶ **Lemma 7.** *The FirstFit algorithm uses at most $3 \cdot N$ chairs when placing N people on a single row.*

Proof. FirstFit uses the most chairs when it places N in N groups of one, because then every person will block 3 chairs. If the groups were bigger the average person would block less chairs because the 2 blocked chairs after a group will be blocked by more than 1 person. Therefore it uses at most $3 \cdot N$ chairs, by placing N groups of 1 each blocking 3 chairs in total. ◀

▶ **Theorem 8.** *The competitive ratio $\frac{OPT}{ALG}$ on a single row cannot exceed 8*

Proof. We assume $\frac{OPT}{ALG} > 8$. We know by lemma 6 that the optimal solution uses a row of length at least $s(OPT) = OPT + \lfloor\frac{OPT-1}{8}\rfloor \cdot 2$. We also know by lemma 7 that the online solution uses at most $3 \cdot ALG$ seats. By our assumption $OPT > 8 \cdot ALG$ and since this is an integer we can say that $OPT \geq 8 \cdot ALG + 1$. The row is at least $s(OPT)$ chairs long and so far we have used $3 \cdot ALG$ chairs, therefore

$$\begin{aligned}\#\text{free chairs} &\geq OPT + \lfloor \frac{OPT-1}{8} \rfloor \cdot 2 - 3 \cdot ALG \\ &\geq 8 \cdot ALG + 1 + \lfloor \frac{8 \cdot ALG + 1 - 1}{8} \rfloor \cdot 2 - 3 \cdot ALG \\ &= 5 \cdot ALG + 1 + \lfloor \frac{8 \cdot ALG}{8} \rfloor \cdot 2 \\ &= 7 \cdot ALG + 1\end{aligned}$$

We know that $ALG \geq 1$, therefore the amount of free seats is greater than or equal to 8. This, however, would suggest that FirstFit would have placed at least one more group and therefore it contradicts our assumption. The competitive ratio cannot exceed 8 on a single row. ◀

4.2.3 For multiple rows

We will now extend the proof to show that the maximum ration of $\frac{8}{1}$ will also hold for multiple rows. Suppose we have a cinema with rows of width m.

▶ **Lemma 9.** *The minimum number of seats needed on a single row, with gaps of size α inbetween groups, for n people $s_\alpha(n)$, is*

$$s_\alpha(n) = n + \lfloor \frac{n-1}{8} \rfloor \cdot \alpha$$

Proof. We generalize the proof of lemma 6. The only difference is that we block α seats after each group. ◀

▶ **Lemma 10.** *We need $s_9(n)$ or more seats in a cinema with multiple rows to seat n people.*

Proof. We construct an optimal solution as follows (see figure 6 for an example):

- On the first row place groups of maximum size until the row is full. Keep a gap of 10 between each group.
- On the second row skip the first 9 seats, then place a group of maximum size until the row is full. Keep a gap of 9 between each group.
- Repeat these steps for 2 rows at a time until we have placed n people.

We now transform the cinema into a single row by appending row_{i+1} after row_i. We now have a single row with gaps of 9 or 10 between each group. Suppose there are β gaps of 10. The number of seats needed to seat all n people is then $s_9(n) + \beta$. There can be 0 or more gaps of 10 in this construction so the amount of seats needed for n people is bigger than or equal to $s_9(n)$. ◀

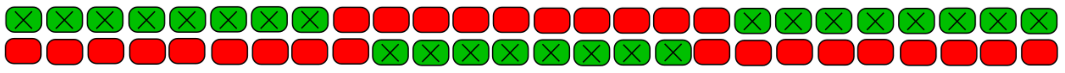


Figure 6 An example setup for $n = 24$ with row width $m = 26$. If n was bigger this pattern is repeated until n is reached

▶ **Lemma 11.** *In the worst case the FirstFit algorithm uses less than $6 \cdot N$ seats for N people.*

Proof. The worst case happens when we only have groups of 1. The first person seated (in the corner) uses his own seat and blocks the two seats to the right and the one directly behind him and the one diagonally behind him, using a total of 5 seats. Every person after that, until the last person on the row, blocks 5 seats. 2 seats to the right (to the left has already been blocked by the previous person) and 3 seats behind him (1 directly, 2 diagonally), using a total of 6 seats. See figure 7 for an example. We have N groups of one, which all use 6 seats or less, the amount of used seats is therefore smaller than $6 \cdot N$. ◀

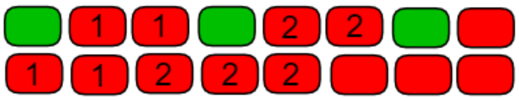


Figure 7 An example FirstFit setup with row width $m = 8$ and $n = 3$. The seats blocked by the first person are marked 1 and the seats blocked by the second person are marked 2. The first person uses 5 seats in total and the second person uses 6 seats in total. If n was bigger this pattern would be repeated until n is reached.

▶ **Theorem 12.** *The ratio $\frac{OPT}{ALG}$ on multiple rows cannot exceed 8.*

Proof. We assume $\frac{OPT}{ALG} > 8$. We know by lemma 10 that the optimal solution uses at least $s_\alpha(OPT)$ seats. We also know by lemma 11 that the online solution uses at most $6 \cdot ALG$ seats. By our assumption $OPT > 8 \cdot ALG$ and, again, since this is an integer we can say that $OPT \geq 8 \cdot ALG + 1$. There are at least $s_9(OPT)$ seats and so far we have used less than $6 \cdot ALG$ seats, therefore

$$\begin{aligned}
\#\text{free chairs} &\geq s_9(OPT) - 6 \cdot ALG \\
&= OPT + \lfloor \frac{OPT - 1}{8} \rfloor \cdot 9 - 6 \cdot ALG \\
&\geq 8 \cdot ALG + 1 + \lfloor \frac{8 \cdot ALG + 1 - 1}{8} \rfloor \cdot 9 - 6 \cdot ALG \\
&= 2 \cdot ALG + 1 + \lfloor \frac{8 \cdot ALG}{8} \rfloor \cdot 9 \\
&= 11 \cdot ALG + 1
\end{aligned}$$

We know that $ALG \geq 1$, therefore the amount of free seats is greater than 8. This, however, would suggest that FirstFit would have placed at least one more group and therefore it contradicts our assumption. The ratio $\frac{OPT}{ALG}$ cannot exceed 8 on a multiple rows. ◀

From theorem 12 and the fact that there is a known input I where $\frac{OPT(I)}{ALG(I)} = 8$, it easily follows that FirstFit is a $\frac{1}{8}$-competitive algorithm in a cinema where every place has an available seat.

5 Results

5.1 Offline problem

In this section, we will compare the results of the approaches for the offline problem. The notation utilized in this comparison part is the following:

NPS number of people sitting in the cinema

Runtime time passed between the start of the actual algorithm and the output of the solution. The runtime is given in seconds

GFO gap from the optimal solution. When the GFO is equal to 0.0%, the solution given is optimal, when the GFO is greater than 0.0%, the gap is calculated between the best solution found till that moment and the latest upper bound found in the solving process

ACS actual cinema saturation, considering the number of people in the solution

The specifics of the hardware components utilized for each approach are the following:

ILP Model:
- MacBook Pro (13-inch, 2017, Four Thunderbolt 3 Ports), 250 GB
- Operating system: macOS Catalina, version 10.15.3
- Processor: 3.1 GHz Intel Core i5 dual-core
- Memory: 8 GB 2133 MHz LPDDR3

FF/LBF algorithms:
- Acer Aspire ES1-572, 250 GB
- Operating system: Microsoft Windows 10 Home, version 10.0.18362 Build 18362
- Processor: Intel i5-7200U CPU @ 2.50GHz
- Memory: 6 GB DDR4 PC4-17000 2133MHz

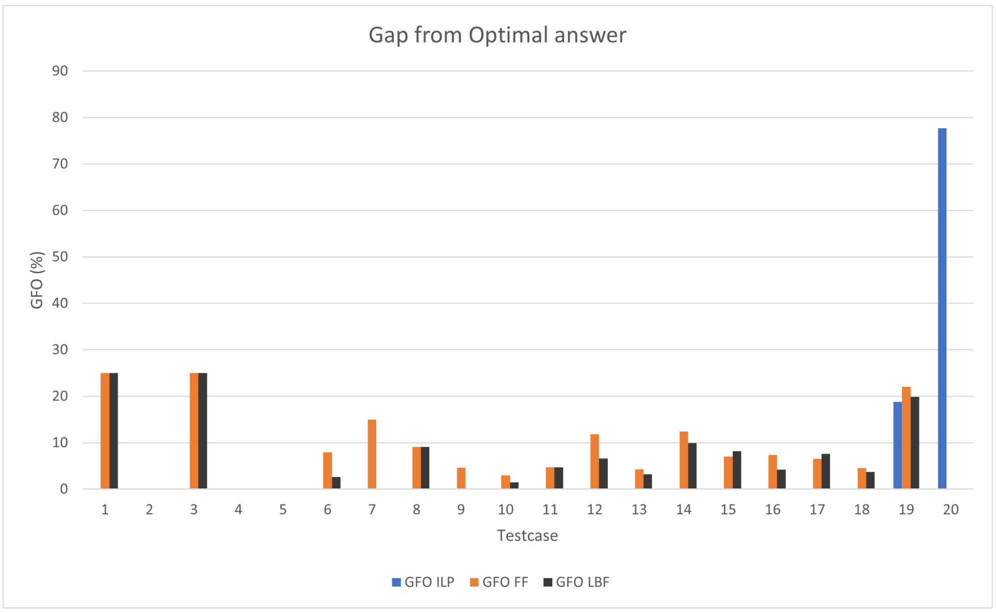


Figure 8 The gap from the optimal answer for the offline approaches.
The GFO for the testcases 21-23 has been omitted due to the fact the ILP solver did not find a first solution. So there was no upperbound available to us to base the GFO on. It can be seen that the ILP solver finds an optimal answer for most testcases and that LBF outperforms FF in most cases.

The results of every test case can be found in Appendix A.

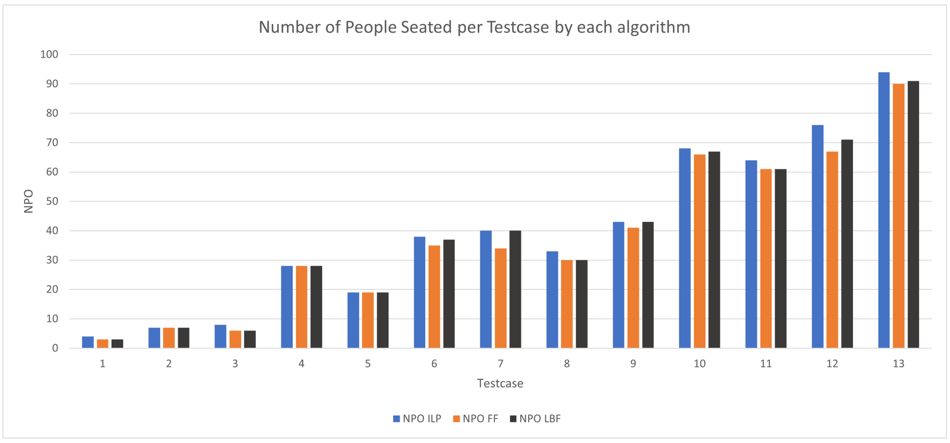


Figure 9 The number of people seated in each small testcase for each approach.
This figure shows the performance of the three offline approaches in the smaller testcases. The ILP solver seems to outperform the other approaches. However the runtime for each algorithm is not considered here.

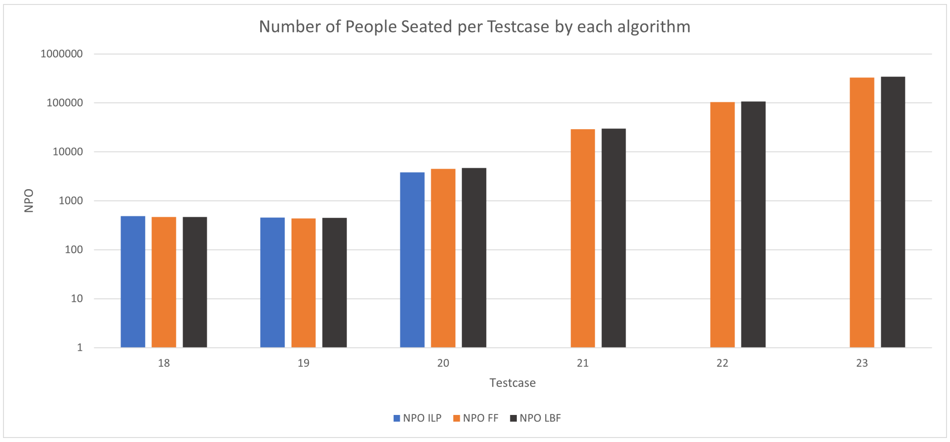


Figure 10 The number of people seated in each big testcase for each approach.
The two figure above were split to keep the difference in performance for each algorithm visible. As can be seen the FF and LBF algorithms start to outperform the ILP solver for these bigger cases.

5.2 Online Problem

5.2.1 Number of seated people

When we look at Table 1, we see that FF does not perform so poor. A big advantage is that it can also compute cases from 16 on, whilst OPT (computed with the offline ILP) cannot. In figure 11 we see OPT in blue, FF in orange and FirstFit, which looks at the future (that is, FF algorithm from the offline version) in gray. As can be seen the online FirstFit algorithm performs very well, although it cannot look into the future.

The percentage is constantly above 70% for our test cases. This constancy is visualised Figure 1, where the ratio OPT/ALG is visualized.

	OPT	FF	FF-Future	FF / OPT	FF-Runtime (s)
1	6	5	5	83.3%	165
2	9	9	9	100%	172
3	17	15	15	88.2%	306
4	17	12	17	70.6%	524
5	27	22	25	81.4%	473
6	38	29	34	76.3%	451
7	45	41	41	91.1%	616
8	40	38	39	95.0%	650
9	79	69	74	87.3%	1067
10	94	74	90	78.7%	1202
11	52	45	52	86.5%	1338
12	164	144	157	87.8%	2024
13	262	234	245	89.3%	3842
14	456	330	438	72.4%	5811
15	488	414	466	84.8%	6465
16		9629	13,022		202,310
17		143,089			1,966,486
18		331,483			8,952,997

Table 1 The amount of seated persons per test case OnlineA 1-18 for the algorithms OPT, FF and FF-Future

In the table, we see that the runtime behaves linear. The runtime is very constant per seated person linear as is visualized from figure 14. The specifics of the hardware components utilized for running FF for the online problem are:

- Lenovo ThinkPad P51 20HJM
- Operating system: Windows 10
- Processor: 3.8 GHz Intel Core i7-7700HQ
- Memory: 8 GB 2400 MHz DDR4

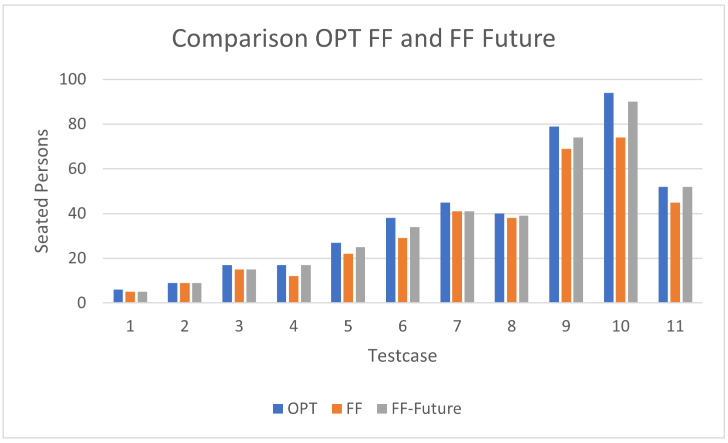


Figure 11 The amount of seated persons per test case Online A1-11 for the algorithms OPT, FF and FF-Future

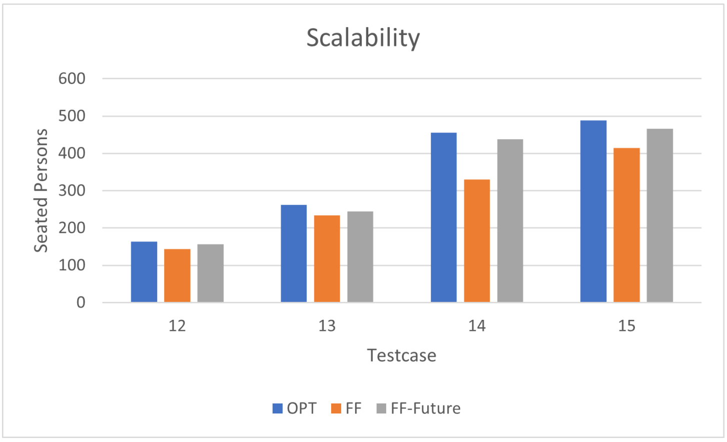


Figure 12 The amount of seated persons per test case Online A12-15 for the algorithms OPT, FF and FF-Future

As can be seen in Figure 12, the scalability is not so bad. It does not perform worse than 71%, as we can see in Table 1.

5.2.2 Number of seated people

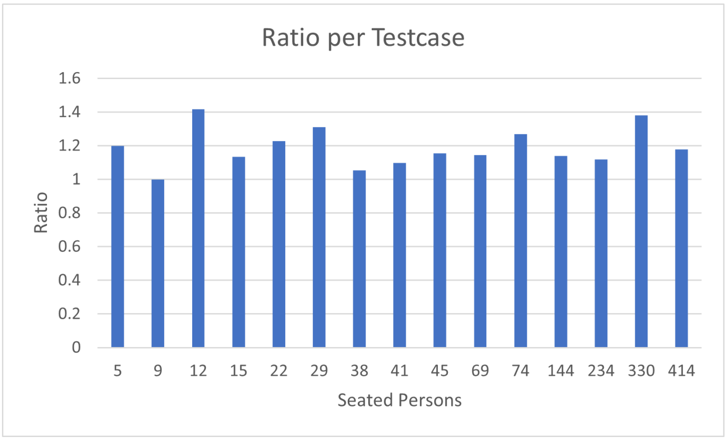


Figure 13 The ratio per test case

In Figure 14 we see that the runtime is quite equal per seated person.

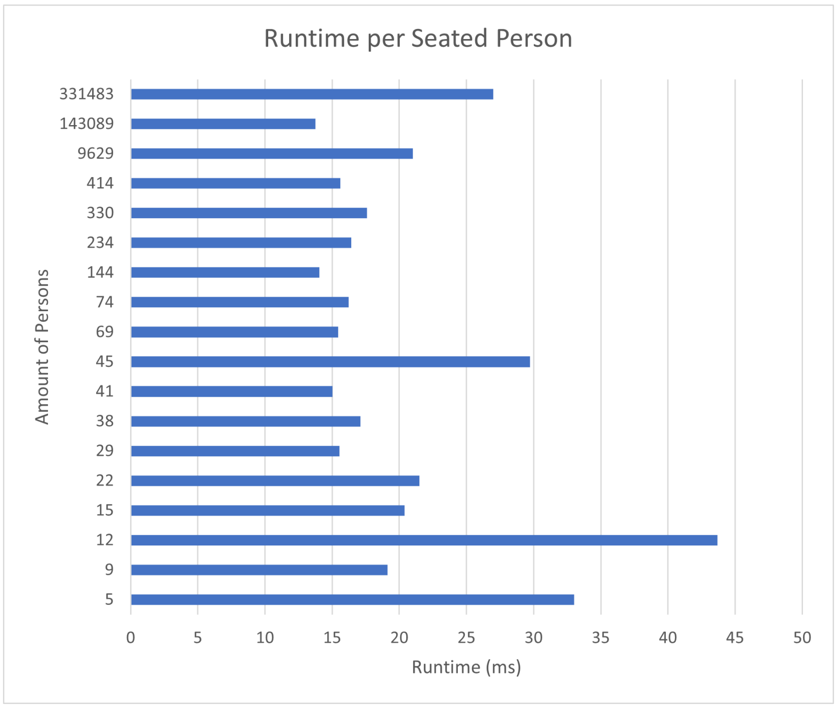


Figure 14 The amount of seated persons per test case Online A1-15 sorted on number of seated persons

6 Conclusions

6.1 Offline problem

As can be seen from the results, we were able to find an optimal solution with up to 1020 seats available, although, in that layout, there were many 0s missing. In fact, it can be noted that there is a dependence between runtime and the layout of the corridors in the cinema: the more “ordered” the cinema is (meaning that there are many 1’s all together), the more time it takes to compute the solution, as there are combinatorially more possibilities to choose from, making the calculation more difficult. This statement is true only for the ILP, while the other two algorithms doesn’t care about that.

Finally, to sum up the strategy to adopt for the offline problem, it should be firstly used the ILP model in order to see if an optimal (or nearly optimal) solution could be computed for a specific input, if not, because the input size gets too large, the FirstFit based algorithm and the LeastBlockFit algorithm should provide a very good solution even for n and m both equal to 1000 and, the difference between those last two algorithms basically involve a trade-off between find a nearly optimal solution and the runtime needed in order to compute that, that is, a trade-off between the velocity and the quality of the solution, especially for large inputs.

6.2 Online problem

For FirstFit we proved a very strong $\frac{1}{8}$ competitive ratio for the case that the cinema has no gaps. In the scenario that there are gaps, we have seen that we still can prove a competitive ratio of $\frac{1}{17}$, which is only about a fraction of two off with the scenario without gaps. We also proved a competitive ratio of $\frac{1}{25}$ for the worst case scenario in the most general case. This case, however, does not apply to FirstFit.

In practice the FirstFit performs way better than worst case competitive ratios. For all our test cases it fitted at least 70% of the actual optimum, whilst the optimum can look into the future. In practice cases FirstFit can give a good solution. FirstFit is also a very scalable option and linearly computable, which makes it perfect for cases where you cannot look into the future and that cannot be computed using integer linear programming.

References

1 Béla Bollobás. *Modern Graph Theory*. Graduate Texts in Mathematics 184. Springer-Verlag New York, 1 edition, 1998.

2 Konstantinos Georgiou, Evangelos Kranakis, and Danny Krizanc. Random maximal independent sets and the unfriendly theater seating arrangement problem. *Discrete Mathematics*, 309(16):5120 – 5129, 2009. URL: `http://www.sciencedirect.com/science/article/pii/S0012365X09001721`, `doi:https://doi.org/10.1016/j.disc.2009.03.049`.

3 Robert Endre Tarjan and Anthony E Trojanowski. Finding a maximum independent set. *SIAM Journal on Computing*, 6(3):537–546, 1977.

A Test cases for offline algorithms

In these results, we refer with (clickable) monospace text to files that can be found relative to `http://webspace.science.uu.nl/~3898733/ads-2020/`. So, for example, `input/1.txt` refers to `http://webspace.science.uu.nl/~3898733/ads-2020/input/1.txt`.

A.1 Test 1

Input: `input/1.txt`

Total number of people in input: 6
Number of seats available: 8
Theoretical cinema saturation (with no 1.5m rule): 75.0%

Output:

	ILP	FirstFit	LeastBlockFit
NPS	4	3	3
Runtime	0.00171	0.007	0.001
GFO	0.0%	25.0%	25.0%
ACS	50.0%	37.5%	37.5%
Layout	`ilp/1.txt`	`firstfit/1.txt`	`leastblockfit/1.txt`

A.2 Test 2

Input: `input/2.txt`

Total number of people in input: 30
Number of seats available: 16
Theoretical cinema saturation (with no 1.5m rule): 187.5%

Output:

	ILP	FirstFit	LeastBlockFit
NPS	7	7	7
Runtime	0.0077	0.000	0.000
GFO	0.0%	0.0%	0.0%
ACS	43.75%	43.75%	43.75%
Layout	`ilp/2.txt`	`firstfit/2.txt`	`leastblockfit/2.txt`

A.3 Test 3

Input: `input/3.txt`

Total number of people in input: 14
Number of seats available: 22
Theoretical cinema saturation (with no 1.5m rule): 63.639%

Output:

	ILP	FirstFit	LeastBlockFit
NPS	8	6	6
Runtime	0.00235	0.000	0.000
GFO	0.0%	25.0%	25.0%
ACS	36.36%	27.28%	27.28%
Layout	`ilp/3.txt`	`firstfit/3.txt`	`leastblockfit/3.txt`

A.4 Test 4

Input: `input/4.txt`

Total number of people in input: 28
Number of seats available: 56
Theoretical cinema saturation (with no 1.5m rule): 50.0%

Output:

	ILP	FirstFit	LeastBlockFit
NPS	28	28	28
Runtime	0.0123	0.000	0.000
GFO	0.0%	0.0%	0.0%
ACS	50.0%	50.0%	50.0%
Layout	`ilp/4.txt`	`firstfit/4.txt`	`leastblockfit/4.txt`

A.5 Test 5

Input: `input/5.txt`

Total number of people in input: 19
Number of seats available: 49
Theoretical cinema saturation (with no 1.5m rule): 38.78%

Output:

	ILP	FirstFit	LeastBlockFit
NPS	19	19	19
Runtime	0.0334	0.000	0.000
GFO	0.0%	0.0%	0.0%
ACS	38.78%	38.78%	38.78%
Layout	`ilp/5.txt`	`firstfit/5.txt`	`leastblockfit/5.txt`

A.6 Test 6

Input: `input/6.txt`

Total number of people in input: 64
Number of seats available: 80
Theoretical cinema saturation (with no 1.5m rule): 80.0%

Output:

	ILP	FirstFit	LeastBlockFit
NPS	38	35	37
Runtime	0.21397	0.000	0.000
GFO	0.0%	7.89%	2.63%
ACS	47.5%	43.75%	46.25%
Layout	`ilp/6.txt`	`firstfit/6.txt`	`leastblockfit/6.txt`

A.7 Test 7

Input: `input/7.txt`

Total number of people in input: 56
Number of seats available: 84
Theoretical cinema saturation (with no 1.5m rule): 66.67%

Output:

	ILP	FirstFit	LeastBlockFit
NPS	40	34	40
Runtime	0.07187	0.000	0.000
GFO	0.0%	15.0%	0.0%
ACS	47.62%	40.48%	47.62%
Layout	`ilp/7.txt`	`firstfit/7.txt`	`leastblockfit/7.txt`

A.8 Test 8

Input: `input/8.txt`

Total number of people in input: 35
Number of seats available: 69
Theoretical cinema saturation (with no 1.5m rule): 50.72%

Output:

	ILP	FirstFit	LeastBlockFit
NPS	33	30	30
Runtime	0.25998	0.000	0.000
GFO	0.0%	9.09%	9.09%
ACS	47.83%	43.48%	43.48%
Layout	`ilp/8.txt`	`firstfit/8.txt`	`leastblockfit/8.txt`

A.9 Test 9

Input: `input/9.txt`

Total number of people in input: 43
Number of seats available: 73
Theoretical cinema saturation (with no 1.5m rule): 58.9%

Output:

	ILP	FirstFit	LeastBlockFit
NPS	43	41	43
Runtime	0.02281	0.000	0.000
GFO	0.0%	4.65%	0.0%
ACS	58.9%	56.16%	58.9%
Layout	`ilp/9.txt`	`firstfit/9.txt`	`leastblockfit/9.txt`

A.10 Test 10

Input: `input/10.txt`

Total number of people in input: 136
Number of seats available: 131
Theoretical cinema saturation (with no 1.5m rule): 103.82%

Output:

	ILP	FirstFit	LeastBlockFit
NPS	68	66	67
Runtime	0.31743	0.000	0.000
GFO	0.0%	2.94%	1.47%
ACS	51.91%	50.38%	51.14%
Layout	`ilp/10.txt`	`firstfit/10.txt`	`leastblockfit/10.txt`

A.11 Test 11

Input: `input/11.txt`

Total number of people in input: 78
Number of seats available: 168
Theoretical cinema saturation (with no 1.5m rule): 46.43%

Output:

	ILP	FirstFit	LeastBlockFit
NPS	64	61	61
Runtime	2.10301	0.000	0.000
GFO	0.0%	4.69%	4.69%
ACS	38.1%	36.3%	36.3%
Layout	`ilp/11.txt`	`firstfit/11.txt`	`leastblockfit/11.txt`

A.12 Test 12

Input: `input/12.txt`

Total number of people in input: 330
Number of seats available: 163
Theoretical cinema saturation (with no 1.5m rule): 202.45%

Output:

	ILP	**FirstFit**	**LeastBlockFit**
NPS	76	67	71
Runtime	1.05081	0.000	0.000
GFO	0.0%	11.84%	6.58%
ACS	46.63%	41.1%	43.56%
Layout	`ilp/12.txt`	`firstfit/12.txt`	`leastblockfit/12.txt`

A.13 Test 13

Input: `input/13.txt`

Total number of people in input: 168
Number of seats available: 200
Theoretical cinema saturation (with no 1.5m rule): 84.0%

Output:

	ILP	**FirstFit**	**LeastBlockFit**
NPS	94	90	91
Runtime	71.2664	0.000	0.000
GFO	0.0%	4.26%	3.19%
ACS	47.0%	45.0%	45.5%
Layout	`ilp/13.txt`	`firstfit/13.txt`	`leastblockfit/13.txt`

A.14 Test 14

Input: `input/14.txt`

Total number of people in input: 128
Number of seats available: 252
Theoretical cinema saturation (with no 1.5m rule): 50.79%

Output:

	ILP	**FirstFit**	**LeastBlockFit**
NPS	121	106	109
Runtime	402.52202	0.000	0.000
GFO	0.0%	12.4%	9.92%
ACS	48.02%	42.06%	43.26%
Layout	`ilp/14.txt`	`firstfit/14.txt`	`leastblockfit/14.txt`

A.15 Test 15

Input: `input/15.txt`

Total number of people in input: 114
Number of seats available: 191
Theoretical cinema saturation (with no 1.5m rule): 59.69%

Output:

	ILP	FirstFit	LeastBlockFit
NPS	86	80	79
Runtime	5.45584	0.000	0.000
GFO	0.0%	6.98%	8.14%
ACS	45.03%	41.88%	41.36%
Layout	`ilp/15.txt`	`firstfit/15.txt`	`leastblockfit/15.txt`

A.16 Test 16

Input: `input/16.txt`

Total number of people in input: 201
Number of seats available: 405
Theoretical cinema saturation (with no 1.5m rule): 49.63%

Output:

	ILP	FirstFit	LeastBlockFit
NPS	191	177	183
Runtime	1673.21206	0.000	0.001
GFO	0.0%	7.33%	4.19%
ACS	47.16%	43.7%	45.19%
Layout	`ilp/16.txt`	`firstfit/16.txt`	`leastblockfit/16.txt`

A.17 Test 17

Input: `input/17.txt`

Total number of people in input: 552
Number of seats available: 586
Theoretical cinema saturation (with no 1.5m rule): 94.19%

Output:

	ILP	FirstFit	LeastBlockFit
NPS	262	245	242
Runtime	15.64719	0.000	0.001
GFO	0.0%	6.49%	7.63%
ACS	44.71%	41.8%	41.3%
Layout	`ilp/17.txt`	`firstfit/17.txt`	`leastblockfit/17.txt`

A.18 Test 18

Input: `input/18.txt`

Total number of people in input: 765
Number of seats available: 1020
Theoretical cinema saturation (with no 1.5m rule): 74.119%

Output:

	ILP	**FirstFit**	**LeastBlockFit**
NPS	488	466	470
Runtime	16541.85147	0.009	0.005
GFO	0.0%	4.5%	3.7%
ACS	47.839%	45.69%	46.08%
Layout	`ilp/18.txt`	`firstfit/18.txt`	`leastblockfit/18.txt`

A.19 Test 19

Input: `input/19.txt`

Total number of people in input: 1000
Number of seats available: 1065
Theoretical cinema saturation (with no 1.5m rule): 93.89%

Output:

	ILP	**FirstFit**	**LeastBlockFit**
NPS	456	438	450
Runtime	515.23510	0.002	0.010
GFO	18.8%	22.0%	19.87%
ACS	42.82%	41.13%	42.25%
Layout	`ilp/19.txt`	`firstfit/19.txt`	`leastblockfit/19.txt`

► Note. In the ILP Model, after 39112.9729 seconds the GFO is equal to 12.5% but always 456 people are seated. Then, the GFO of the FirstFit based algorithm should be 15.95% and the GFO of the LeastBlockFit algorithm should be 13.65% in this case.

A.20 Test 20

Input: `input/20.txt`

Total number of people in input: 5587
Number of seats available: 12289
Theoretical cinema saturation (with no 1.5m rule): 45.46%

Output:

	ILP	**FirstFit**	**LeastBlockFit**
NPS	3816	4523	4681
Runtime	674.2301	0.144	0.375
GFO	77.664%	–	–
ACS	31.05%	36.8%	38.09%
Layout	`ilp/20.txt`	`firstfit/20.txt`	`leastblockfit/20.txt`

From now on, the ILP solver is becoming useless for this size of input since it takes too much time even for compute a first solution, thus, the results are reported only for the FirstFit based algorithm and for the LeastBlockFit algorithm.

A.21 Test 21

Input: `input/21.txt`

Total number of people in input: 34737
Number of seats available: 78991
Theoretical cinema saturation (with no 1.5m rule): 43.98%

Output:

	ILP	FirstFit	LeastBlockFit
NPS	–	28863	29871
Runtime	–	5.644	16.349
GFO	–	–	–
ACS	–	36.54%	37.82%
Layout	–	`firstfit/21.txt`	`leastblockfit/21.txt`

A.22 Test 22

Input: `input/22.txt`

Total number of people in input: 124786
Number of seats available: 280662
Theoretical cinema saturation (with no 1.5m rule): 44.46%

Output:

	ILP	FirstFit	LeastBlockFit
NPS	–	104521	107491
Runtime	–	68.229	202.356
GFO	–	–	–
ACS	–	37.24%	38.3%
Layout	–	`firstfit/22.txt`	`leastblockfit/22.txt`

A.23 Test 23

Input: `input/23.txt`

Total number of people in input: 442956
Number of seats available: 868537
Theoretical cinema saturation (with no 1.5m rule): 51.0%

Output:

	ILP	FirstFit	LeastBlockFit
NPS	–	330348	343494
Runtime	–	668.021	2593.646
GFO	–	–	–
ACS	–	38.03%	39.55%
Layout	–	`firstfit/23.txt`	`leastblockfit/23.txt`