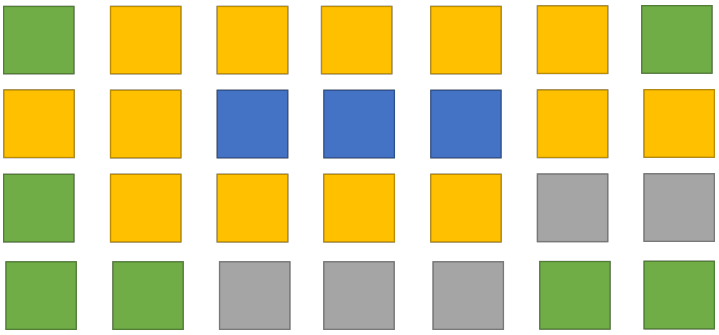
Solutions to the Distance Constrained Cinema Seating Problem

Master thesis

Submitted by:	Stoll, Maximilian
Course:	Master of Computer Science
Student number:	6352421
Supervisor:	Prof. Dr. Hans Bodlaender



Utrecht University

Heidelberglaan 8, 3584 CS Utrecht

Abstract

As a consequence of COVID-19 regulation, cinemas in the Netherlands are not allowed to seat groups of different households within 1.5 meters of each other. This regulation inspired an optimization problem called the Distance Constrained Cinema Seating (DCCS) problem. The first input to this problem is a cinema layout of fixed width and height that consists of seats and spaces, where a space represents a location without a seat. E.g., corridors can create spaces. The second input is a list of groups, where each group consists of one to eight people. The goal is to seat as many people as possible according to the cinema layout while ensuring a fixed number of empty seats or spaces between each group.

The approach described in this paper is to construct a cinema graph from a given cinema layout, where the seats are modelled as vertices and the distance constraints between the seats are modelled as edges. An Independent Set (IS) on a cinema graph could then represent the possible starting locations from where to seat groups from. Ideally this Independent Set (IS) should be of maximum cardinality. To find a Maximum Independent Set (MIS) on a cinema graph, three algorithms are presented in this paper: A polynomial approximation algorithm, an Integer Linear Program (ILP) algorithm and a Fixed Parameter Tractable (FTP) algorithm.

Then three algorithms are presented to seat the given groups using a cinema graph: Two polynomial approximation algorithms and an ILP algorithm. Furthermore a heuristic is given for the ILP algorithm to improve its run time.

Lastly, several experiments were conducted to evaluate the efficiency of these algorithms. The experiment data is visualized, and several conclusions are presented in this paper. The main conclusion is that for small or sparse cinemas or if solving time is not a factor, the ILP algorithm is the ideal choice as it can seat the most people and as a result has a lower percentage of revenue loss than all other algorithms. For large or very dense cinemas or when solving time is important the approximation algorithms continue to remain the only choice.

Contents

List of Figures

List of Tables

List of Abbreviations

DCCS Distance Constrained Cinema Seating

DP Dynamic Programming

ETR Existential Theory of the Reals

FPTAS Fully Polynomial Time Approximation Scheme

FTP Fixed Parameter Tractable

ILP Integer Linear Program

IS Independent Set

MIS Maximum Independent Set

PCS Percentage of Capacity Seated

PPS Percentage of People Seated

UDG Unit Disk Graph

1 Introduction

In late 2019, a virus was discovered that would change the world: COVID-19. As a result of the pandemic, many regulations were put in place to prevent further spread of the virus in the Netherlands. One of the regulations was requiring that people who are not in the same household are to always keep 1.5 meters distance at all times. For many indoor events, this rule can be logistically challenging, including cinemas and theatres.

The idea of seating groups of various sizes in a confined seating space such as a cinema, while ensuring sufficient empty seats between each group, was inspiration for the optimization problem called the Distance Constrained Cinema Seating (DCCS) problem. The first input to this problem is a cinema layout of fixed width and height consisting of seats and spaces, where a space represents a location without a seat. E.g., corridors can create spaces. The second input is a list of groups, where each group consists of one to eight people. The goal is to seat as many people as possible according to the cinema layout, while ensuring the following distance constraints:

1. All persons in the same group should sit next to each other in the same row, without interruptions by empty seats or spaces.
2. There must be at least two empty seats or spaces between two groups seated in the same row.
3. There must be at least one empty seat or space between two groups seated in the same column.
4. Two groups cannot sit directly diagonally from each other.
5. No person can be seated on a space.

For clarity these distance constraints have been visualized in Figure 1. In this figure, the blue squares are seats on which a group of three people are seated and the yellow squares around this group are the invalidated seats as a result of the above-mentioned distance constraints. The green squares are empty seats where other groups can be seated and the grey squares are spaces where no person can be seated.

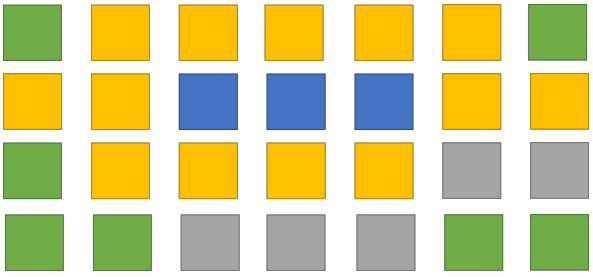


Figure 1: Example of a cinema layout with width 7 and height 4.

Given a layout of seats in a cinema such as the one shown in Figure 1, a graph seems like a natural data abstraction where the seats are modelled as vertices and the distance constraints between the seats are modelled as edges. This type of graph will from now on be referred to as a cinema graph. An Independent Set (IS) on this cinema graph could represent the possible starting locations to seat groups from. Ideally this IS should be of maximum cardinality. But considering that the MIS problem is NP-hard, implementing an optimal algorithm that runs in polynomial time is not possible, unless $P = NP$. A solution could be to adapt an algorithm used on Unit Disk Graphs (UDG). Given a set of n equally sized circles on a plane, a UDG is an intersection graph with n vertices where each vertex corresponds to a circle. An edge is added for each vertex pair if the corresponding circles intersect on the plane. Algorithms can exploit the underlying geometric constraints of UDG to exhibit different behaviour for the MIS problem. Similar to a UDG, the underlying geometric structure of a cinema could be exploited to implement a more efficient algorithm for the MIS problem.

Using the concepts of cinema graphs and IS, the following contributions are given as part of this research:

1. Present and implement a polynomial time algorithm for constructing a cinema graph from a cinema layout (Section 4).
2. Present and implement two algorithms to the MIS problem on a cinema graph: A deterministic polynomial time 4-approximation algorithm and a non-deterministic Integer Linear Program (ILP) algorithm. Furthermore, a third theoretical approach for a deterministic Fixed Parameter Tractable (FTP) algorithm is also presented. (Section 5).
3. Present and implement three algorithms for solving the DCCS problem using cinema graphs: Two deterministic polynomial time approximation algorithms and a non-deterministic ILP algorithm. A heuristic for improving the run time of the ILP algorithm is also presented. (Section 6).
4. Conduct experiments on the above-mentioned algorithms. (Section 7).
5. Present conclusions based on the results of the experiments. (Section 8).

2 Related Research

2.1 Distance Constrained Seating

In this section, papers are presented that are related to seating problems in general and were written during the pandemic as a direct consequence of the distance regulations imposed on confined locations.

Blom et al. [1] define and implement an ILP to maximize the number of guests in the Music Building Eindhoven, while respecting the 1.5m distancing rules for members of different households. Their ILP model is based on the connection they described between safe seating arrangements and the problem of packing a maximum number of trapezoids in a polygon shape. They extend their model to also support consecutive shows, where the same show is performed twice in an evening with the constraint that no seat can be used twice for both shows. The model was tested on two rooms: Grand (1250 seats) and Small (400 seats), with varying household sizes ranging from one to four. Based on the computational results, the authors concluded that the distancing constraint has a huge impact on the occupancy of a theatre, with a maximum occupancy of up to 40% for one show and 70% for consecutive shows, regardless of room size.

Very recently, Langedijk et al. [2] introduce two solutions to the DCCS problem: The first solution is a polynomial time algorithm, called SEAT_GREEDY_MADS that attempts to seat each group (largest groups first) at a starting coordinate where the total number of invalidated seats would be minimized. The second algorithm is called SEAT_ILP_MADS that models the cinema problem as an ILP to optimally seat groups in a cinema. The model is based on a concrete approach where the cinema is mapped to a 2D coordinate plane and each seat and empty space is represented using an x and y coordinate. For each combination of coordinate and group size, a binary decision variable is added to the model to denote whether this group is seated at coordinate (x, y). The following constraints are added to the model:

1. No group can be seated at two different starting coordinates.
2. No group can be seated within each other's distance constraints.
3. No group can be seated outside of the bounds of the 2D plane.

Their experiments showed that on small instances the SEAT_ILP_MADS seats more people than its greedy counterpart. However, for larger instances SEAT_ILP_MADS is unable to solve instances within reasonable time.

There also exists research on distance seating individual people in confined locations such as classrooms. To do this Greenberg et al. [3] represent the classroom as a graph where

they model seats as vertices and insert edges between seats that are closer than a given threshold. A MIS on this graph then corresponds to the optimal seating plan. Their tool was used at Cornell University, where it is used to automate the classroom planning process during the pandemic. The results indicated that their tool could run on hundreds of classrooms and helped identify over 400 additional seats. Further research on the classroom seating problem includes Bortolete et al. [4] using circle packing techniques and Murray [5] where he models this problem as an ILP. Alternatively, the individual seating problem also occurs in other locations such as: Workspaces [6], planes [7] and stadiums [8, 9].

Lastly, an interesting problem introduced in the pre-pandemic era is known as the Unfriendly Theatre Seating Arrangement problem. This problem was originally defined by Freedman and Shepp [10] in 1962. In this problem people arrive one at a time to a theatre and they choose seats at random so that no one is in front of them, behind them or on either side. The goal is to determine the number of people seated in the theatre when it becomes full. In their paper Gergiou et al. [11] equate this problem to generating a MIS on a $m \cdot n$ grid graph and randomly removing nodes and its neighbours until there are no more nodes left. They then proceed to prove the asymptotic limit for theatre sizes: $2 \cdot n$.

2.2 Unit Disk Graphs

Unit Disk Graphs are a class of intersection graphs of close unit-diameter disks on a plane that can exploit their geometric constraints to exhibit different behaviour on known graph problems. Some problems continue to remain NP-complete for this class, other problems are polynomial time solvable, and some can be efficiently approximated. In 1990 Clark et al. [12] surveyed the landscape of known graph problems on a UDG, the result of their research is shown in Table 1.

Graph Problem	**UDG**
Chromatic Number	NP-complete
Clique	Polynomial
Independent Set	NP-complete
Vertex Cover	NP-Complete
Dominating Set	NP-complete
Connected Dominating Set	NP-complete
Hamiltonian Circuit	NP-complete
Steiner Tree	NP-complete

Table 1: Complexity classes of known graph problems on a UDG [12, p. 167].

In the same paper, the authors proved that the MIS problem continues to remain NP-

complete for UDG graphs by polynomial time reduction from the Planar Vertex Cover problem. However, despite this intractability, there exists literature defining efficient constant approximation algorithms and approximation schemes. An overview of the constant approximation algorithms for the MIS problem is shown in Table 2.

Literature	Year	Factor	Time	Space
Marethe et al. [13]	1995	3	$O(n^2)$	$O(n)$
Das et al. [14]	2015	2	$O(n^3)$	$O(n^2)$
Jallu and Das [15]	2016	2	$O(n^2)$	$O(n \log n)$
Nandy et al. [16]	2017	2	$O(n^2)$	$O(n^2)$
Das et al. [17]	2020	2.16	$O(n \log^2 n)$	$O(n^2)$

Table 2: The evolution of MIS approximation algorithms on a UDG

As for the approximation schemes, Nieberg and Hurink [18] describe a robust Polynomial Time Approximation Scheme (PTAS) for both the weighted and unweighted version of the MIS problem and is the first PTAS in this research area that does not require a disk representation.

If the disk representation is given, many approximation schemes use the *shifting technique* to approximate the MIS on a UDG [19, 20, 21]. This technique was independently discovered by Hochbaum and Maass [22] as a solution in image processing and later used by Baker [23] on planar graphs. The general idea of this technique is to segregate the plane into smaller sub-problems using an equally spaced separator and solve each sub-problem individually. In the end, the sub-solutions are combined to form a solution to the global problem. Then the separator is *shifted* to a new placement and the process repeats. The best solution over all placements is selected as the closest approximation to the optimum. Van Leeuwen [24] uses this technique to present an asymptotic Fully Polynomial Time Approximation Scheme (FPTAS) which is bounded by the density of a UDG, where the density describes the maximum number of disk centres in a 1 by 1 grid square. In the same paper, he also introduces a FTP algorithm that is polynomial bounded on the thickness of a UDG, where the thickness is determined by segregating the disk representation into equally spaced slabs of width 1 and counting the maximum number of disk centres that fit in a slab.

2.2.1 UDG Recognition and Construction

In addition to the graph problems there exist two specific problems for a UDG. The first is *UDG Recognition*: Given a graph determine if it has a realization, which is a mapping of vertices to 2D-coordinates on a plane. This problem is also known as the k-SPHERICITY problem, which is the recognition problem for k-sphere graphs. A graph G is considered

a k-sphere graph if there are k-dimensional vectors $\{v_1, v_2, \ldots, v_n\} : v_n \in \mathbb{R}$ such that an edge is added to the graph: $i, j \in E$ iff the euclidean distance between the two vectors i and j is at most 1. A UDG is considered a 2-sphere graph. Clark et al. [12] first conjectured in 1990 that UDG recognition is NP-hard. This conjecture was resolved in 1998 by Breu and Kirkpatrick [25] who proved that 2-SPHERICITY and 3-SPHERICITY is NP-hard by polynomial time reduction from the Satisfiability (SAT) problem. In 2012 Kang and Müller [26] provided a simpler NP-hardness proof that extends to all higher dimensions: $k > 1$ by reduction from the Simple Stretchability problem. Furthermore, Mnëv [27, 28] and later on Shor [29] prove that the Simple Stretchability problem is polynomial time equivalent to the Existential Theory of the Reals (ETR) decision problem, where given a sentence in the following form:

$$(\exists X_1) \ldots (\exists X_k) F(X_1, \ldots, X_k)$$

and $F(X_1, \ldots, X_k)$ denotes a quantifier-free boolean formula containing the following signatures: $(0, 1, +, \times, <)$. The ETR decision problem is to decide the truth or falsity of this sentence interpreted over the universe of real numbers [26, 30, 31]. To classify problems that have a connection between geometry and logic, Schäfer introduces a new complexity class denoted by $\exists\mathbb{R}$ and shows that these problems share a similar complexity to deciding the truth of the existential theory of the reals [31]. He then describes three natural $\exists\mathbb{R}$-complete problems, including the Simple Stretchability problem. This means that by proving that k-SPHERICITY is polynomial time equivalent to ETR, Kang and Müller implicitly prove that the UDG Recognition problem is $\exists\mathbb{R}$-complete.

The other problem is *UDG Construction*: Given a set of closed disk diameters on a plane, construct a UDG as defined in 3.4. Breu [32] compares this problem to the Fixed-Radius Near Neighbours problem as defined by Bently et al. [33], Dickerson and Drysdale [34] and Lenhof and Smid [35]. Fortunately, this is an extensively researched problem with literature focusing mainly on brute-force, tree-based or grid-based solutions. With the inefficient brute force approach, every disk centre is checked to see if the corresponding disk centres are within distance of each other and reports all edges in $O(n^2)$ time. Alternatively, Zhou et al. [36], Qiu et al. [37] Otair [38] and Gieseke et al. [39], all show different approaches using KD-trees. However, due to the dynamic manner of trees, they are not well suited for GPU's. Faster approaches are obtained using grid structures as demonstrated by Green [40] and Hoetzlein [41]. Later on Groß et al. [42] improved on the memory consumption of these approaches by utilizing the shared memory of a GPU. Most recently, de Balsch [43] presents an algorithm implemented in Julia, that uses cell lists to search for fixed radius nearest neighbours in $O(n)$ time in low dimensional spaces, which is ideal for the two dimensional space of a UDG.

2.2.2 UDG Application

The idea of modelling geometric objects on a plane as a UDG is applicable in many areas. In a wireless network, devices are mapped to a disks on a plane where the radius is the broadcasting range. An example of a distributed wireless network is a mobile ad-hoc network as described by Santi [44], Cordeiro and Agrawal [45], and Kiess and Mauve [46]. This network type does not rely on centralized base stations, instead messages are routed through multiple connected mobile devices, where two mobile devices are connected if their signal range intersect. This results in a robust and decentralized wireless network.

Some problems that arise in wireless networks are directly solvable using optimization problems for graphs. According to Clark et al. [12] the Frequency Assignment problem [47], assign different frequencies to transmitters whose ranges intersect, is solved using the intersection model with graph colouring and the Emergency Sender problem [48], find the minimum set of transmitters than can transmit to all others, is solved using the containment model with graph dominating set. Furthermore according to van Leeuwen [49] the MIS is used to find the largest set of devices that could transmit simultaneously without interfering one another and the Minimum Set Cover is used to find the minimum number of base stations required to provide wireless coverage to all points in an area.

In other areas such as biology, the graph clique problem corresponds to the problem of counting the number of clumped particles under a microscope as described by Armitage [50]. Another application area is Very Large Scale Integration (VLSI) where Hochbaum [22] proposes to use the shifting technique on planar graphs as a solution to the Square Packing problem, i.e. to increase the yield of VLSI chip manufactures by packing the maximum number of working chips onto a wafer.

3 Preliminaries

3.1 Distance Constrained Cinema Seating Problem

In this section, a formal definition of the DCCS problem is provided. The problem is originally inspired by the distance regulations imposed by the Dutch government during the Corona pandemic for seating groups of people from different households in a cinema.

Definition 3.1. An input instance of DCCS problem consists of the following information:

1. The width of a cinema: $w \in \mathbb{N}$
2. The height of a cinema: $h \in \mathbb{N}$
3. A (0,1)-matrix of size $w \cdot h$: M where a 1 denotes an empty seat and a zero denotes a space.
4. A multi-set $T = \{x_1, x_2, ..., x_8\}$ that indicates the number of groups of size i.

The problem is to seat as many people as possible according to the structure of M and output a (0,1,2)-matrix: M' of size $w \cdot h$, where a 2 denotes a seated person. The seating matrix M' is a valid solution to the input instance if all the following constraints hold on it:

1. More groups cannot be seated than defined by the input.
2. All persons in the same group should sit next to each other in the same row, without interruptions by empty seats or spaces.
3. There must be at least two empty seats or spaces between any two groups seated in the same row.
4. There must be at least one empty seat or space between two groups seated in the same column.
5. Two groups cannot sit directly diagonally from each other.
6. No person can be seated on a space.

An example input instance for the DCCS problem and a valid output for it are shown in Figure 2.

```
2
7
1110111
1110111
0 2 0 0 0 0 0 0
```

```
2210111
1110122
```

Figure 2: Example of an input instance for the DCCS (left) and a valid output for it (right).

3.2 Graph Theory

This section defines the relevant graph theory concepts, but also introduces the concept of Seating Paths and their degree, both of which are repeatedly referred to in Chapter 6.

Definition 3.2 (Graph)**.** A graph is an abstract data structure used to show a binary relation between entities and is denoted as $G = (V, E)$. V is the set of entities, which are called vertices; E is the set of edges, where each pair of entities that is in relation to its other is called an edge. In this paper, all graphs are assumed to be undirected, i.e., the relation between entities is symmetric. Also weighted graphs are considered, where vertices and/or edges are assigned an integer or real value, called the weight of the vertex or edge.

Definition 3.3 (Maximum Independent Set)**.** Given a graph $G = (V, E)$, an IS is a subset of vertices $S \subseteq V$ of G such that no two vertices in S are adjacent to each other in G. A MIS is an IS of maximum cardinality in G. Then the MIS problem is defined as: Given a graph G, find a MIS in G.

In the weighted version of the problem, each vertex in G is assigned a weight. The maximum weighted independent set is the IS of maximum cardinality in G where the sum of all weights in S is the maximum possible value.

Definition 3.4 (Unit Disk Graphs)**.** Given a set of *n* equally sized circles on a plane, a UDG is an intersection graph with *n* vertices where each vertex corresponds to a circle. An edge is added for each vertex pair, if the corresponding circles intersect on the plane. Clark et al. [12] refers to this type of UDG as the *intersection* model. They also describe two additional models: *containment* and *proximity*. For the containment model an edge is added for each vertex pair if one of the corresponding circles contains the other's centre point. As for the proximity model, an edge is added for each vertex pair if and only if the Euclidean distance between the corresponding centre points is a most some specified bound. In this paper only the *intersection* model is considered.

Definition 3.5 (Seating Path)**.** Given a cinema graph $G = (V, E)$ a seating path p_v is a

finite sequence of horizontally adjacent vertices ordered by their indices:

$$p = \langle v, v+1, v+2, ..., v+n \rangle, \qquad \begin{array}{l} 0 \leq n < 8, \\ \forall\, v' \in p : v' \in V \\ \forall\, 0 \leq i \leq (n-1) \wedge i \in \mathbb{N}^+ : (p_i, p_{i+1}) \in E \end{array}$$

Definition 3.6 (Degree of a Seating Path)**.** The degree of a seating path is equal to the summed degree of all vertices contained in the seating path.

$$degree(p_v) = \sum_{v \in p_v} degree(v)$$

3.3 Ordering

Whenever an ordering is mentioned in this paper it refers to the following definition:

Definition 3.7. An ordering is a sequence of items and three types of orderings are implemented as part of this research:

1. **Largest First** orders items from largest to smallest:

$$O : \langle o_0, o_1, ..., o_{n-1} \rangle, \qquad \begin{array}{l} n \in \mathbb{N}^+ \\ \forall\, 0 \leq i \leq (n-1) : O_i \geq O_{i+1} \end{array} \tag{1}$$

2. **Smallest First** orders items from smallest to largest:

$$O : \langle o_0, o_1, ..., o_{n-1} \rangle, \qquad \begin{array}{l} n \in \mathbb{N}^+ \\ \forall\, 0 \leq i \leq (n-1) : O_i \leq O_{i+1} \end{array} \tag{2}$$

3. **Random** orders the items in no particulate order.

$$O : \{o_0, o_1, ..., o_{n-1}\}, \quad n \in \mathbb{N}^+ \tag{3}$$

4 Graph Construction

In this section an algorithm is given to construct an undirected graph G from a given seating matrix M of size $w \cdot h$. The first step is to map each seat and space in the seating matrix M to an integer coordinate on a plane starting from the bottom left. Then for each coordinate, add a vertex to G and label it according to the labelling function defined in Equation 4.

$$label(M, x, y) = \begin{cases} e & \text{if there is a 1 at position } (x, y) \text{ in } M \\ s & \text{if a there is a 0 at position } (x, y) \text{ in } M \end{cases} \tag{4}$$

The second step is to add edges based on the distance constraints defined in 3.1 for a group of size 1. For example, if a seat is positioned at coordinate $(0, 0)$, the invalid seats are located at coordinates: (1,0); (2,0); (0,1); (1,1) on the plane. An edge is added to G from the vertex represented at coordinate $(0, 0)$ to each vertex represented by the coordinates of the invalidated seats. Finally, each edge is weighted according to the weight function defined in Equation 5 to denote if an edge is horizontally oriented or not.

$$weight(y_1, y_2) = \begin{cases} 0 & \text{if } y_1 = y_2 \\ 1 & \text{else} \end{cases} \tag{5}$$

A resulting cinema graph of applying both above steps is shown in Figure 3.

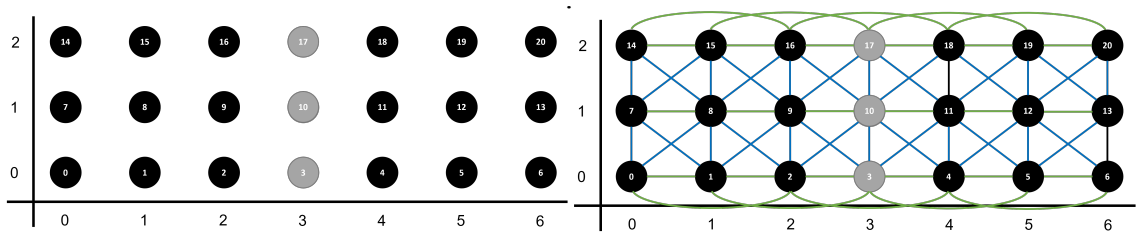


Figure 3: Construction result of applying step 1 (left) and step 2 (right) to the problem instance shown in Figure 2. The green colored edges have a weight of 0 and the blue colored edges have a weight of 1. The black vertices are labelled e and the grey vertices are labelled s.

Given the constructing constraints of a cinema graph it has some unique properties which are described in the following three Lemmas:

Lemma 4.1. Given a seating matrix M of width w and height h, the order of the resulting graph G is: $w \cdot h$.

Proof. Lemma 4.1 holds because both empty spaces and seats in M are added to G as vertices. Therefore the number of vertices in graph G is always equal to the area of M. □

Lemma 4.2. Given a seating matrix of width $w > 1$ and height h, the number of edges in the resulting cinema graph G is denoted by Equation 6.

Proof. The left component of Equation 6 denotes the total number of horizontal edges in graph G. Whereas the right component denotes the total number of vertical and diagonal edges in graph G. However, for seating matrices with $h = 1$ the overall result of this equation would be negative, which is why the extra constraint on the width of the seating matrix M is required for Lemma 4.2 to hold. □

$$(((w-1)+(w-2)) \cdot h) + ((w + ((w-1) \cdot 2)) \cdot (h-1)) \tag{6}$$

Lemma 4.3. Given a seating matrix of width $w \geq 5$ and height $h \geq 3$ The maximum degree of a vertex in G is 10.

Proof. Lemma 4.3 holds because of the seating constraints defined in Section 3.1. Based on these constraints the maximum number of illegal seats for a group of size one is 10 and only occurs for seating matrices of width $w \geq 5$ and height $h \geq 3$. Then there exist seats that have all distance constraints (vertical, horizontal, and diagonal) applied in all compass directions. Since the construction algorithm adds edges to vertices based on their illegal seats and does not differentiate between spaces and seats, the resulting graph G contains vertices with a maximum degree of 10. □

To implement the construction algorithm a data structure is required to represent the vertices and edges of an undirected cinema graph $G = (V, E)$. The choice of data structure is evaluated on the following two criteria: *space* and *querying* complexity. The first criterion is because the graph is stored in memory and using a data structure that has a worse space complexity could result in memory issues for larger cinema instances. The second criterion is that the graph querying is used in Chapters 5 and 6, so a worse querying complexity impacts the performance of finding a MIS and seating groups. Based on these two criteria the construction algorithm is implemented using adjacency list instead of a adjacency matrix because of the following three reasons:

1. The space complexity of an adjacency matrix is $\mathcal{O}(n^2)$, where n is the order of the graph. By applying Lemma 4.1 it shows that the space requirement of an adjacency matrix is quadratic in the width and height of a cinema. Alternatively, the space complexity of an adjacency list is $\mathcal{O}(n + k)$, where n is the order and k are the

number of edges of the graph. This time by applying Lemma 4.2 it shows that the space complexity of an adjacency list is linear in the width and height of a cinema.

2. The querying complexity of an adjacency list is $\mathcal{O}(k)$, where k is the degree of the vertex that is being queried. Since Lemma 4.3 shows that the maximum degree of a vertex in G is bounded, the querying complexity on G is $\mathcal{O}(1)$. On the other hand, the querying complexity of an adjacency matrix is a simple lookup which also has an $\mathcal{O}(1)$ complexity.

3. It is the consensus that adjacency lists are better suited as a data structure for sparse graphs. There is no formal definition on when a graph is considered sparse. Let Equation 7 denote the size of a complete graph where n is its order.

$$n(n-1)/2 \tag{7}$$

By replacing n with $w \cdot h$ according to Lemma 4.1 and comparing it to the result of Equation 6, it shows that the actual size of graph G moves away from its complete graph counterpart as the cinema size increases. For example, by applying both equations to a cinema with 25 seats the resulting cinema graph has 76% less edges than its complete version. Increasing the seats to 100 results results in 92% less edges. This means that with increasing cinema size the resulting graph becomes sparser, which makes the adjacency list more favourable for larger cinema instances.

After selecting the data structure, the next step is to implement the algorithm according to the two steps described above using an adjacency list as the cinema graph representation. The pseudo code for the construction process is shown in Algorithm 1.

Lemma 4.4. The time complexity of Algorithm 1 is $\mathcal{O}(w \cdot h)$.

Proof. By ignoring the constant time operations, the remaining relevant operations are found on lines: 4 and 9. Since line 4 is bounded by the number of vertices denoted by N and line 9 is bound by Lemma 4.3, the resulting complexity is $\mathcal{O}(N)$. By replacing N according to Lemma 4.1, the algorithm has a linear complexity to the width and height of a cinema: $\mathcal{O}(w \cdot h)$. □

Algorithm 1 Pseudo code for the construction algorithm.

```
1:  function construct(w, h, M, coords, indices)
2:      N ← w × h
3:      G ← adjList(N)
4:      for n ← 0 to N do
5:          x_1, y_1 ← coords_n                ▷ Lookup the coordinate of a vertex
6:          l ← Label(M, x_1, y_1)
7:          SetLabel(G, l)
8:          invalidSeats ← invalid(x_1, y_1)
9:          for all x_2, y_2 ∈ invalidSeats do
10:             is ← indices_{x_2,y_2}         ▷ Lookup the vertex of a coordinate
11:             w ← Weight(y_1, y_2)
12:             AddEdge(G, n, is, w)
13:         end for
14:     end for
15:     return G
16: end function
```

5 MIS Algorithms

Once the graph representation of the cinema is constructed, the next step is to determine a MIS of the cinema graph. Considering that the MIS problem is NP-hard, implementing an optimal algorithm that runs in polynomial time is not possible, unless $P = NP$. The first algorithm is called IS_GREEDY and is a simple deterministic polynomial time algorithm used to find an IS of **maximal** cardinality for a given cinema graph. The second is called IS_ILP and is a non-deterministic ILP implementation used to find the MIS of a given cinema graph. The reasons for implementing these two algorithms as part of this research are:

1. IS_GREEDY algorithm can be applied on larger cinema sizes where IS_ILP may fail to compute a solution within reasonable time bounds.
2. IS_GREEDY may produce a smaller IS size than IS_ILP, this can be used in Section 7 to determine the impact of IS size on the ability of seating algorithms to seats groups.

The third algorithm is a deterministic Fixed Parameter Tractable (FTP)-algorithm that exploits the geometric structure of a cinema to find a MIS of a given cinema graph. However, unlike the other two algorithms, only the theory of this algorithm is presented as part of this research.

5.1 Greedy

This algorithm is based on the *Minimum Degree Greedy* algorithm as described in [51, 52, 53]. Given a graph $G = (V, E)$ it finds an IS of maximal cardinality by iteratively selecting a vertex with lowest degree from V, adding it to the result set and then removing it and all of its neighbours from V. Once all vertices have been removed the algorithm halts and returns the resulting set. To ensure that IS_GREEDY only includes seats and not spaces, the algorithm is adjusted to filter out vertices labelled with s. The pseudo code based on this description is shown in Algorithm 2.

Lemma 5.1. IS_GREEDY is a 4-approximation algorithm for all cinema graphs of of width $w \geq 5$ and height $h \geq 3$.

Proof. According to [51] the *Minimum Degree Greedy* algorithm achieves a performance ratio of $(\Delta + 2)/3$ for approximating an IS of maximum cardinality for a given graph G, where G belongs to the class of graphs with maximum degree bounded by Δ. From Lemma 4.3 it follows that cinema graphs of sufficient size belong to the class of graphs with maximum degree bounded by 10. Therefore the resulting approximation guarantee of IS_GREEDY on cinema graphs is 4. □

Algorithm 2 Pseudo code for the IS_GREEDY algorithm.

```
1:  function IS_GREEDY(G = (V, E))
2:      W ← V \ labels("s")
3:      IS ← ∅
4:      while W ≠ ∅ do
5:          v ← MinDegree(E, W)
6:          W ← W \ neighbours(v) ∪ {v}
7:          IS ← IS ∪ {v}
8:      end while
9:      return IS
10: end function

11: function MinDegree(E,W)
12:     min ← degree(E, W_0)
13:     for all w ∈ W do
14:         if degree(E, w) < min then
15:             min ← w
16:         end if
17:     end for
18:     return min
19: end function
```

Lemma 5.2. IS_GREEDY runs in polynomial time to the number of seats in the cinema.

Proof. The dominant operations in Algorithm 2 are on lines 2 and 4. If in the worst case no additional vertices are removed during each iteration step, then IS_GREEDY has a linear time complexity of $\mathcal{O}(n)$, where n denotes the order of the graph. From Lemma 4.1, it follows that the order of graph G is equal to the number of seats in the cinema. □

5.2 ILP

The ILP encoding in Equation 9 shows a solution to the weighted version of the MIS problem on a cinema graph:

- Let the **decision variable** x_v denote whether the vertex v is included in the resulting set or not. To decode the ILP result all variables where $x_v = 1$ are selected and added to the IS.
- The **objective** of the model is to maximize the number of vertices. To ensure that the resulting set only contains seats and not spaces, the weight function in Equation 8 favours vertices labelled with *e*.

$$w(v) = \begin{cases} 1 & label(v) = e \\ -1 & \text{otherwise} \end{cases} \tag{8}$$

- Lastly, to ensure the solution of the ILP is a valid IS the following **constraint groupings** are added to the model:

  Firstly, the *One vertex per edge* constraint grouping ensures that for every edge $e \in E$ at most one of its connecting vertices can be included in the IS.

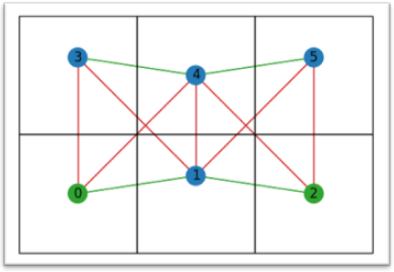
  Secondly, the *Binary* constraint grouping ensures that every vertex is either included in the IS or is not included.

$$\begin{aligned} &\max_{v} \quad \sum_{v \in V} w(v) \times x_v && \textit{(Objective)} \\ &\text{s.t.} \\ &\qquad x_u + x_v \leq 1, \quad \forall \{u, v\} \in E && \textit{(One vertex per edge)} \\ &\qquad x_v \in \{0, 1\}, \qquad \forall v \in V && \textit{(Binary)} \end{aligned} \tag{9}$$

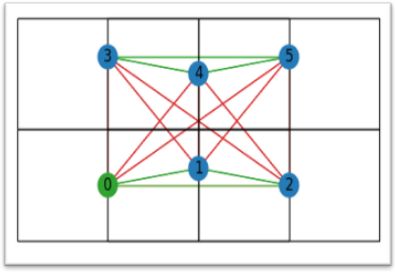
5.3 FPT-algorithm

Both previously introduced algorithms find the IS using the provided cinema graph as their only input. By also providing a geometric structure of the cinema layout, this can be exploited to implement a more efficient algorithm for finding the MIS of a cinema graph.

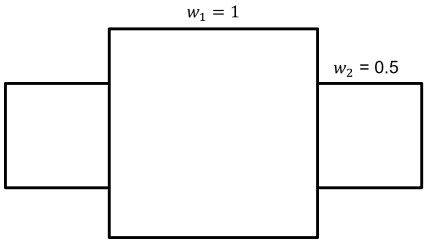
Before discussing this algorithm, it is important to consider that the shape of unit disks does not exactly match the distance requirements defined by the DCCS problem. This is because the problem defines the horizontal distance between two groups to be two seats and the vertical and diagonal distances to be one seat. However, a disk with radius r has equal distances in all compass directions. This means that if r is sufficiently large enough to include two seats horizontally then it also includes two seats vertically and diagonally and if the r is sufficiently small, then only one seat is included in all directions. Since disks are not the ideal shape for modelling these distance constraints, some other simple shapes were investigated as part of this research: Squares, rectangles, and crosses. Even though the intersection of a rectangle or square is easy to calculate, these two shape types do not model the seating constraints accurately either as shown in Figures 4a and 4b respectively. Squares have the same issue as disks and rectangles that have sufficient width to include two horizontal seats and sufficient height to include one vertical seat, will also include the seat adjacent to the diagonal seat. Alternatively, a cross with the dimensions shown in Figure 4c models the distance constraints perfectly as shown in Figure 4d. Calculating the intersection of two crosses is also simple, because a cross consists of three squares.



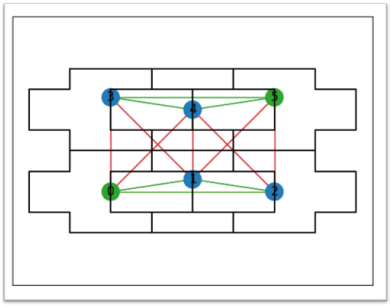
(a) Cinema graph using squares with width of one unit.



(b) Cinema graph using rectangles with width of two units and height of one unit.



(c) Dimensions of a cross shape where the inner square (w_1) has a width of 1 unit and the two outer squares (w_2) have a width of 0.5 units



(d) Cinema graph using crosses based on the dimensions shown in the Figure (c).

Figure 4: Showing the resulting cinema graphs based on the intersection model of various shapes. The green vertices are included the IS.

With the cross shape in mind, the algorithm described in this section was originally defined by van Leeuwen [24] and is a FTP-algorithm that is polynomial bounded on the thickness of a UDG, where the thickness is determined by segregating the disk representation into equally spaced slabs and counting the maximum number of disk centres that fit in a slab. An example of this concept is shown in Figure 5.

Definition 5.1 (Slab Decomposition)**.** According to van Leeuwen, a slab decomposition $s = \langle \alpha, p \rangle$ is a set of infinite parallel lines on a plane, that are one unit apart, intersect the x-axis at angle α and exactly one line must pass through the coordinate p. [24]

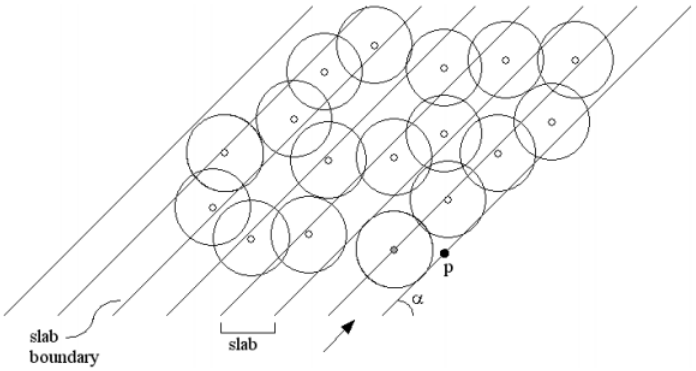


Figure 5: Example of a UDG with a slab decomposition created by van Leeuwen. [24, p.6]

In his paper van Leeuwen applies the *Optimality Principle* to a slab decomposition of a UDG and uses Dynamic Programming (DP) to compute the MIS on that graph.

Definition 5.2 (Optimality Principle)**.** Bellmans optimality principle holds when an optimal solution to a given problem can be obtained by using the optimal solutions to its sub-problems.

Given a UDG: $G = (V, E)$ and a slab decomposition: s, he proceeds to denote Y_j as:

> *"Let $Y_1, Y_2, \ldots, Y_b$ be subsets of: V, such that Y_j contains those vertices corresponding to disk centres of D in the j-th non-empty slab of s." [24, p.6]*

Van Leeuwen proves that the Optimally Principle holds for the MIS problem on UDGs with disk radius 0.5 because of the geometric constraints between slabs and disks, where any vertex in slab Y_j can only have edges to vertices contained in slabs Y_{j-1}, Y_j and Y_{j+1}. The proof of Lemma 5.3 shows that this same proposition also holds on a cinema graph, which means his DP algorithm can be applied to a cinema graph to more efficiently compute a MIS.

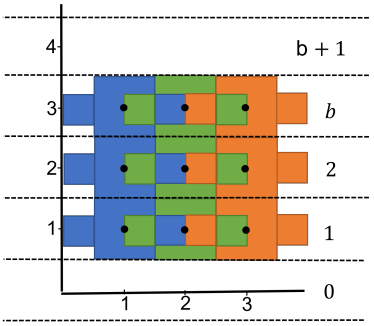
Lemma 5.3. There exists a slab decomposition s for a cinema graph $G = (V, E)$ such that any vertex in Y_j can only have edges to vertices contained in Y_{j-1}, Y_j and Y_{j+1}

Proof. Given a cross representation $C = \{(c_i, (w_1, w_2)) \mid i = 1, \ldots, n\}$ of a cinema layout. Where n is the total number of seats in the cinema, c_i is the 2D integer coordinate of seat i, w_1 is the width of the cross's inner square and w_2 is the width of the cross's two outer squares. To model the distances constraints of the DCCS problem the following widths in units are assumed: $w_1 = 1$ and $w_2 = 0.5$.

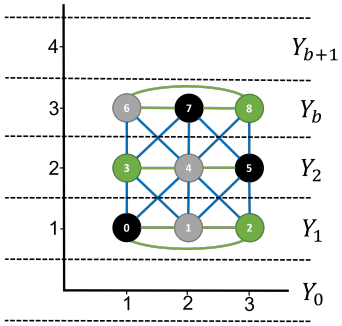
Let $s = \langle \alpha, p \rangle$ be a slab decomposition of C. By setting $\alpha = 0$ and $p = (0, -0.5)$ the plane is split into $j = 1, 2, \ldots b$ horizontal slabs where each slab contains the seats belonging to the j^{th} row of the cinema. A slab decomposition of C is shown in Figure 6a. It is important to note, that the slabs 0 and $b + 1$ are dummy slabs that are introduced by van Leeuwen for convenience and are later required by the algorithm.

Then let $\{Y_1, Y_2, \ldots, Y_b\}$ be the subsets of vertices V for the cinema graph: $G = (V, E)$, such that Y_j contains those vertices corresponding to the seats in the j^{th} non-empty slab of s. A representation of G is shown in Figure 6b. The figure clearly illustrates that all vertices contained in any slab Y_j only have edges to other vertices contained in Y_{j-1}, Y_j and Y_{j+1}.

The is because, even though a disk with $r = 0.5$ and a cross with dimensions $w_1 = 1$ and $w_2 = 0.5$ occupy different widths $2r \neq w_1 + 2w_2$, they both occupy the same height: $2r = w_1$. By positioning the slabs horizontally, this height ensures that two crosses in different slabs only intersect if they are in adjacent slabs. If the slabs were positioned vertically or diagonally this proposition would not hold because the width of a cross is larger than a unit disk with $r = 0.5$. □



(a) An example of a cross representation of a cinema layout with slab decomposition $s = \langle 0, (0, -0.5) \rangle$. The thickness of this cinema graph is three.



(b) An example of a cinema graph with slab decomposition $s = \langle 0, (0, -0.5) \rangle$. Grey vertices represent empty spaces and green vertices are included in the IS.

To demonstrate how this algorithm computes the optimal MIS, the pseudo code shown in Figure 7 is applied to the cinema graph shown in Figure 6b. The steps are described below and the result of each step is summarised in Table 3. To ensure that IS_FPT only includes seats and not spaces, the check on line 6 can be adjusted to only consider vertices labelled with *e*.

1. The first slab is selected and all subsets of vertices in the slab Y_1: $\{0, 1, 2\}$ are compared to the subsets of vertices in the previous slab Y_0: $\emptyset$. Since the previous

slab is an dummy slab, only independent subsets in Y_1 are included in the solution: $\{\{0,2\},\{0\},\{2\},\emptyset\}$.

2. Then the second slab is selected, but this time the previous slab contains other vertices. Every subset of vertices W_2 in slab Y_2: $\{\{3,5\},\{3\},\{5\},\emptyset\}$ is unified with every subset of vertices W_1 in Y_1: $\{\{0,2\},\{0\},\{2\},\emptyset\}$ and each result is checked for independence. If the set is independent, W_2 is unified with the solution to W_1 of the previous slab. In the case where no vertex is selected in Y_2 i.e. $W_2 = \emptyset$ the previous solution with the largest IS is selected.

3. The third step is like step 2. However, the subset of vertices $\{7,8\}$ does not have an independent with any subset of the previous slab Y_2. Therefore the size and solution are set to -∞ and $\emptyset$ respectively. Unlike the previous step, when no vertex is selected in Y_3 i.e. $W_3 = \emptyset$ there are multiple IS in the previous solutions of the same size. In that case, the first occurrence of largest IS is selected instead.

4. The last slab Y_4 is a dummy slab containing no vertices, so the first occurrence of an IS of maximum cardinality is selected from the previous solution.

5. Since there are no more slabs left to iterate, the algorithm halts and returns the MIS located at $\text{sol}_4(\emptyset) = \{2,3,8\}$

Steps	**Subsets**			
Step 0:	$W_0 = \emptyset$			
size_0	0			
sol_0	$\emptyset$			
Step 1:	$W_1 = \{0,2\}$	$W_1 = \{0\}$	$W_1 = \{2\}$	$W_1 = \emptyset$
size_1	2	1	1	0
sol_1	$\text{sol}_0(\emptyset) \cup \{0,2\}$	$\text{sol}_0(\emptyset) \cup \{0\}$	$\text{sol}_0(\emptyset) \cup \{2\}$	$\text{sol}_0(\emptyset) \cup \emptyset$
Step 2:	$W_2 = \{3,5\}$	$W_2 = \{3\}$	$W_2 = \{5\}$	$W_2 = \emptyset$
size_2	2	2	2	2
sol_2	$\text{sol}_1(\emptyset) \cup \{3,5\}$	$\text{sol}_1(\{2\}) \cup \{3\}$	$\text{sol}_1(\{0\}) \cup \{5\}$	$\text{sol}_1(\{0,2\}) \cup \emptyset$
Step 3:	$W_3 = \{7,8\}$	$W_3 = \{7\}$	$W_3 = \{8\}$	$W_3 = \emptyset$
size_3	-∞	1	3	2
sol_3	$\emptyset$	$\text{sol}_2(\emptyset) \cup \{7\}$	$\text{sol}_2(\{3\}) \cup \{8\}$	$\text{sol}_2(\{3,5\}) \cup \emptyset$
Step 4:	$W_4 = \emptyset$			
size_4	3			
sol_4	$\text{sol}_3(\{8\}) \cup \emptyset$			

Table 3: Table showing results of each step of IS_FPT computing the MIS for the cinema graph shown in Figure 6b

```
1.  Set size_0(∅) = 0 and solution_0(∅) = ∅
2.  for   j ← 1 to b + 1
3.  do    for each  W_j ⊆ Y_j
4.        do    size_j(W_j) = −∞; solution_j(W_j) = ∅
5.              for each  W_{j−1} ⊆ Y_{j−1}
6.              do    if    W_j ∪ W_{j−1} is an independent set
7.                    then  if    |W_j| + size_{j−1}(W_{j−1}) > size_j(W_j)
8.                          then  size_j(W_j) ← |W_j| + size_{j−1}(W_{j−1})
9.                                solution_j(W_j) ← W_j ∪ solution_{j−1}(W_{j−1})
10.                         fi
11.                   fi
12.             od
13.       od
14. od
15. return (size_{b+1}(∅), solution_{b+1}(∅))
```

Figure 7: Pseudo code for the IS_FPT algorithm written by van Leeuwen [24, p.11]

Lemma 5.4. IS_FPT runs in $\mathcal{O}(w^2 2^{2w} h)$ time, where w and h are the width and height of the cinema respectively.

Proof. According to van Leeuwen IS_FPT runs in $\mathcal{O}(t^2 2^{2t} n)$ time, where t is the maximum thickness of the UDG G and n is the number of slabs in the slab decomposition s of G. Given that the slab decomposition of a cinema graph dissects the cinema into horizontal slabs, where each slab contains the seats belonging to the j^{th} row of the cinema. It is clear that the thickness of a cinema graph is equal to the number of seats and spaces in a row (width) and the number of slabs (ignoring dummy slabs) is equal to the number of rows (height). □

6 Cinema Seating Algorithms

In this section three seating algorithms and a heuristic are presented as a solution to the DCCS problem.

6.1 Greedy

The first algorithm is a greedy algorithm that takes a cinema graph $G = (V, E)$ and an ordering of group sizes O as input and returns a mapping μ of vertices to integers. A formal definition of this algorithm is defined below:

$$seat\_greedy : (G, O) \to \mu : V' \mapsto \{1, 2...8\}, \quad V' \subseteq V(G) \tag{10}$$

To correctly seat groups, SEAT_GREEDY utilizes the concept of seating paths. For this algorithm a valid seating path for a given vertex $v \in V$ is constructed by pathing to the right of v and including all vertices encountered (14) until: A invalid vertex is encountered (11), the maximum group size of g is exceeded (12) or there are no more seats to iterate (13). A formal definition for this function: $p_v = spath(v, \langle\rangle, \oplus, 8)$ is defined below:

$$spath(v, p, \oplus, g) = \begin{cases} p & \text{if } label(v) = \text{"s"} \quad (11) \\ p & \text{if } |p| = g \quad (12) \\ \langle p, v \rangle & \text{if } (v, v \oplus 1) \notin E \quad (13) \\ spath(v \oplus 1, \langle p, v \rangle) & \text{else} \quad (14) \end{cases}$$

The algorithm iterates through the group sizes according to the given ordering $\langle o_0, ..., o_n \rangle$. For each group size, denoted by o_i, the algorithm tries to find a vertex $v \in V$ whose seating path p_v matches the current group size. If multiple vertices are found, the vertex whose seating path has the lowest degree is selected instead. Once a vertex has been selected, all vertices in p_v and adjacent to it are removed from G and the selected vertex is mapped to the current group size. In the case that there exists no seating path that satisfies the current group size, a vertex whose seating path is one larger is selected instead. This process is repeated until either a vertex is found, or the maximum group size is exceeded, in which case the current group cannot be seated and is skipped. The algorithm halts when either all vertices are removed from G, or the ordering has been exhausted. The pseudo code based on this description is shown in Algorithm 3 and its run time complexity is described in Lemma 6.1.

Algorithm 3 Pseudo code for the SEAT_GREEDY algorithm.

```
1:  function SEAT_GREEDY(G = (V, E), O)
2:      μ ← ∅
3:      while V ≠ ∅ ∨ O ≠ ∅ do
4:          p_v ← FindPath(V, O_i)
5:          if p_v ≠ null then
6:              μ(v) = o
7:              V ← V \ neighbours(p_v) ∪ {p_v}
8:          end if
9:      end while
10:     return μ
11: end function

12: function FindPath(V, o)
13:     P ← ∅
14:     while P = ∅ ∧ o ≤ 8 do
15:         for all v ∈ V do
16:             p_v ← spath(v, ⟨⟩, ⊕, 8)
17:             if |p_v| = o then
18:                 P ← P ∪ {p_v}
19:             end if
20:         end for
21:         o ← o + 1
22:     end while
23:     if P ≠ ∅ then
24:         p_v ← min_{p∈P} degree(p)
25:         return p_v
26:     end if
27:     return null
28: end function
```

Lemma 6.1. SEAT_GREEDY runs in polynomial time in the number of seats in a cinema.

Proof. The dominant operation in Algorithm 3 is on line 4 and is dependent on either the order G or the total number of groups. Of these two, the order of G is the dominant variable because groups are dependent on vertices existing in V. In the worst case, SEAT_GREEDY has a linear time complexity of $\mathcal{O}(n)$, where n denotes the order of the graph. From Lemma 4.1, it follows that the order of graph G is equal to the number of seats in the cinema. □

To illustrate how SEAT_GREEDY works, an example is described below and illustrated in Figure 8, where it is applied to a cinema graph of size $w = 7$ and $h = 3$ and a *Largest First* ordering of: $O = \langle 2, 2, 1, 1 \rangle$.

1. According to the ordering, the first group to be seated has a size of two. The vertices that can accommodate that size are located at indices: $\{p_1, p_5, p_8, p_{12}, p_{15}, p_{19}\}$ Since there are several seating paths, the set is further reduced to only include the following seating paths of minimum degree: $\{p_1, p_5, p_{15}, p_{19}\}$. From this set, the first vertex is selected and all vertices in the seating path p_1 and adjacent to it are removed from the graph.

2. In the second iteration, another group of size two needs to be seated. Since the first iteration removed vertices, the remaining valid seating paths are: $\{p_5, p_{12}, p_{15}, p_{19}\}$, reducing this set further results in a single seating path: $\{p_{15}\}$. This vertex is selected and identical to the first step, all vertices in the seating path and adjacent to it are removed from the graph.

3. In the third iteration a substantial number of vertices have already been removed from the graph. Seating a group of size one only results in the following two seating paths: $\{p_6, p_{20}\}$, which reduced to: $\{p_6\}$.

4. Finally, the last group of size one has seating paths: $\{p_{11}, p_{20}\}$. Since both seating paths have the same degree, the first one is selected. With all groups iterated and one unused vertex remaining, the algorithm halts and outputs the following mapping:

$$\{\{1, 2\}, \{15, 2\}, \{6, 1\}, \{11, 1\}\}$$

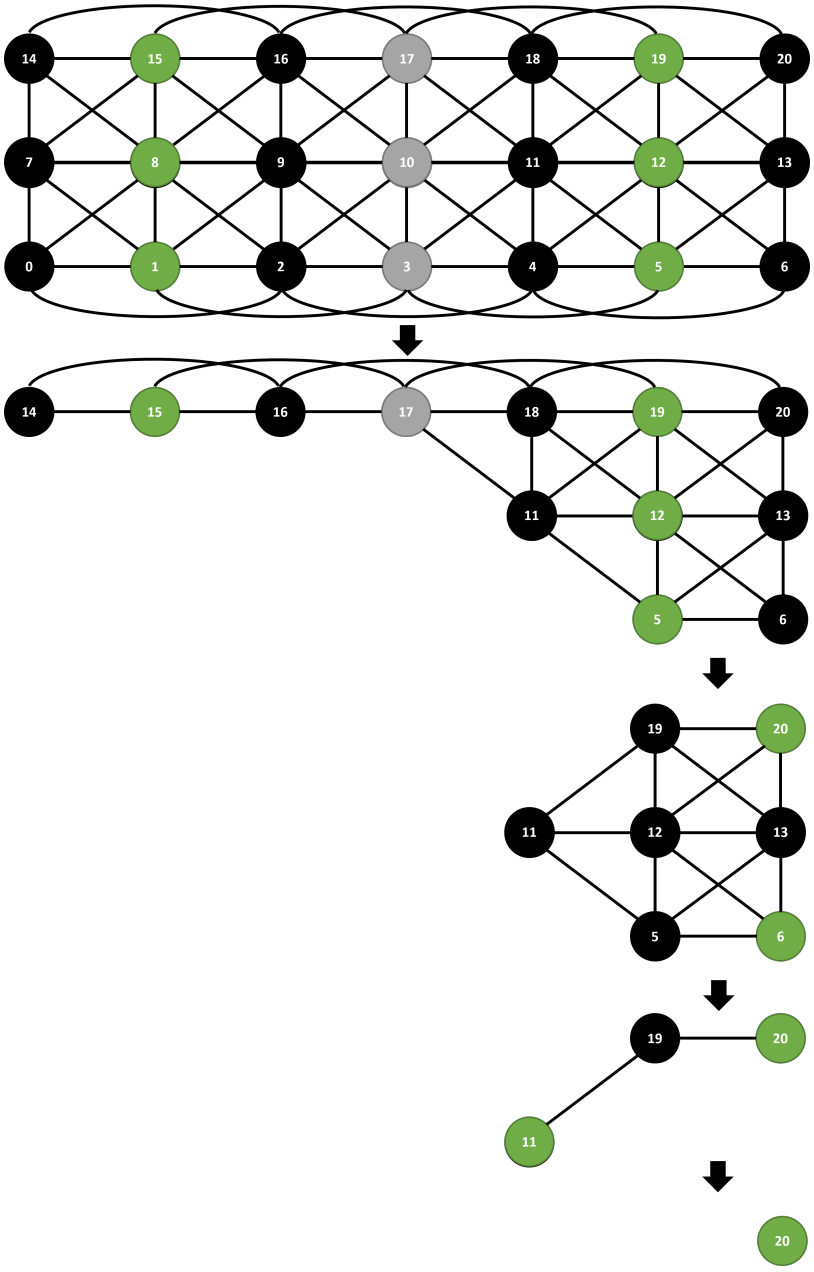


Figure 8: Step by step example of SEAT_GREEDY applied to the problem instance shown in Figure 2 with a LargestFirst ordering. The green coloured vertices represent the possible vertices where the current group could be seated.

6.2 Greedy with IS

Like SEAT_GREEDY, the SEAT_GREEDY_IS algorithm also utilizes the concept of seating paths but only considers vertices in the IS. SEAT_GREEDY_IS defined in Equation 19 is a function that takes a cinema graph $G = (V, E)$, a multiset of group sizes T and an ordering of vertices O as input and returns a mapping μ of vertices to integers. The ordering influences the way this algorithm iterates through the cinema: *Smallest First* (SF) ordering means from bottom left to the top right, *Largest First* (LF) means from top

right to bottom left and *Random* (RAND) means the algorithm jumps from seat to seat randomly.

$$seat\_greedy\_is : (G, T, O) \rightarrow \mu : V' \mapsto \{1, 2...8\}, \quad V' \subseteq V(G), O \subseteq V(G) \tag{15}$$

The construction of a seating path is also adjusted. Where given a vertex $v \in O$ a seating path is constructed that includes v and all encountered vertices to the left and right of v until one of the conditions described in Equations: 11, 12 and 13 are met. The formal definition of this function $spath'(v, p)$ is described in Equation 16.

$$spath'(v, p, g) = \langle spath(v, p, \ominus, g), spath(v, p, \oplus, g) \rangle \tag{16}$$

To seat groups, the algorithm iterates through the vertices in the order defined by its ordering O. For each vertex, denoted by o_i, the algorithm checks that o_i was not removed from G by a previous iteration. If it still exists, the algorithm constructs a seating path p_o for it and uses the Best Fit seating strategy S to find a group size s that matches p_o. It then maps o_i to s and removes all vertices in the p_o and adjacent to it from G. The algorithm halts when either all vertices V have been removed from G, O is depleted or all groups have been seated. The pseudo code for SEAT_GREEDY_IS is shown in Algorithm 4 and its run time complexity is described in Lemma 6.2.

Definition 6.1 (Best Fit)**.** A seating strategy is a function that takes preferred group size n and a multiset of group sizes T as input and outputs and removes the best possible group size in T depending on the strategy. Best Fit tries to find and remove a group size in T that matches the input n. If no such group size exists in T, the next smaller group size is returned instead. The strategy returns 0 if there does not exist a group size smaller or equal to n in T. A formal definition of BestFit is provided in Equations 17 and 18.

$$BF : (n, T = \{g_0, g_1, ...,_{m-1}\}) \rightarrow (x, T \setminus x) \qquad \begin{array}{l} x = fits(n, T) \\ n \in \{1, 2, ..., 8\} \\ \forall\, 0 \leq i \leq (m-1) : T_i \in \{1, 2, ..., 8\} \end{array} \tag{17}$$

$$fits(n, T = \{g_0, g_1, ..., g_{n-1}\}) = \begin{cases} n & \text{if } n \in T \\ 0 & \text{if } n = 0 \\ fits(n-1, T) & \text{else} \end{cases} \tag{18}$$

Algorithm 4 Pseudo code for the SEAT_GREEDY_IS algorithm.

```
1:  function SEAT_GREEDY_IS(G = (V, E), BF, T, O)
2:      μ ← ∅
3:      while V ≠ ∅ ∨ O ≠ ∅ ∨ T ≠ ∅ do
4:          if o_i ∈ V then
5:              p_v ← spath'(o_i, ⟨⟩, 8)
6:              (s, T) ← BF(|p_v|, T)
7:              if s ≠ 0 then
8:                  μ(v) = s
9:                  V ← V \ neighbourhood(p_v) ∪ {p_v}
10:             end if
11:         end if
12:     end while
13:     return μ
14: end function
```

Lemma 6.2. SEAT_GREEDY_IS runs in polynomial time in the number of seats in a cinema.

Proof. The same proof used in Lemma 6.1 is also applied here. The dominant operation in Algorithm 4 is on line 4 and is dependent on either the order G, the IS size or the total number of groups. Of these three, the order of G is the dominant variable because both groups and IS are dependent on vertices existing in V. In the worst case, SEAT_GREEDY_IS has a linear time complexity of $\mathcal{O}(n)$, where n denotes the order of the graph. From Lemma 4.1, it follows that the order of graph G is equal to the number of seats in the cinema. □

To help understand how SEAT_GREEDY_IS works, an example is described below and illustrated in Figure 8, where it is applied to a cinema graph of size $w = 7$ and $h = 3$, a *Smallest First* ordering of: $O = \langle 0, 4, 9, 13, 14, 18 \rangle$ and group sizes $T = [1, 1, 2, 2]$

1. Following the *Smallest First* ordering, a seating path for vertex $v = 0$ is constructed first. Since v is a corner vertex, the seating path p_v extends two extra vertices to the right, meaning a group size three can be seated at p_v. Using the *Best Fit* strategy, the best available group size in T is two. So a group of size two is seated at v and all vertices in and adjacent to p_v are removed from V.

2. The second iteration is identical to the first, except it uses the next vertex in O which is $v = 4$.

3. The third and fourth iterations are skipped because vertices $\{9, 13\}$ were removed by the previous iterations. Therefore the next considered vertex is $v = 14$. Since all

groups of size two are already seated, *Best Fit* returns the next best available group size, which is one.

4. Finally, the last remaining vertex $v = 18$ is selected, the last group of size one is seated there, and the adjacent vertices are removed from V. With no vertices remaining and all groups seated, the algorithm halts and outputs the following mapping:

$$\{\{0, 2\}, \{4, 2\}, \{14, 1\}, \{18, 1\}\}$$

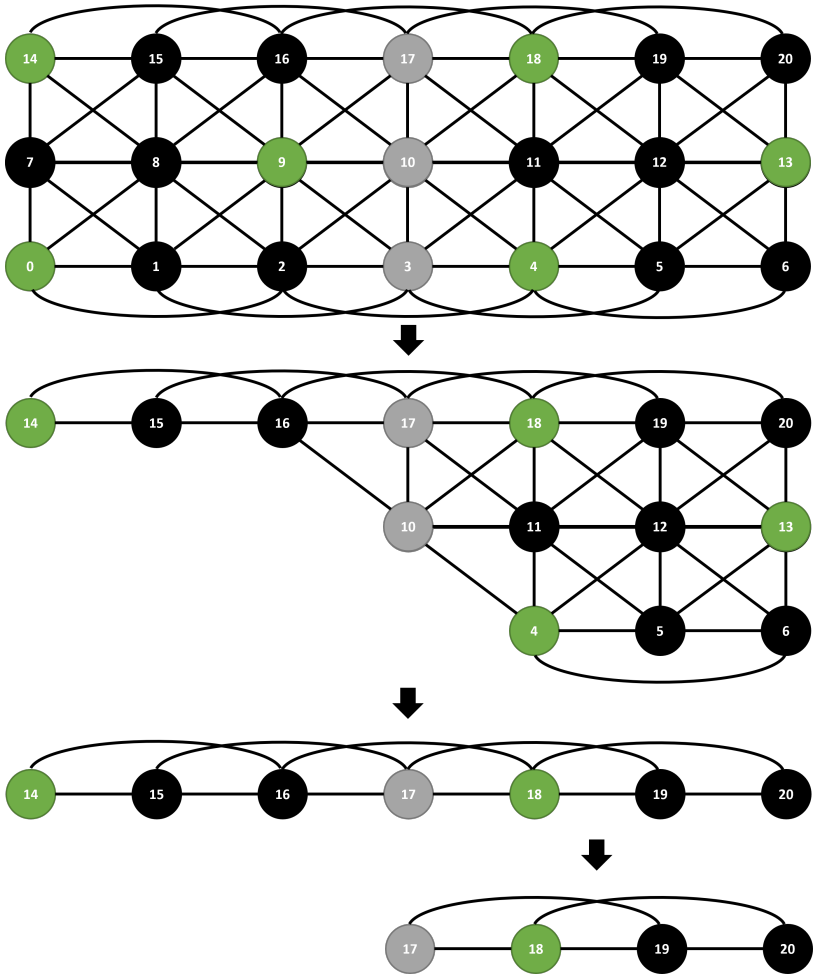


Figure 9: Step by step example of SEAT_GREEDY_IS applied to the problem instance shown in Figure 2 with a SF ordering. The green coloured vertices represent the vertices in the IS.

6.3 ILP

According to the greedy nature of the previous two algorithms, the total number of groups to seat is known upfront; however where a group is seated is decided during each step without consideration of future group placements. Unless there exists a proof that either

greedy algorithm can seat each group optimally during each step, there is no guarantee that their greedy solution is also an optimal solution.

To output an optimal solution to the DCCS problem a feasible approach would be to compare all possible combinations of group sizes and starting seats. By implementing the distance constraints using an ILP model and maximizing the number of groups seated, a solver could compare all combinations and find a feasible solution that would also be an optimal seating plan to the DCCS problem. It is important to note that this model uses similar constraints to the ILP model defined by [2]. Instead of using a 2D plane a cinema graph is used to encode the model. This algorithm, referred to as SEAT_ILP from now on, takes a cinema graph G and a multi-set of groups T and encodes an ILP based on the seating paths in G. Once encoded, it is run using a solver and the output is decoded back to a mapping μ of vertices to integers.

$$seat\_ilp : (G, T) \rightarrow \mu : V' \mapsto \{1, 2...8\}, \quad V' \subseteq V(G), O \subseteq V(G) \tag{19}$$

The formal definition of the model is defined in Equation 25 and a description of the model is provided below:

- Let the **decision variable** $x_{v,g}$ shown in Equation 20 denote whether a group of size g is seated at vertex v. To decode the ILP result to a mapping μ, all variables where $x_{v,g} = 1$ are selected and added to the mapping as: $\mu(v) = g$. Initially, in [2] the model considered every group size in T as a possible candidate variable for every seat. By using seating paths the number of variables added to the model is reduced. Take as an example a single row shown in Figure 10: The maximum group sizes that can be seated at vertices $\{0, 3\}$ are at most two and four respectively. A group of size five would not be able to fit in this row, therefore adding a variable of this size for every valid vertex in this row is unnecessary. Furthermore, consider the vertex at index 6. Adding a variable for a group of size four is also unnecessary, because that is already covered by the variable $x_{3,4}$.

$$x_{v,g} = \begin{cases} 1 & \text{if a group of size } g \text{ is seated at vertex } v \\ 0 & \text{else} \end{cases} \tag{20}$$

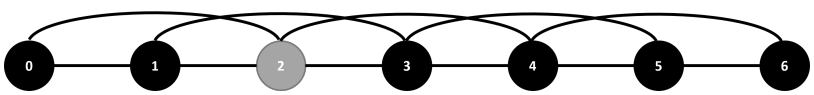


Figure 10: Example of a cinema graph with a single row.

- To simplify the encoding process, four **helpers** are defined. The first helper S is a set of vertices that denotes all the valid starting seats in a cinema where a group can be seated:

$$S = [v \mid v \in V(G), label(v) = e] \tag{21}$$

The second helper is a set $S'_{v,g}$ that contains vertices which represent the invalid starting seats if a group of size g were to be seated at vertex v. For example, given the cinema graph shown in Figure 11, if a group of size two were to be seated at vertex 0 then $S'_{0,2} = \{3, 7, 8, 9\}$:

$$S'_{v,g} = neighbours(spath(v, \langle\rangle, \oplus, g)) \cap S \tag{22}$$

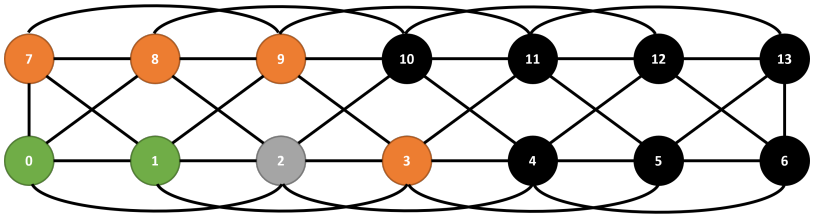


Figure 11: Example of invalid vertices (orange) if a group of size 2 is seated an vertex 0 (green).

The third helper T_v is a multi set that contains group sizes that can be seated at v based on the maximum size of its seating path. For example, given a cinema graph as depicted in Figure 10 and a multi set $T = [8, 5, 4, 4, 3, 2, 1]$ of groups to seat. Then $T_3 = [4, 4, 3, 2, 1]$ because the maximum group size that can be seated at vertex 3 is four, so all group sizes in T that are less than or equal to four are included in T_3. The formal definition of this helper is defined below:

$$T_v = [g \mid g \in T, g \leq |spath(v, \langle\rangle, \oplus, 8)|] \tag{23}$$

The last helper X_g denotes all decision variables for a given group index:

$$X_g = [x_{v,g'} \mid v \in S, g' \in T_g, g' = g] \tag{24}$$

- The **objective** of the model is to maximize the number of groups that can be seated. To ensure that groups of larger size are factored in appropriately, each variable is assigned its group size as a weighting coefficient.

- Lastly, to ensure the solution of the ILP is a valid solution to the cinema problem the following **constraint groupings** are added to the model:

Firstly, the *Only one group per seat* constraint grouping ensures that a starting seat can only seat one group, where for each valid vertex v in S, a constraint is added to the model that ensures that at most only one group from the multi set T_v can be seated.

Secondly, the *Distance* constraint grouping ensures that two seated groups do not violate each other's distance constraints, where for each combination of valid vertex v in S and group sizes in T_v a constraint is added to the model that ensures no other group regardless of its size can be seated at any vertex in $S'_{v,g}$.

Thirdly, the *Maximum Group Count* constraint grouping ensures that the maximum group count for a given size is not exceeded. This is achieved by ensuring that the total sum of decision variables for a group size denoted by X_g does not exceed its length $|X_g|$.

Finally, the *Binary* constraint grouping ensures that a group is either seated at a seat or is not.

$$
\begin{aligned}
&\max_{x_{v,g}} \quad \sum_{v \in S} \sum_{g \in T_v} x_{v,g} \times g && && \textit{(Objective)} \\
&\text{s.t.} \\
&\qquad \sum_{g \in T_v} x_{v,g} \leq 1, && \forall v \in S && \textit{(Only one group per seat)} \\
&\qquad x_{v,g} + x_{v',g'} \leq 1, && \begin{array}{l} \forall v \in S, \forall g \in T_v, \\ \forall v' \in S'_{v,g}, \forall g' \in T_{v'} \end{array} && \textit{(Distance)} \\
&\qquad \sum_{x \in X_g} x \leq |X_g|, && \forall g \in T && \textit{(Maximum Group Count)} \\
&\qquad x_{v,g} \in \{0,1\}, && \forall v \in S, \forall g \in T_v && \textit{(Binary)}
\end{aligned}
\tag{25}
$$

6.4 ILP with IS

In this sub-section, a heuristic is given which can be utilized in SEAT_ILP to improve its run time. This is achieved by further reducing the number of decision variables added to the ILP model. Instead of considering all valid vertices in G as possible starting seat candidates, this heuristic only considers vertices that are part of the IS as starting seat candidates. To use this heuristic the following changes are made to ILP:

- Firstly, the IS denoted by M is required as input. These vertices do not need to be ordered as the ILP considers all combinations regardless in what order they were added to the model.

$$seat\_ilp\_is : (G, T, M) \rightarrow \mu : V' \mapsto \{1, 2...8\}, \quad V' \subseteq V(G), M \subseteq V(G) \quad (26)$$

- Secondly, the helpers $S'_{v,g}$ and S are also adjusted. $S'_{v,g}$ only returns invalid vertices in M as the *Distance* constraints only apply to other starting seats and the set of valid starting seats S is equal to the IS: $S = M$

7 Experimentation

Several experiments were conducted as part of this research in order to measure how efficient these seating algorithms are at seating groups in a cinema. This chapter describes some key implementation details of the seating algorithms, explains the experiment setup, and visually presents the collected data.

7.1 Implementation

Each experiment uses two separate programs[1]. The first program constructs a cinema graph from a given problem instance using the construction algorithm defined in Section 4 and then finds an IS of the cinema graph using the first two algorithms described in Section 5. The second program uses the seating algorithms defined in Section 6 to seat groups according to the given cinema graph. Additionally, to have an external benchmark, the two seating algorithms: SEAT_GREEDY_MADS and SEAT_ILP_MADS described in [2] are also implemented in the second program.

The reason the implementation is split into two programs is because the cinema graph is static and only needs to be generated once, whereas the groups to be seated are dynamic and change for each movie showing. The first program writes the cinema graph and IS to a file and the second program can repeatedly re-use this file to seat groups. This split is advantageous for larger cinemas, where finding a IS could take a long time. The specification used by both programs to read and write the cinema graph is called GraphML[2]. This XML-based specification is useful as it can easily be parsed and supports custom attribute data. For example, this allows the degree and seating path to be calculated during construction and stored with its vertex. An example of the GraphML specification of a cinema graph is shown in Appendix 8.1.

Lastly, the Gurobi[3] library is used by both programs to solve their respective ILP models.

7.2 Setup

The experiment is setup to run against two types of problem instances: Random and Exact. To produce enough data to achieve sufficient statistical significance many problem instances are required. However, creating these individually by hand is not feasible and instead instances are pseudo randomly generated. The following parameters are used for generating random cinema instances:

- **Size** is the total number of empty seats and spaces in the cinema combined. The

[1] https://github.com/maxstoll94/master-thesis-phase-2/tree/main/Program
[2] http://graphml.graphdrawing.org/
[3] https://www.gurobi.com/

following sizes are used to measure the impact of cinema size on the seating algorithms:

$$\{30, 40, \ldots, 150, 200, 300, \ldots, 2000, 2250, 2500, 2750, 3000\}$$

- **Occupancy** is the percentage of seats in the cinema on which people want to be seated. Depending on the movie and time of day, a cinema may vary in its occupancy. To simulate this variance the following parameters are used:

$$\{0.3, 0.5, 0.7, 0.9\}$$

- **Density** is the ratio of seats to space in a cinema. To simulate various densities of a cinema the following parameters are used:

$$\{0.2, 0.4, 0.6, 0.8, 1\}$$

To randomly generate the group sizes for each cinema instance a discrete probability distribution is required. At the time of writing this paper, there is no public statistical data related to group sizes in cinemas from which such a distribution could be derived. Therefore an alternate approach is taken in which three Pathé cinemas in Amsterdam are sampled for a total of 15 movie showings equally distributed over the following genres: *House of Gucci* (Drama), *Encanto* (Family) and *Ghostbusters Legacy* (Action). For each showing the current group sizes are counted and are summarised in Table 4. If a group size exceeds the maximum size, they are considered as two separate households. For example, a group of size ten is counted towards both eight and two. Based on this data, the discrete probability distribution shown in Table 5 is generated. Checks are added to the group generator to ensure it only generates group sizes that could be seated, never generates more people than empty seats in the cinema and never generates groups larger than the number of sequential empty seats. For each cinema configuration of size, occupancy, and density a cinema problem instance is independently sampled five times. This results in a total of: $4 \times 5 \times 36 \times 5 = 3600$ generated problem instances[4].

[4]https://github.com/maxstoll94/master-thesis-phase-2/tree/main/Experiments/Random/Instances

Group Size	Count
1	34
2	107
3	17
4	9
5	3
6	1
7	0
8	3
Total	176

Table 4: Table showing the total count of collected group sizes

Group Size	Probability
1	0.20
2	0.61
3	0.10
4	0.05
5	0.02
6	0.01
7	0.00
8	0.02
Total	1

Table 5: Table showing the resulting distribution based on the total count.

Even though these generated instances attempt to simulate real cinemas and their demand, it is also useful to compare the seating algorithms to the real-world seating arrangements of cinemas. For this reason, the seating algorithms are also run against the following five cinemas in the Netherlands[5]:

1. Pathé Arena, Zaal 2 [6]
2. Pathé Maastricht, Zaal 2[7]
3. Pathé Spuimarkt, Zaal 2[8]
4. Pathé Tilburg, Zaal 4[9]
5. Pathé Ede, Zaal 9[10]

Each cinema is then filled up to the following occupancy levels: $\{0.3, 0.5, 0.7, 0.9\}$ with group sizes sampled from the distribution shown in Figure 5.

7.3 Results

The presented data in this section is the result of running the random and exact instances using the construction and solving programs on the following hardware:

[5]https://github.com/maxstoll94/master-thesis-phase-2/tree/main/Experiments/Exact/Instances
[6]https://www.pathe.nl/bioscoop/arena
[7]https://www.pathe.nl/bioscoop/maastricht
[8]https://www.pathe.nl/bioscoop/spuimarkt
[9]https://www.pathe.nl/bioscoop/tilburg
[10]https://www.pathe.nl/bioscoop/ede

- **CPU**: AMD Ryzen 5 3600 (3.56 Ghz) 6-Core Processor
- **RAM**: 16 GB
- **Operating System**: 64-bit Windows 11 Pro

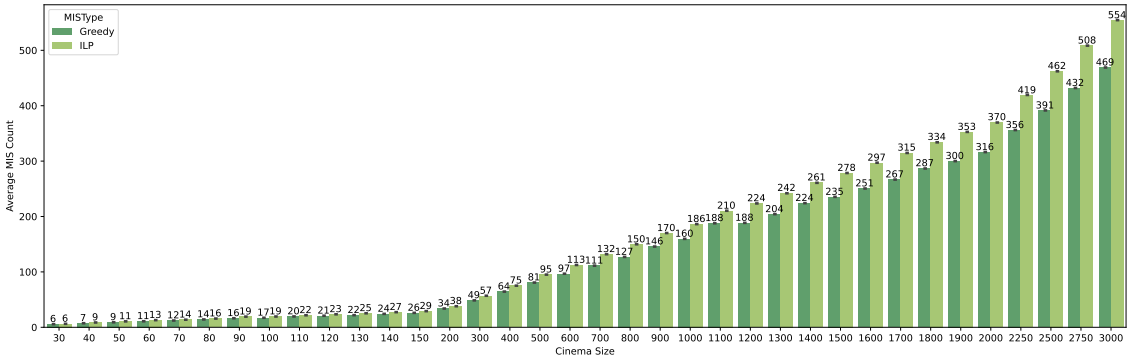
7.3.1 Construction Results

The first plot in Figure 12 shows a summary of the construction results over the random instances. Here the first Figure 12a shows a bar plot that compares the average IS count produced by the IS_GREEDY and IS_ILP against cinema sizes. As theorized, the bars show that for almost all cinema sizes IS_ILP produces on average a larger IS than IS_GREEDY, but IS_GREEDY on average always remains within a factor of 4 of IS_ILP for all cinema sizes.

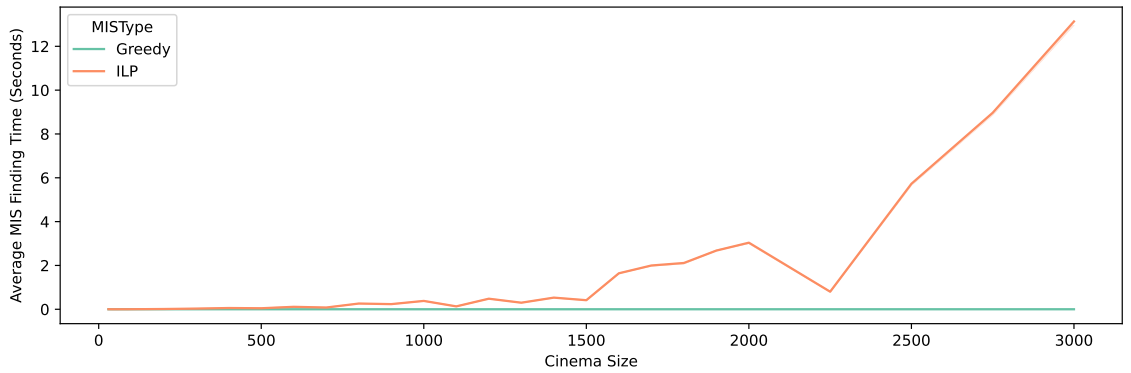
$$\frac{|\text{IS\_ILP}|}{|\text{IS\_GREEDY}|} \leq 4$$

The larger IS comes at a cost as shown in Figure 12b. IS_ILP's average finding time has an exponential correlation to the cinema size, whereas IS_GREEDY remains constant to the cinema size.

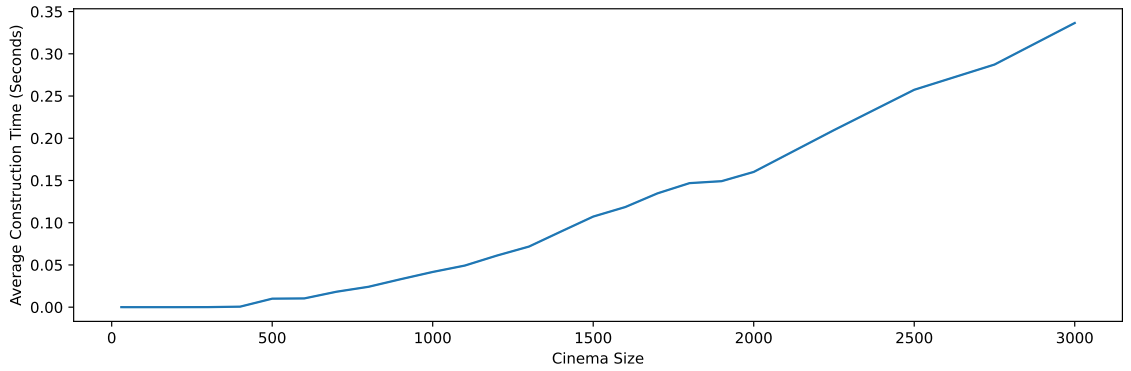
The third Figure 12c shows a positive correlation between construction time and cinema size.



(a) MIS Count for Greedy and ILP



(b) MIS Finding Time



(c) Construction Time

Figure 12: Summary of Construction Data obtained from Random Cinema Instances.

7.3.2 Solving Results

The seating algorithms are evaluated on three dimensions: PCS, PPS and solving time.

1. The PCS is defined as the ratio of people seated by the algorithm to empty seats in the cinema. For example, the problem instance depicted in Figure 2 has twelve available seats, if an algorithm A were to seat five (3 + 2) of the people and algorithm B would only seat 3, then their resulting PCS would be 41.6% and 25% respectively. Thus, algorithm A occupies 16.6% more of the cinema than algorithm B does.

2. The PPS is defined as the ratio of people seated by the algorithm to the people expected to be seated. Using the same example as before, the problem instance has seven people expected to be seated and therefore algorithm A seats 71.4% of the expected people which is 30% more than algorithm B.

3. If two algorithms share a similar PCS and PPS, they can also be compared on solving time. This is the time, in seconds, the algorithm takes to only seat the people, excluding reading of the cinema graph and writing of the results.

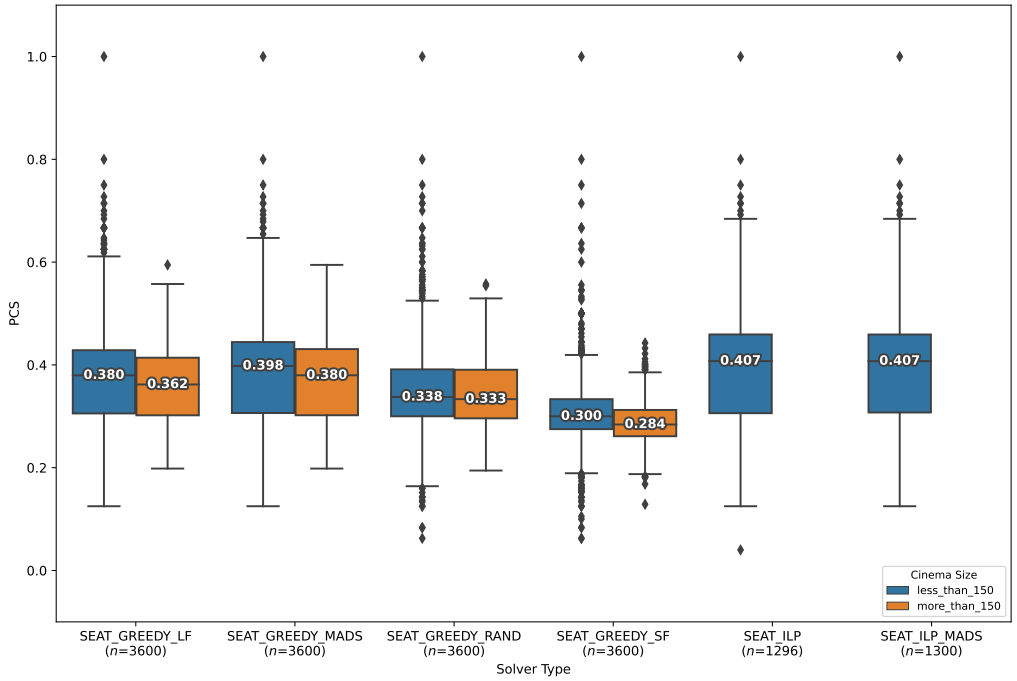
It is also important to mention that even though the problem specifies groups, the algorithms are evaluated on how many people they can seat. This is because cinemas usually charge a ticket per person and an algorithm that seats many small groups might sell less tickets than an algorithm that seats fewer large groups. By counting the number of people seated instead, this problem is avoided.

7.3.3 Random

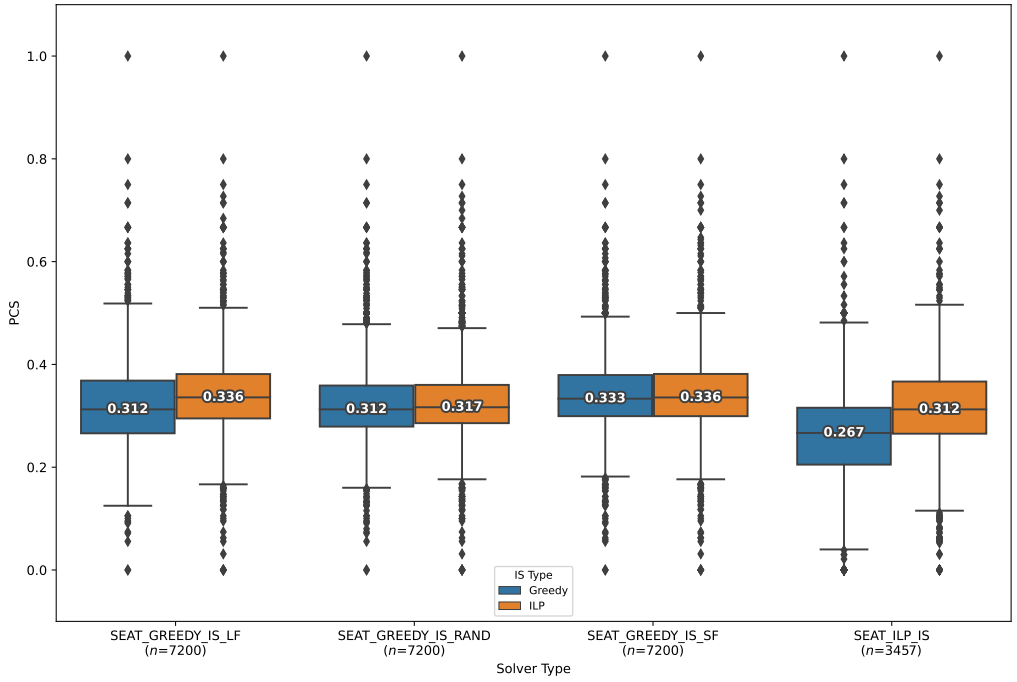
The following questions are answered by visualizing the sampled data collected from running the seating algorithms against the randomly generated cinema instances:

1. **How effectively do the algorithms seat people compared to each other?**

   To compare the seating algorithms performance to each other the first step is to get an overview of the sampled data. Figure 13 summarizes the distributions on PCS across all seating algorithms using two box plots. The first box plot shown in Figure 13a compares the algorithms that do not use an IS. Comparing the samples shows two interesting observations: Firstly, the length of all whiskers are approximately symmetric indicating that these samples are normally distributed. Secondly, as expected both SEAT_ILP and SEAT_ILP_MADS share a similar distribution and the highest median with 40.7%. However, SEAT_ILP solved four less instances as indicated by the sample size n due to time outs. On the topic of sample size, both ILP algorithms repeatedly timed out the instances larger than 150 so there is no data available beyond this point. Therefore on instances of size larger than 150, SEAT_GREEDY_MADS has the highest sampled median of 38%.



(a) Without MIS



(b) With MIS

Figure 13: Boxplot comparing percentage seated and showing the effect of MIS type on percentage seated.

The second Figure 13b compares the distributions of the seating algorithms that use the IS. Here the sampled data indicates two observations about the distribution. Firstly, similar to the first box plot, the symmetrical whiskers indicate that these samples are normally distributed. Secondly, the bottom outliers indicate that there

are instances where algorithms using the IS are unable to seat any people at all. Inspecting these instances uncovers a short coming in these algorithms on sparse cinemas as shown in Figure 14. For this instance the groups of size two can never be seated as the starting vertices where it can be seated are not included in a MIS.

```
3
5
10000
11010
10000
0,1,0,0,0,0,0,0
```

Figure 14: Example of instance on which SEAT_GREEDY_IS and SEAT_ILP_IS cannot seat any people.

Given that both box plots indicate the sampled data to be normally distributed, Table 6 shows the 95% confidence intervals of average PCS for each seating algorithm. This data indicates that an optimal solution to the DCCS problem occupies on average between 39.5% and 40.5% of the seats (for cinemas with at most 150 seats). In comparison, the heuristic for SEAT_ILP has the lowest average PCS and only occupies between 28.2% to 28/8% of the seats in a cinema.

Solver Type	Mean
SEAT_ILP, SEAT_ILP_MADS	[0.395, 0.405]
SEAT_GREEDY_MADS	[0.381, 0.386]
SEAT_GREEDY_LF	[0.369, 0.374]
SEAT_GREEDY_IS_SF	[0.339, 0.342]
SEAT_GREEDY_IS_LF	[0.326, 0.329]
SEAT_GREEDY_RAND	[0.323, 0.327]
SEAT_GREEDY_IS_RAND	[0.321, 0.324]
SEAT_GREEDY_SF	[0.292, 0.295]
SEAT_ILP_IS	[0.282, 0.288]

Table 6: Table showing the 95% confidence internal of average PCS for each seating algorithm. The table is ordered by the lower bound of the confidence interval.

2. **Does the size of the IS affect how effectively the algorithms can seat people?**

   Figure 13b shows that the seating algorithms seem to benefit from a larger IS and their sampled mean of MIS: (95% CI [0.330, 0.332]) is slightly higher than greedy IS: (95% CI [0.314, 0.317]) counterpart. To provide support for this hypothesis,

a two sampled one sided t-test is conducted where the null hypothesis (H_0) states: There is no statistical difference between the average PCS of a seating algorithm using the MIS and the IS and the alternate hypothesis (H_1) states: The MIS has a higher average PCS than the IS. The test rejects H_0 for SEAT_GREEDY_IS_LF ($p < .001$), SEAT_GREEDY_IS_RAND ($p < .001$) and SEAT_ILP_IS ($p < .001$), however fails to reject it for SEAT_GREEDY_IS_SF ($p > .228$).

3. **Does the order in which group are seated impact how effectively the greedy algorithms can seat people?**

   Looking at the sample distributions shown in Figure 13a already indicates that SEAT_GREEDY_LF has the highest median PCS. Comparing the sample means of the algorithms in Table 6: SEAT_GREEDY_LF, SEAT_GREEDY_RAND and SEAT_GREEDY_SF show no overlap in confidence intervals. This means that on average, seating the largest groups first tends to result in more people being seated compared to the other two orders.

4. **Does the order in which the cinema is traversed impact how effectively the greedy algorithms that uses the IS seat people?**

   Comparing the sample PCS means of these algorithms: SEAT_GREEDY_IS_LF, SEAT_GREEDY_IS_RAND and SEAT_GREEDY_IS_SF shows a confidence interval overlap between LargestFirst (LF) and SmallestFirst (SF). This means there is no clear advantage of starting to seat from either corner of the cinema but both are more advantageous on average than randomly iterating through the cinema.

5. **What is the variance on seating algorithms that use randomness?**

   To answer this question, the algorithms SEAT_GREEDY_RAND and SEAT_GREEDY_IS_RAND are run 50 times against the same random instances with different seeds and the average PCS is recorded for each run. The data shows that the difference in average PCS between runs is at most 0.2% and 0.13% for SEAT_GREEDY_RAND and SEAT_GREEDY_IS_RAND respectively.

6. **How do the seating constraints impact the revenue of cinema?**

   To provide a real-world context on the impact the distance constraints has on revenue Figure 15 shows the percentage of the total average revenue lost of each algorithm compared to cinema size. The darker areas of the heat map indicate a higher percentage of revenue lost. Comparing the coloured areas shows two interesting properties about the seating algorithms. Firstly, SEAT_ILP_IS has on average the highest percentage loss of all seating algorithms. Oppositely, SEAT_ILP, SEAT_ILP_MADS and SEAT_GREEDY_MADS have the lowest average percent-

age loss. Secondly, the cinema size on average has a small impact on the revenue as the areas only get slightly lighter with increasing cinema size.

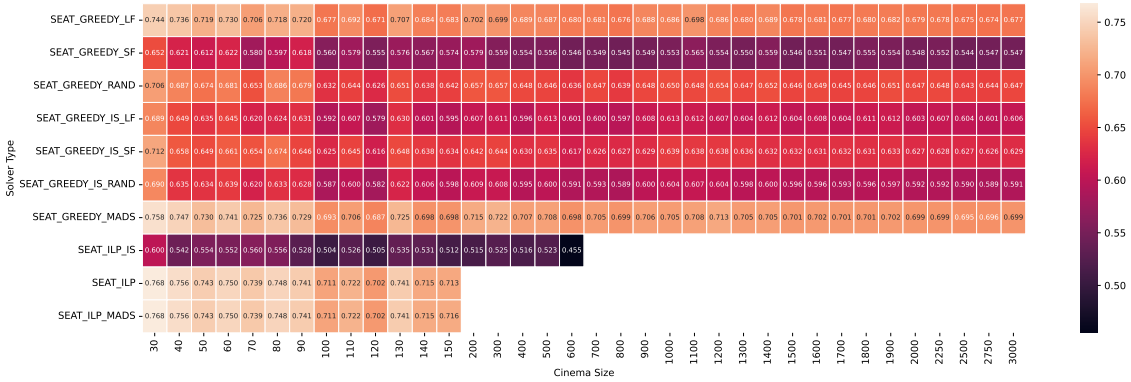


Figure 15: Heat map showing the percentage of the average potential revenue loss for each seating algorithm compared to cinema size.

7. **Does the density and size of a cinema impact how effectively the algorithms can seat people?**

   Looking at the density first, Figure 16 shows a line plot for each seating algorithm comparing the average PCS to cinema size across three density types:

   - Sparse (20 - 40% of cinema size is seats)
   - Dense (40 - 80% of cinema size is seats)
   - Very Dense (80 - 100% of cinema size is seats)

   Comparing the line plots shows five interesting observations:

   (a) SEAT_GREEDY have a similar seating performance on sparse and dense instances.

   (b) SEAT_GREEDY_IS, SEAT_GREEDY_MADS, SEAT_ILP and SEAT_ILP_MADS show to have on average a higher PCS on dense instances.

   (c) SEAT_ILP_IS has on average its lowest PCS on sparse cinemas.

   (d) All greedy algorithms have the lowest average PCS on very dense cinemas.

   (e) There is little variation in average PCS with increasing cinema size for most algorithms. This is indicated by the lines flattening as the cinema size increases. The shaded areas show huge spikes in variance on small instances, but as the cinema size increases so does the variance. The exception to this observation is SEAT_ILP_IS that shows a drop in average PCS for cinema sizes 400 and 500 due to missing data caused by timeouts.

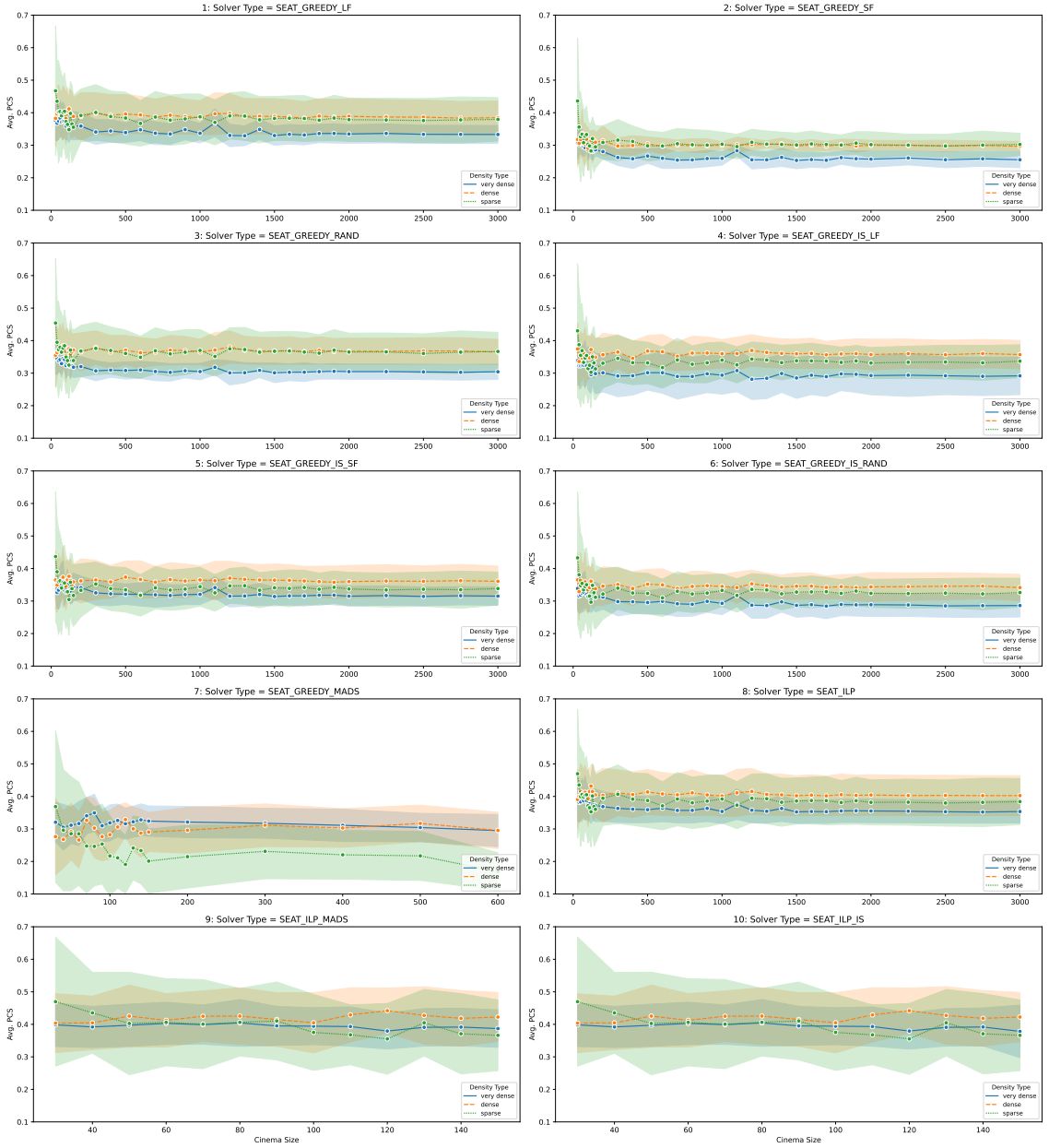
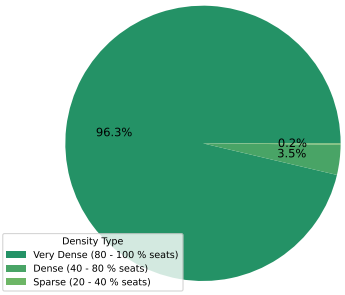
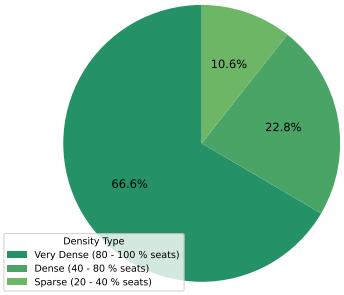


Figure 16: Line plots showing the effect of density and cinema size on the average PCS for each seating algorithm.

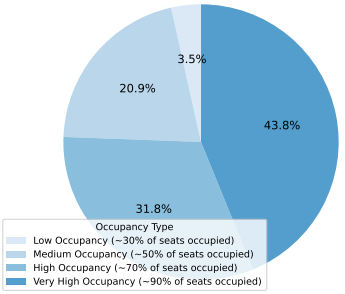
Then looking at the solving time, Figure 17a shows a pie plot dissecting the percentage of time the ILP algorithms spent solving cinema instances categorized by density. The plot clearly indicates that they spend substantially more time solving very dense cinema instances compared to the other two density types. A similar result is also observed for the greedy algorithms and is shown in Figure 17b.



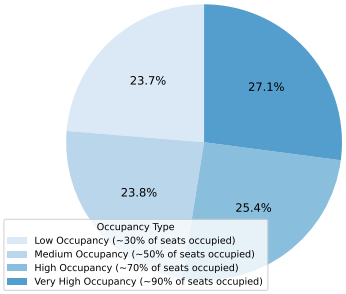
(a) Pie plot comparing the percentage of time spent by ILP algorithms on solving cinemas categorized by density.



(b) Pie plot comparing the percentage of time spent by greedy algorithms on solving cinemas categorized by density.



(c) Pie plot comparing the percentage of time spent by ILP algorithms on solving cinemas categorized by occupancy.



(d) Pie plot comparing the percentage of time spent by greedy algorithms on solving cinemas categorized by occupancy.

Figure 17: Summary showing the impact of density and occupancy on the solving time of the seating algorithms.

8. **Does the occupancy of a cinema impact how effectively the algorithms can seat people?**

   Unlike the previous question, the PCS is not a useful indicator as it does not factor in the people expected to be seated. So instead Figure 18 shows the average PPS categorized by these three occupancy types:

   - Low Occupancy (~30% of the cinema capacity is occupied)
   - Medium Occupancy (~50% of the cinema capacity is occupied)
   - High Occupancy (~70% of the cinema capacity is occupied)
   - Very High Occupancy (~90% of the cinema capacity is occupied)

   The bars show that if a cinema has a low occupancy most algorithms on average can seat more than 90% of these people. However, this percentage decreases as the

occupancy increases. For example, the algorithms can only seat on average between 34 to 51% of the people for cinemas with a very high occupancy.

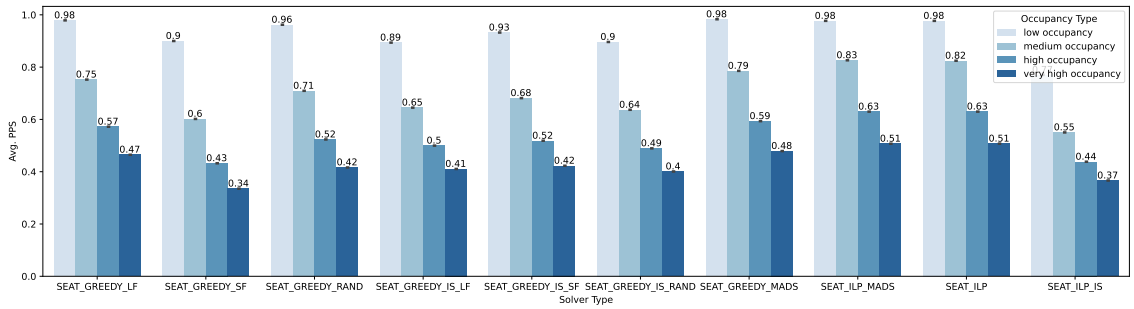
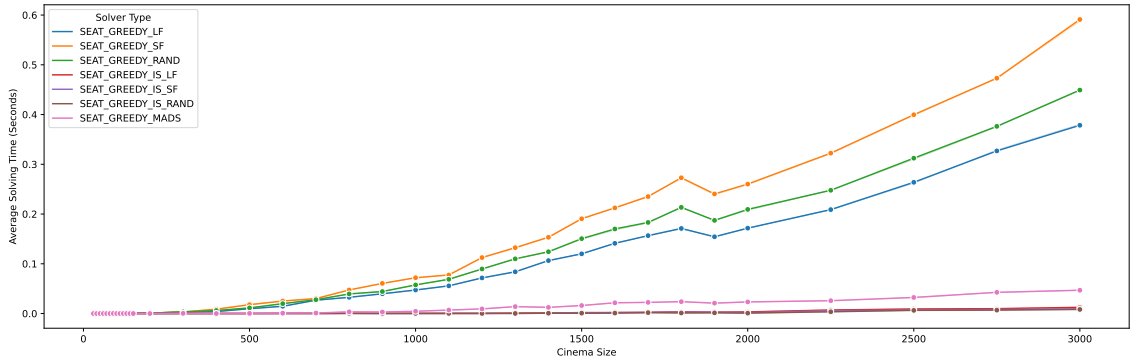


Figure 18: Bar plot showing the average PPS for each seating algorithm categorized by occupancy.

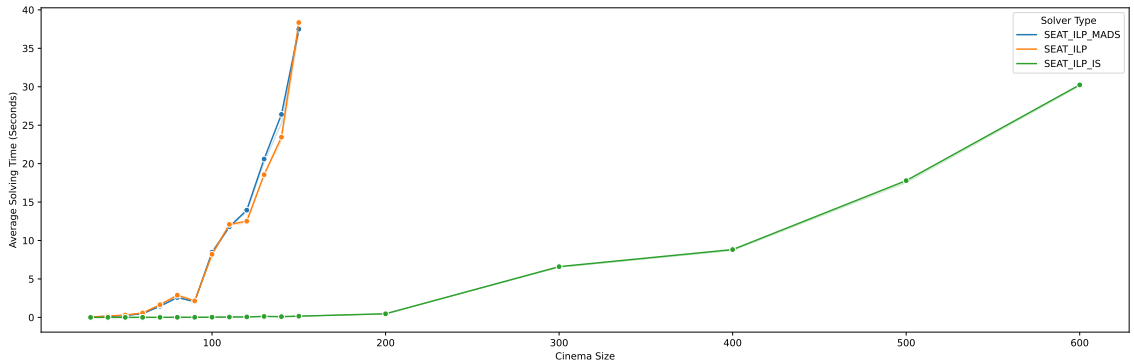
Figure 17c shows how the occupancy impacts the ILP algorithms, as they spent the largest percentage of their time on very highly occupied cinemas and the least on low occupied cinemas. However, the difference in percentages between each occupancy category is not as extreme compared to the differences between the density categories. The greedy algorithms share a similar observation as shown in Figure 17d.

9. **Does the cinema size impact the solving time of the algorithms?**

   To determine the impact of cinema size on the solving time of the seating algorithms, two line plots are shown in Figure 19. The first line plot 19a shows that on average all greedy algorithms run in polynomial time to the cinema size and can solve very large instances in less than a second. The second line plot 19a shows that the average solving time of the ILP algorithm scales exponentially to the cinema size. For SEAT_ILP and SEAT_ILP_MADS an increase in average solving time is already observed at a size of 50, whereas for SEAT_ILP_IS a noticeable increase only occurs at 200.



(a) Categorized by Greedy

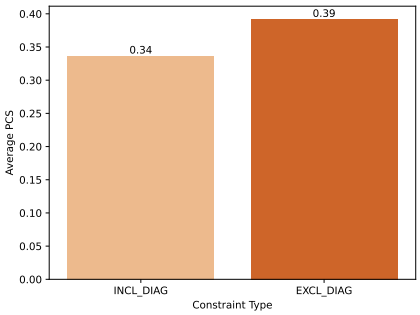


(b) Categorized by ILP

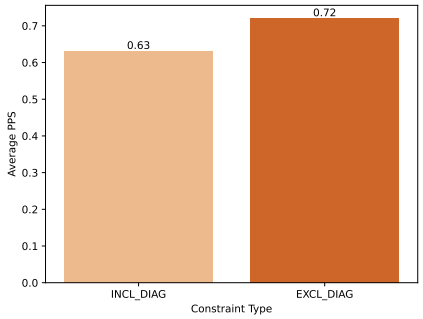
Figure 19: Two line plots showing the effect of cinema size on the average solving time categorized by algorithm type.

10. **Does removing the diagonal distance constraint impact how effectively the algorithms can seat people?**

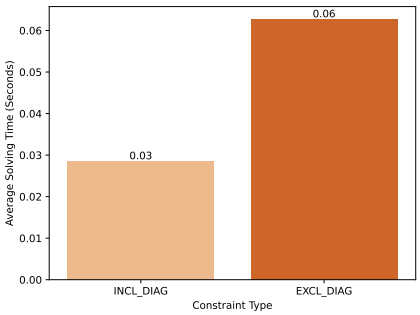
    The same instances where re-run using cinema graphs constructed without the diagonal distance constraint in place. The two data sets are compared to each other on their average PCS, PPS and solving time using plots shown in Figure 20. The first two plots 20a and 20b indicate that through the removal of the diagonal constraint, the seating algorithms on average occupy 5% more of the seats and can on average seat 9% more people. However, inspecting the average solving time in the third plot 20c shows that the greedy seating algorithms take on average twice as long to solve problem instances without diagonal constraints than with. A similar observation is made in the fourth plot 20d where removing the diagonal constraint shows a significant increase of 23% and 44% in timeouts for SEAT_ILP_IS and SEAT_ILP respectively.



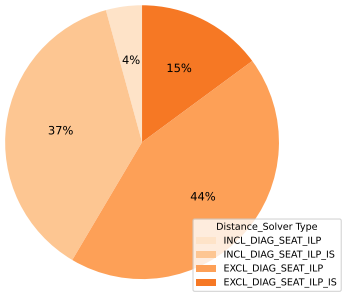
(a) Bar plot showing the difference in average PCS between cinemas with different distance constraints.



(b) Bar plot showing the difference in average PPS between cinemas with different distance constraints.



(c) Bar plot showing the difference in average solving time between cinemas with different distance constraints.

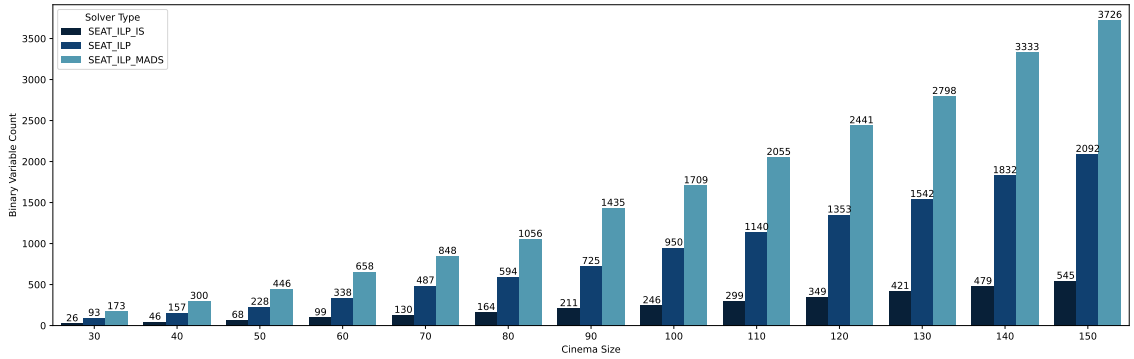


(d) Pie plot showing the difference in number of timeouts between cinemas with different distance constraints.

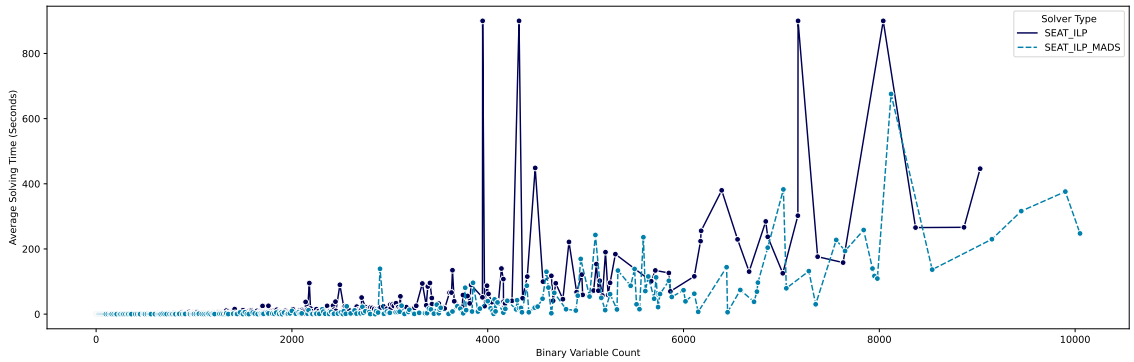
Figure 20: Summary showing the impact of removing the diagonal distance constraints.

11. **What is the difference in the number of decision variables used by the ILP algorithms?**

    As mentioned in Section 6.3, SEAT_ILP and SEAT_ILP_MADS both model similar constraints. By calculating the seating paths in advance, it is possible to determine the maximum group size that can fit at any given seat. By filtering out the larger group sizes, the number of binary variables required by SEAT_ILP is reduced compared to SEAT_ILP_MADS. SEAT_ILP_IS then reduces the number of variables further, by only considering the IS as starting seats. These expectations are also reflected by the data as shown in Figure 21a. Comparing the number of binary variables to solving time in Figure 21b, shows that despite defining fewer variables, the spikes in average solving time for SEAT_ILP's are much higher than SEAT_ILP_MADS.



(a) Bar plot showing the average number of binary variables compared to cinema size.



(b) Line plot showing the average number of binary variables compared to solving time.

Figure 21: Two plots summarizing the impact of the number of binary variables for ILP algorithms.

12. **Does a width and height of a cinema impact how effectively the algorithms can seat people?**

    The random instances were generated with various widths and heights in mind. If the width of cinema exceeds its height, it is categorized as a *Wide* layout. Oppositely, if its height exceeds its width, it is categorized as a *Narrow* layout. Lastly, if its height and width are equal then it is categorized as a *Square* layout. Figure 22 shows a bar plot illustrating the average PCS for all seating algorithms categorized by layout type. Comparing the bars shows that on average most algorithms have a higher PCS on wider cinemas compared to the other layouts. However, the black bars represent the 95% confidence intervals for the average PCS. Comparing these bars shows overlap, which means there is little support that the impact a cinema layout has on the number of people seated is statistically significant.

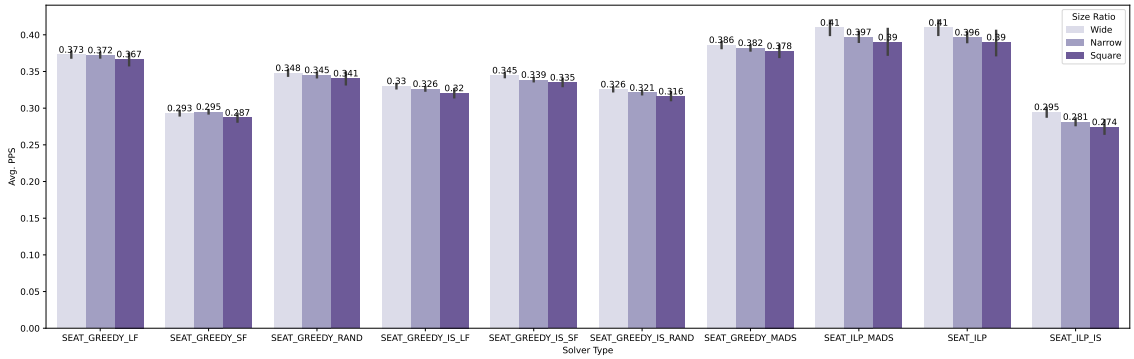


Figure 22: Bar plot showing the average PCS for each seating algorithm categorized by cinema layout.

7.3.4 Exact

The experiment on the exact instances is conducted on the same hardware configuration as the previous experiment and the results are summarized in Figure 23 using stacked bar plots. Comparing the sampled PPS of the algorithms at different occupancy levels shows similar results to the previous experiment on the random instances. For smaller cinemas such as Arena or Maastricht an optimal seating plan is guaranteed regardless of its occupancy. On slightly larger cinemas such as Spuimarkt and Tilburg, an optimal seating plan is only guaranteed if the cinema has a low occupancy. If the cinema is very large such as the one located in Ede, an optimal seating plan is not guaranteed due to memory issues.

Figure 23: Stacked box plots showing the total PPS split over four different occupancy type.

8 Conclusion

As a consequence of the COVID-19 virus, seating groups of different households, while ensuring 1.5 meter distance constraints between them, became a logistical problem for many cinemas. Given the layout of seats in a cinema, a graph is a natural data abstraction where the seats are modelled as vertices and the distance constraints between the seats are modelled as edges. The IS on this graph then could represent the possible starting locations to seat groups from. Using these two ideas, three algorithms are presented as solutions to the DCCS problem: Two deterministic greedy algorithms and one non-deterministic ILP algorithm with heuristic.

Then two types of experiments are conducted as part of this research to compare how effective these algorithms are at seating groups. In the first experiment the algorithms are run against randomly generated cinema instances of various densities and occupancy and the group sizes are sampled from a uniform distribution based on real world data. Visualizing the collected data using various plots provides support for the following conclusions:

1. On average for cinemas of up to 1500 seats, a graph and MIS can be constructed and found in under one second.
2. The average count of the greedy IS remains within an approximation factor of 4 to the optimal count.
3. The ILP algorithm seats more people on average than all other algorithms, but runs into timeout and memory issues for cinemas with more than 150 seats. For SEAT_ILP_IS this cutoff point is at 600 seats. For any cinema with more seats, the greedy algorithms remain the only options. Where SEAT_GREEDY_MADS on average seats more people than the greedy algorithms introduced in this paper.
4. The heuristic for SEAT_ILP improves the solving time but seats 12% less people on average.
5. On average, a larger IS results in more people being seated.
6. Seating largest groups first seats more people are seated on average.
7. Iterating the cinema from bottom left to top right or vice versa has little impact on the average number of people seated.
8. Randomly selecting group sizes or seats shows to have little variance between each seating run.

9. Depending on the cinema size and algorithm a cinema owner loses between 24% to 55% of their revenue through distance constrained seating and the smaller cinemas lose less revenue than larger ones.

10. On average, less people are seated on very dense cinemas and the ILP algorithms also take significantly longer to solve them.

11. The cinema size does not impact the average number of people most algorithms can seat with exception of SEAT_ILP_IS.

12. If a cinema has a low occupancy, then on average the algorithms can seat most of the attending people. However, as the occupancy increases this average drops significantly.

13. There is a positive correlation between cinema size and average solving time. Where SEAT_GREEDY and SEAT_GREEDY_IS can seat people in cinemas with 3000 seats in under one second. On the other hand, SEAT_ILP and SEAT_ILP_IS take significantly longer to solve cinemas and run into time out issues for cinemas beyond 150 and 600 seats respectively.

14. Removing the diagonal seating constraints increases the average number of people that can be seated by the algorithms. However, it also results in an increase in average solving time for greedy algorithms and for SEAT_ILP and SEAT_ILP_IS a significant increase in timeouts.

15. As theorized, by using seating paths the average number of binary variables used by SEAT_ILP is reduced by around 50% compared to SEAT_ILP_MADS. The expected outcome is that this would reduce the solving time of SEAT_ILP. Instead, the data indicates the opposite. Not only does SEAT_ILP have a longer average solving time, but also has more timeouts than SEAT_ILP_MADS. The data also shows that for the same number of binary variables SEAT_ILP has significantly higher spikes in average solving time than SEAT_ILP_MADS does. The exact reason as to why SEAT_ILP solves slower on average than SEAT_ILP_MADS despite having less variables is currently still unknown.

16. On average, the layout type of a cinema has little impact on the number of people seated.

The second experiment involved running the algorithms against five problem instances based on real cinema layouts in the Netherlands. The data shows that on smaller cinemas an optimal seating plan is guaranteed regardless of its occupancy. On slightly larger cinemas, an optimal seating plan is only guaranteed if the cinema has a low occupancy. If the cinema is very large an optimal seating plan is not guaranteed due to memory issues.

In the future there are three improvements to consider that due to time constraints could not be included in this thesis. The first improvement occurs during the construction of a cinema graph. At its current implementation the cinema graph models the distance constraints as if only individual people are seated in the cinema. If this were the case, then a MIS would be the optimal seating plan [3]. The distribution of group sizes shows that the most common group size by a large margin is two. It might be beneficial for the algorithms to use cinema graphs based on the distance constraints of their most common group size. The second improvement is based on the removal of the diagonal distance constraints. By moving SEAT_ILP to use a less constrained model its average solving time substantially increased. So moving SEAT_ILP to use a more constrained model could potentially decrease its average solving time. One way to do this could be to increase the horizontal distance constraints to three seats instead of two. The last suggestion is an improvement to the way the groups are generated for the random instances. Currently the distribution is based on data that was sampled during this pandemic. Given that during a pandemic it is recommended to avoid large groups this creates a bias towards smaller groups. This is also reflected in the distribution. To reduce this bias it would be recommended to sample data from a pre-pandemic era.

In conclusion the data presented in this research shows that the construction algorithm can efficiently generate cinema graphs and find the MIS for any real world cinema size. By using the DOT file structure, larger cinema graphs can be generated once, stored, and re-used later on. Practically speaking, this means that approximation algorithms or more efficient optimal algorithms such as IS_FPT presented in this paper are not required. The seating algorithms introduced in this paper could not reach the same seating performance as SEAT_GREEDY_MADS defined by [2]. SEAT_GREEDY_IS does require less solving time on average, but if SEAT_GREEDY_MADS already takes less than a second to solve very large cinema instances then a decrease in solving time at the cost of seating less people is not ideal. Looking at the algorithms holistically, the overall conclusion is the following: For small or sparse cinemas or if solving time is not a factor, the optimal algorithms are the ideal choice, as they can seat the most people and as a result have a lower percentage of revenue loss than all other algorithms. For large or very dense cinemas or when solving time is important the greedy algorithms continue to remain the only choice.

Bibliography

[1] D. Blom, R. Pendavingh, and F. Spieksma, *Filling a theater during the covid-19 pandemic*, INFORMS Journal on Applied Analytics, 2021, ahead of print, issn: 2644-0865. doi: https://doi.org/10.1287/inte.2021.1104. [Accessed Jan. 27, 2022].

[2] A. Langedijk, G. Ogilvie, J. Singh, M. Stoll, and R. Schipperheijn: *The Cinema Seating Problem*, unpublished, 2020.

[3] K. K. Greenberg, T. Hensel, Q. Zhu, S. Aarts, D. B. Shmoys, and S. C. Gutekunst, *An automated tool for optimal classroom seating assignment with social distancing constraints*, in *IISE Annual Conference and Expo 2021*, 2021, 429–434. [Online]. Available: https://pesquisa.bvsalud.org/global-literature-on-novel-coronavirus-2019-ncov/resource/pt/covidwho-1589557 [Accessed Jan. 27, 2022].

[4] J. C. Bortolete, L. F. Bueno, R. B. Butkeraites, *et al.*, *A support tool for planning classrooms considering social distancing between students*, Computational and Applied Mathematics, vol. 41, 2022, Article number 22. doi: https://doi.org/10.1007/s40314-021-01718-w. [Accessed Jan. 27, 2022].

[5] A. T. Murray, *Planning for classroom physical distancing to minimize the threat of COVID-19 disease spread*, PLOS ONE, vol. 15, no. 12, 1–15, 2020. doi: https://doi.org/10.1371/journal.pone.0243345. [Accessed Jan. 27, 2022].

[6] M. Barry, C. Gambella, F. Lorenzi, J. Sheehan, and J. Ploennigs: *Optimal seat allocation under social distancing constraints*, 2021. arXiv: 2105.05017. [Online]. Available: https://arxiv.org/pdf/2105.05017.pdf [Accessed Jan. 27, 2022].

[7] M. Salari, R. J. Milne, C. Delcea, L. Kattan, and L. Cotfas, *Social distancing in airplane seat assignments*, Journal of Air Transport Management, vol. 89, no. 101915, 2020, issn: 0969-6997. doi: https://doi.org/10.1016/j.jairtraman.2020.101915. [Accessed Jan. 27, 2022].

[8] I. Lustig. (2020). "Safe social distancing at sports and entertainment venues: Here is how to make it happen." Blog post by Princeton Consultants Optimization, [Online]. Available: https://princetonoptimization.com/blog/blog/safe-social-distancing-sports-and-entertainment-venues-here-how-make-it-happen [Accessed Jan. 27, 2022].

[9] Jackson. (2020). “Finding a seat: 2 business profs and a grad student built an app to help sports venues with physical distancing.” Blog post by Mendoza College of Business, [Online]. Available: https://mendoza.nd.edu/news/safe-seating/?utm_campaign=COVID-19&utm_content=135854132&utm_medium=social&utm_source=facebook&hss_channel=fbp-139024155342 [Accessed Jan. 27, 2022].

[10] D. Freedman and L. Shepp, *An Unfriendly Seating Arrangement*, SIAM Review, vol. 4, no. 2, 150, 1962. doi: https://doi.org/10.1137/1004037. [Accessed Jan. 23, 2022].

[11] K. Georgiou, E. Kranakis, and D. Krizanc, *Random maximal Independent Sets and the Unfriendly Theater Seating Arrangement problem*, Discrete Mathematics, vol. 309, 5120–5129, 2009. doi: https://doi.org/10.1016/j.disc.2009.03.049. [Accessed Jan. 23, 2022].

[12] B. N. Clark, C. J. Colbourn, and D. S. Johnson, *Unit Disk Graphs*, Discrete Mathematics, vol. 86, no. 1, 165–177, 1990, issn: 0012-365X. doi: https://doi.org/10.1016/0012-365X(90)90358-O. [Accessed Jan. 27, 2022].

[13] M. V. Marathe, H. Breu, H. B. Hunt III, S. S. Ravi, and D. J. Rosenkrantz, *Simple heuristics for Unit Disk Graphs*, Networks, vol. 25, no. 2, 59–68, 1995. doi: https://doi.org/10.1002/net.3230250205. [Accessed Jan. 27, 2022].

[14] G. K. Das, M. De, S. Kolay, S. C. Nandy, and S. Sur-Kolay, *Approximation algorithms for Maximum Independent Set of a Unit Disk Graph*, Information Processing Letters, vol. 115, no. 3, 439–446, 2015, issn: 0020-0190. doi: https://doi.org/10.1016/j.ipl.2014.11.002. [Accessed Jan. 27, 2022].

[15] R. K. Jallu and G. K. Das, *Improved algorithm for Maximum Independent Set on Unit Disk Graph*, in *Algorithms and Discrete Applied Mathematics*, S. Govindarajan and A. Maheshwari, Eds., Springer International Publishing, 2016, 212–223, isbn: 978-3-319-29221-2. doi: https://doi.org/10.1007/978-3-319-29221-2_18. [Accessed Jan. 27, 2022].

[16] S. C. Nandy, S. Pandit, and S. Roy, *Faster approximation for Maximum Independent Set on Unit Disk Graph*, Information Processing Letters, vol. 127, 58–61, 2017, issn: 0020-0190. doi: https://doi.org/10.1016/j.ipl.2017.07.007. [Accessed Jan. 27, 2022].

[17] G. K. Das, G. D. da Fonseca, and R. K. Jallu, *Efficient Independent Set approximation in Unit Disk Graphs*, Discrete Applied Mathematics, vol. 280, 63–70, 2020, issn: 0166-218X. doi: https://doi.org/10.1016/j.dam.2018.05.049. [Accessed Jan. 27, 2022].

[18] T. Nieberg, J. Hurink, and W. Kern, *A robust PTAS for Maximum Weight Independent Sets in Unit Disk Graphs*, in *Graph-Theoretic Concepts in Computer Science*, J. Hromkovič, M. Nagl, and B. Westfechtel, Eds., Springer Berlin Heidelberg, 2005, 214–221, isbn: 978-3-540-30559-0. doi: https://doi.org/10.1007/978-3-540-30559-0_18. [Accessed Jan. 31, 2022].

[19] H. B. Hunt, M. V. Marathe, V. Radhakrishnan, S. Ravi, D. J. Rosenkrantz, and R. E. Stearns, *NC-approximation schemes for NP- and PSPACE-hard problems for Geometric Graphs*, Journal of Algorithms, vol. 26, no. 2, 238–274, 1998, issn: 0196-6774. doi: https://doi.org/10.1006/jagm.1997.0903. [Accessed Jan. 27, 2022].

[20] T. Matsui, *Approximation algorithms for Maximum Independent Set problems and Fractional Coloring problems on Unit Disk Graphs*, in *Discrete and Computational Geometry*, J. Akiyama, M. Kano, and M. Urabe, Eds., Springer Berlin Heidelberg, 2000, 194–200, isbn: 978-3-540-46515-7. doi: https://doi.org/10.1007/978-3-540-46515-7_16. [Accessed Jan. 26, 2022].

[21] T. Erlebach, K. Jansen, and E. Seidel, *Polynomial-time approximation schemes for Geometric Graphs*, in *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, Society for Industrial and Applied Mathematics, 2001, 671–679, isbn: 0898714907. doi: https://doi.org/10.1137/S0097539702402676. [Accessed Jan. 31, 2022].

[22] D. S. Hochbaum and W. Maass, *Approximation schemes for Covering and Packing problems in image processing and VLSI*, Journal of the ACM, vol. 32, no. 1, 130–136, 1985, issn: 0004-5411. doi: https://doi.org/10.1145/2455.214106. [Accessed Jan. 31, 2022].

[23] B. S. Baker, *Approximation algorithms for NP-complete problems on Planar Graphs*, J. ACM, vol. 41, no. 1, 153–180, 1994, issn: 0004-5411. doi: https://doi.org/10.1145/174644.174650. [Accessed Jan. 29, 2022].

[24] E. J. van Leeuwen, *Approximation algorithms for Unit Disk Graphs*, in *Graph-Theoretic Concepts in Computer Science*, D. Kratsch, Ed., Springer Berlin Heidelberg, 2005, 351–361, isbn: 978-3-540-31468-4. doi: https://doi.org/10.1007/11604686_31. [Accessed Jan. 27, 2022].

[25] H. Breu and D. G. Kirkpatrick, *Unit Disk Graph recognition is NP-hard*, Computational Geometry, vol. 9, no. 1, 3–24, 1998, issn: 0925-7721. doi: https://doi.org/10.1016/S0925-7721(97)00014-X. [Accessed Jan. 26, 2022].

[26] R. J. Kang and T. Müller, *Sphere and dot product representations of graphs*, in *Discrete Computational Geometry*, vol. 47, Springer Link, 2012, 548–568. doi: https://doi.org/10.1007/s00454-012-9394-8. [Accessed Feb. 10, 2022].

[27] N. E. Mnëv, *Realizability of combinatorial types of convex polyhedra over fields*, Journal of Soviet Mathematics, vol. 28, no. 4, 606–609, 1985. doi: https://doi.org/10.1007/BF02104991.

[28] N. E. Mnev, *The universality theorems on the classification problem of configuration varieties and convex polytopes varieties*, in *Topology and Geometry - Rohlin Seminar*, O. Y. Viro and A. M. Vershik, Eds. Springer Berlin Heidelberg, 1988, 527–543, isbn: 978-3-540-45958-3. doi: https://doi.org/10.1007/BFb0082792.

[29] P. Shor, *Stretchability of pseudolines is NP-hard*, Applied Geometry and Discrete Mathematics: The Victor Klee Festschrift, vol. 4, 531–554, 1991. doi: https://doi.org/10.1090/dimacs/004.

[30] S. Basu, R. Pollack, and M.-F. Roy, *Existential theory of the reals*, in *Algorithms in Real Algebraic Geometry*. Springer Berlin Heidelberg, 2006, 505, isbn: 978-3-540-33099-8. doi: https://doi.org/10.1007/3-540-33099-2_14.

[31] M. Schaefer, *Complexity of some geometric and topological problems*, in *Graph Drawing*, D. Eppstein and E. R. Gansner, Eds., Springer Berlin Heidelberg, 2010, 334–344, isbn: 978-3-642-11805-0. [Accessed Feb. 10, 2022].

[32] H. Breu: *Algorithmic aspects of constrained Unit Disk Graphs*, Ph.D. dissertation, University of British Columbia, 1996. doi: http://dx.doi.org/10.14288/1.0051600. [Accessed Jan. 27, 2022].

[33] J. L. Bentley, D. F. Stanat, and E. Williams, *The complexity of finding Fixed-Radius Near Neighbors*, Information Processing Letters, vol. 6, no. 6, 209–212, 1977, issn: 0020-0190. doi: https://doi.org/10.1016/0020-0190(77)90070-9. [Accessed Jan. 27, 2022].

[34] M. T. Dickerson and R. Drysdale, *Fixed-Radius Near Neighbors search algorithms for points and segments*, Information Processing Letters, vol. 35, no. 5, 269–273, 1990, issn: 0020-0190. doi: https://doi.org/10.1016/0020-0190(90)90056-4. [Accessed Jan. 26, 2022].

[35] H. Lenhof and M. Smid, *An animation of a Fixed-Radius All-Nearest-Neighbors algorithm*, in *Proceedings of the Tenth Annual Symposium on Computational Geometry*, Association for Computing Machinery, 1994, 387, isbn: 0897916484. doi: https://doi.org/10.1145/177424.178099. [Accessed Jan. 26, 2022].

[36] K. Zhou, Q. Hou, R. Wang, and B. Guo, *Real-time KD-tree construction on graphics hardware*, ACM Transactions on Graphics, vol. 27, no. 5, 1–11, 2008, issn: 0730-0301. doi: https://doi.org/10.1145/1409060.1409079. [Accessed Jan. 26, 2022].

[37] D. Qiu, S. May, and A. Nüchter, *GPU-accelerated Nearest Neighbor search for 3D registration*, in *Computer Vision Systems*, M. Fritz, B. Schiele, and J. H. Piater, Eds., Springer Berlin Heidelberg, 2009, 194–203, isbn: 978-3-642-04667-4. doi: https://doi.org/10.1007/978-3-642-04667-4_20. [Accessed Jan. 26, 2022].

[38] M. Otair, *Approximate k-Nearest Neighbour based spatial clustering using K-D tree*, vol. 5, International Journal of Database Management Systems, 2013. doi: https://doi.org/10.5121/ijdms.2013.5108. [Accessed Jan. 26, 2022].

[39] F. Gieseke, J. Heinermann, C. Oancea, and C. Igel, *Buffer K-D trees: Processing massive Nearest Neighbor queries on GPUs*, in *Proceedings of the 31st International Conference on Machine Learning*, E. P. Xing and T. Jebara, Eds., vol. 32, Proceedings of Machine Learning Research, 2014, 172–180. [Online]. Available: https://proceedings.mlr.press/v32/gieseke14.html [Accessed Jan. 26, 2022].

[40] S. Green, *Particle simulation using CUDA*, 2010, Whitepaper. [Online]. Available: https://wrf.ecse.rpi.edu/wiki/ParallelComputingSpring2015/cuda/nvidia/samples/5_Simulations/particles/doc/particles.pdf [Accessed Jan. 26, 2022].

[41] R. C. Hoetzlein, *Fast Fixed-Radius Nearest Neighbors: Interactive million-particle fluids*, Presentation slides, GPU Technology Conference, 2014. [Online]. Available: https://on-demand.gputechconf.com/gtc/2014/presentations/S4117-fast-fixed-radius-nearest-neighbor-gpu.pdf [Accessed Jan. 26, 2022].

[42] J. Gross, M. Köster, and A. Krüger, *Fast and Efficient Nearest Neighbor Search for Particle Simulations*, in *Computer Graphics and Visual Computing*, F. P. Vidal, G. K. L. Tam, and J. C. Roberts, Eds., The Eurographics Association, 2019, isbn: 978-3-03868-096-3. doi: https://doi.org/10.2312/cgvc.20191258. [Accessed Jan. 26, 2022].

[43] J. T. de Balsch. (2021). "Searching for Fixed-Radius Near Neighbors with cell lists algorithm in Julia language." Blog post, [Online]. Available: https://jaantollander.com/post/searching-for-fixed-radius-near-neighbors-with-cell-lists-algorithm-in-julia-language [Accessed Jan. 26, 2022].

[44] P. Santi, *Topology control in wireless ad hoc and sensor networks*, Association for Computing Machinery, vol. 37, no. 2, 164–194, 2005, issn: 0360-0300. doi: https://doi.org/10.1145/1089733.1089736. [Accessed Jan. 25, 2022].

[45] C. Cordeiro and D. Agrawal, *Wireless sensor networks*, in *Ad Hoc and Sensor Networks*, World Scientific Publishing, 2006, 403–447, isbn: 978-981-256-681-2. doi: https://doi.org/10.1142/9789812774149_0008. [Accessed Jan. 27, 2022].

[46] W. Kiess and M. Mauve, *A survey on real-world implementations of mobile ad-hoc networks*, Ad Hoc Networks, vol. 5, no. 3, 324–339, 2007, issn: 1570-8705. doi: https://doi.org/10.1016/j.adhoc.2005.12.003. [Accessed Jan. 26, 2022].

[47] W. Hale, *Frequency assignment: Theory and applications*, Proceedings of the IEEE, vol. 68, no. 12, 1497–1514, 1980. doi: 10.1109/PROC.1980.11899. [Accessed Jan. 27, 2022].

[48] C. Toregas, R. Swain, C. ReVelle, and L. Bergman, *The location of emergency service facilities*, Operations Research, vol. 19, no. 6, 1363–1373, 1971, issn: 0030364X, 15265463. [Online]. Available: http://www.jstor.org/stable/169241 [Accessed Jan. 27, 2022].

[49] E. J. van Leeuwen: *Optimization and approximation on systems of geometric objects*, Ph.D. dissertation, Korteweg-de Vries Institute for Mathematics, 2009. [Online]. Available: https://hdl.handle.net/11245/1.307107 [Accessed Jan. 27, 2022].

[50] P. Armttage, *An overlap problem arising in particle counting*, Biometrika, vol. 36, no. 3-4, 257–266, 1949, issn: 0006-3444. doi: https://doi.org/10.1093/biomet/36.3-4.257. [Accessed Jan. 26, 2022].

[51] M. Halldórsson and J. Radhakrishnan, *Greed is good: Approximating Independent Sets in sparse and Bounded-Degree Graphs.*, Algorithmica, vol. 18, 145–163, 1997. doi: 10.1007/BF02523693. [Accessed Jan. 27, 2022].

[52] M. Mari, *Study of greedy algorithm for solving Maximum Independent Set problem*, 2017. [Online]. Available: https://www.di.ens.fr/~mmari/content/papers/rapport.pdf [Accessed Jan. 27, 2022].

[53] J. C. Ballard-Myer, *Deterministic greedy algorithm for Maximum Independent Set problem in graph theory*, 2019. [Online]. Available: https://www.gcsu.edu/sites/files/page-assets/node-808/attachments/ballardmyer.pdf [Accessed Jan. 27, 2022].

Appendix

8.1 Experimentation

Figure 24 shows an example of a cinema graph serialized to into Extensible Markup Language (XML) structure according to the GraphML[11] specification.

1. **d1:** The label of this vertex.
2. **d2:** The x and y coordinate of the seat this vertex represents.
3. **d3:** The weight of this edge.
4. **d4:** The number of vertices in the Seat Path starting from this vertex.
5. **d5:** Whether this vertex is included in an IS.
6. **d6:** The degree of this vertex.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml>
  <key id = "d1" for="node" attr.name="label" attr.type="string"/>
  <key id = "d2" for="node" attr.name="pos" attr.type="string"/>
  <key id = "d3" for="edge" attr.name="weight" attr.type="int" />
  <key id = "d4" for="node" attr.name="maxSeated" attr.type="int"/>
  <key id = "d5" for="node" attr.name="isInMIS" attr.type="boolean"/>
  <key id = "d6" for="node" attr.name="Degree" attr.type="int"/>
  <graph id = "G" edgedefault="undirected">
    <node id="0">
        <data key = "d1">e</data>
        <data key = "d2">0,0</data>
        <data key = "d4">2</data>
        <data key = "d5">True</data>
        <data key = "d6">2</data>
    </node>
    <node id="1">
        <data key = "d1">e</data>
        <data key = "d2">1,0</data>
        <data key = "d4">1</data>
        <data key = "d5">False</data>
        <data key = "d6">2</data>
    </node>
    <edge source="0" target="1">
        <data key="d3">1</data>
    </edge>
    <edge source="1" target="0">
        <data key="d3">1</data>
    </edge>
    </graph>
</graphml>
```

Figure 24: XML structure of a cinema graph according to the GraphML specification.

[11] http://graphml.graphdrawing.org/