Master Thesis

Algorithms for Automated Floor Planning

Felix Klesen

Date of Submission:	September 4, 2020
Advisors:	Prof. Dr. Alexander Wolff
	Dr. Jonathan Klawitter

Julius-Maximilians-Universität Würzburg
Lehrstuhl für Informatik I
Algorithmen und Komplexität

Abstract

This work is about automating the process of floor-planning using algorithmic tools. That is, how to distribute required entities like rooms to available space while respecting constraints and minimizing the scattering of grouped entities. We show theoretical results about properties of the underlying algorithmic problems as well as propose, implement and evaluate two different approaches to solving them. One using algorithmic insight into the problem and one using integer programming.

Zusammenfassung

In dieser Arbeit geht es um das automatisierte Erstellen von Gebäudeplänen mit algorithmischen Methoden. Dabei werden erforderliche Einheiten wie z.B. Räume auf den verfügbaren Platz verteilt, während Nebenbedingungen erfüllt sein müssen und die Zerstreung zusammengehöriger Einheiten minimiert wird. Wir zeigen theoretische Resultate zu Eigenschaften der zugrunde liegenden algorithmischen Probleme. Außerdem definieren, implementieren und testen wir zwei verschiedene Ansätze, diese Probleme zu lösen. Dabei nutzt einer algorithmische Einsichten in das Problem und der Andere ganzzahlige Programmierung.

Contents

1 Introduction **5**
- 1.1 Large Scale . . . 5
- 1.2 Small Scale . . . 5
  - 1.2.1 KD-Trees . . . 6
  - 1.2.2 Rectangle Packing . . . 6
  - 1.2.3 Elastic Map Labeling . . . 8
- 1.3 Definitions . . . 9

2 Area Distribution **11**
- 2.1 Properties . . . 12
  - 2.1.1 Sequence Model . . . 12
  - 2.1.2 Fragmentation . . . 13
  - 2.1.3 Flips . . . 13
- 2.2 Nice Sequences . . . 13
- 2.3 Approximation . . . 16
- 2.4 Exact . . . 18
- 2.5 Distance Measures . . . 18
- 2.6 Buildings . . . 19
  - 2.6.1 Single Building . . . 20
  - 2.6.2 Disjoint Buildings . . . 21
  - 2.6.3 Connected Buildings . . . 22

3 Room Assignment **25**
- 3.1 Nice sequences . . . 25
- 3.2 Relaxation . . . 26
- 3.3 Approximability . . . 27
- 3.4 Bi-criteria Approximation . . . 27
  - 3.4.1 Greedy Heuristic . . . 28
  - 3.4.2 Improved Heuristic . . . 29
  - 3.4.3 Karmarkar-Karp Heuristic . . . 29
  - 3.4.4 Buidlings . . . 30
- 3.5 Heuristic . . . 30
- 3.6 Exact . . . 30
  - 3.6.1 ILP . . . 31

4 Floor Planning **32**
- 4.1 NP-hardness . . . 33

4.2 ILP . . . 34
4.3 Optimization . . . 35
4.4 Multiple Floors . . . 38

5 People Assignment **41**
5.1 ILP . . . 42

6 Benchmark **43**
6.1 People Assignment . . . 44
6.2 Floor Planning . . . 46
6.2.1 ILP . . . 47
6.2.2 Approximation . . . 49
6.3 CPLEX-Remarks . . . 55

7 Conclusion **56**

Bibliography **57**

1 Introduction

A common problem in architecture is the planning of office buildings. After defining the shape and number of floors, there is still a long way towards a complete floor plan. Even though it is a creative task, we are going to automate the latter part since it is a huge share of the actual work with fairly objective goal. The problem can be broken down into two different algorithmic problems for different scales.

1.1 Large Scale

On a large scale, we want to assign each required room to a floor. For simplicity we do not account for the actual shape of the floors here, but use their total area as a capacity. Since each room has a size and needs to be placed within a floor while no floor is over-packed, we are interested in solving the decision version of a *bin-packing* problem.

But offices usually belong to some sort of work groups and we may not want to scatter these groups across the floors. This can be incorporated into the problem definition with an objective function. Therefore we do not have a decision problem anymore, but an optimization problem.

The problem is now similar to *event seating*, where groups of people are to be seated at tables while avoiding to break up parties. Modelling the rooms (people) as items, the groups (parties) as colours and the floors (tables) as bins, this problem has been recently defined by Bergmann et al. [BCM19] as *Bin Packing with Minimum Colour Fragmentation* (BPMCF).

After showing NP-completeness by reduction from *number-partitioning*, they also propose an exact algorithm to solve the problem using a binary decision diagram and a mixed-integer program.

Further expanding on this idea, we are going to investigate a more general version of this problem in Chapter 3, where the cost of splitting a colour set into multiple bins depends on a function measuring the distance of those bins. With this generalization it is possible to incorporate into the model that splitting a work group across two floors is worse, if there are other floors in between, or if the floors even are in different buildings.

1.2 Small Scale

After assigning the rooms to floors, we still need to create the actual floor plans, but we may do so independently for each floor. Given the sizes of the rooms and the shape of the floor, we want to find an arrangement of the rooms within their floor such that we

do not violate any constraints we might have. Reasonable constraints might include the following:

- the hallway has a minimum width,
- each room needs to have a common edge of minimum length with the hallway,
- specific room pairs need to be adjacent (sharing a common wall),
- some rooms need sunlight (have common edge with outer face, and
- aspect ratio of rooms is upper-/lower-bounded.

The goal may be to minimize the total area needed, a distance measure on the set of rooms, or a combination of both. We call this problem *Floor Planning* and further discuss it in Chapter 4 after looking at some related work in the following sections.

1.2.1 KD-Trees

Knecht [Kne11] has studied the use of kd-trees to generate floor layouts where the cells generated by the tree correspond to rooms. To find good positions for the defining points, they use a genetic algorithm. While they are able to search for solutions according to neighbour constraints, given room sizes and allow the user to manually manipulate intermediate steps, they do not account for hallways yet and room-size requirements can not be met exactly. This approach is also limited to *slicing* solutions, in which every wall is a subdivision of a part of the floor, which is not feasible for all instances. Hamacher et al. [HK14] however, have later discussed non-slicing solutions in this setting.

1.2.2 Rectangle Packing

Another way to look at floor-planning is *rectangle-packing* where a set of given rectangles are to be placed within an enclosing rectangle without overlaps while minimizing the enclosing rectangles size. This problem has a wide range of applications including cutting raw material, placing integrated circuits on a die and packing items for shipping. The most important difference to our problem is that the rectangles each have a given aspect ratio and sometimes also a fixed orientation. In our problem the aspect ratio is an additional variable a solver needs to set.

Due to the wide range of applications there has been a lot of work on the algorithmic problem which Bortfeldt [Bor13] has compiled in a list. While he discusses a new heuristic using reductions to two-dimensional *knapsack* and *strip-packing* instances, most of the other listed work is also on heuristics.

However, Huang et al. [HYC11] showed that if an instance with given enclosing rectangle is solvable, a realization can be achieved by successively placing rectangles in a bottom-left corner. This insight can be used for both heuristic and exact approaches.

Korf [Kor03] has started a series of papers investigating an exact approach; the latest release together with Huang [HK13]. While their research is benchmark-driven and

therefore not analysed in terms of runtime complexity, some of the algorithmic insights might still prove useful for our purposes.

Their general idea is a linear search for the smallest enclosing rectangle accepted by an algorithm that solves the containment version of the problem. In the latter, the rectangles are placed iteratively from large to small and branching is done on possible positions until a leaf of the search tree is reached, and therefore a solution is found, or until the branch's partial problem is unsolvable.

The key contribution to reducing the runtime then is finding ways to prune those branches as high in the search tree as possible. One such way is slicing up the empty space and the unplaced rectangles in order to define a bin-packing problem which is unsolvable if and only if the partial rectangle-packing problem is unsolvable. This approach calculates a lower bound on space that cannot be filled up anymore in order to abort when becoming too much.

Another important insight is, that some placement positions *dominate* others in terms of partial solution. For example placing a rectangle close to a corner is dominated by placing it in that corner since the set of partial solutions is a subset and hence the dominated placement can be pruned. This ultimately leads to the conclusion that rectangles may only be placed in corners.

While Korf and Huang [Kor03, HK13] dedicated a decade of work into rectangle-packing and have developed quite a complex algorithm, the largest solved instance has merely 32 squares, which are yet a bit easier than arbitrary rectangles. The solution to this instance can be seen in Fig. 1.1. Hence they conclude rectangle-packing to be a rather hard problem.

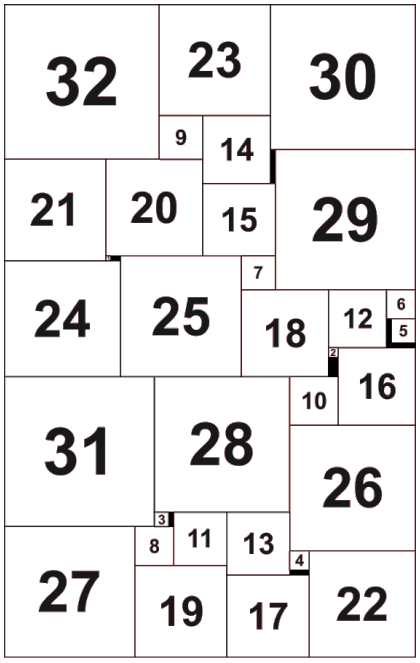


Fig. 1.1: Optimal solution for packing the smallest 32 different squares in a rectangle ([HK13]).

1.2.3 Elastic Map Labeling

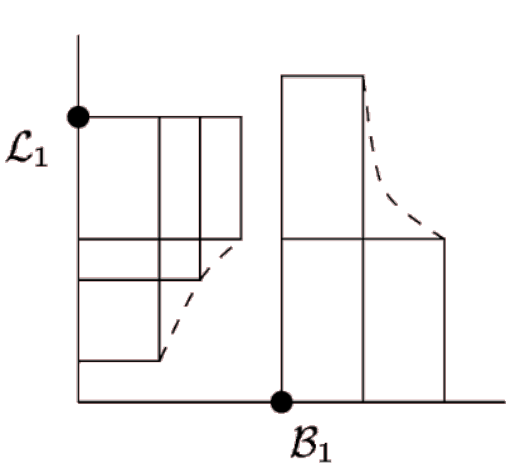
Another related problem is the placement of elastic labels around the perimeter of a map. In this problem, given a number of anchor points on the perimeter of an enclosing rectangle, each with a size, place an anchored rectangle of the given size at each point, while no pair of rectangles overlaps.

This problem has been discussed by Iturriaga et al. [IL03] in terms of labeling point features on the perimeter of a map with texts. Given the length of those texts and the direction in which the label should grow from its anchor, the height and width of the label depend on the number of line breaks. The area needed for a label, however, is roughly constant across those realizations.

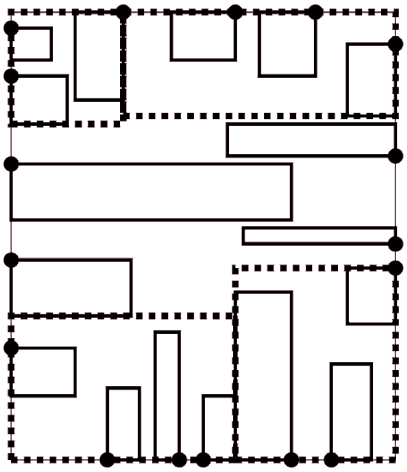
Therefore the labels are modeled as *elastic* rectangles, featuring the area, anchor point, possible directions of growth and ranges for width and height. An elastic rectangle is therefore the set of possible realizations for a label. Two elastic rectangles in a corner of the perimeter can be seen in Fig. 1.2a. The dashed curve marks possible positions for the second defining anchor points.

While the problem is NP-hard, they attempt to solve it in polynomial time. Their algorithm tries to partition the anchor points into five sets in a way that all points within a set lie on only two different straight lines. Such a subdivision can be seen in Fig. 1.2b. For the subsets it is then possible to find a good order of the points for a greedy approach to restraining and finally assigning the rectangles.

The relation to our problem is that the aspect ratio is not fixed and the idea of elastic rectangles might prove useful for our floor-planning problem. The main difference is that we do not know in advance where a given room must be placed. Also this only features a reduced set of instances where all rooms need sunlight and it does not account for having a corridor.



(a) Two elastic rectangles ([IL03]).



(b) Splitting of the rectangle in 5 regions ([IL03]).

Fig. 1.2

1.3 Definitions

The problems we define and solve all belong to the same family of problems. Therefore we give a small overview of the family and the relations within it. The basic problem the whole family is built upon is *number-partitioning*, which was shown to be NP-hard by Karp [Kar72]. We use the following notation:

Problem NUMBER-PARTITIONING (DECISION)

Item: A multi-set S of integers.

Question: Is it possible to find S_1 and S_2 such that $S_1 \uplus S_2 = S$ and $\sum_{s \in S_1} s = \sum_{s \in S_2} s$?

We use $|S|$ to refer to the sum of numbers in S instead of their count. If we generalize the problem definition to allow a number of subsets other than two, the problem is called *multi-way number-partitioning* and can be written in the following form:

Problem MULTI-WAY NUMBER-PARTITIONING (DECISION)

Item: A multi-set S of integers and a number k.

Question: Is it possible to find S_1 to S_k such that $\biguplus_{i=1}^{k} S_i = S$ and $\sum_{s \in S_i} s = \sum_{s \in S_j} s$ for any $1 \leq i < j \leq k$?

Since the two definitions are identical for $k = 2$, multi-way number-partitioning is obviously also NP-hard. At this point the family splits into multiple sub-families of optimization problems.

One such sub-family contains the *scheduling* problems. In these problems we understand the multi-set S as a set of tasks and k as the number of machines we can use to solve the tasks. Hence each of the subsets corresponds to the schedule of one machine. Algorithmically these problems exchange the requirement of all the subsets sums to be equal for an optimization expression. Such an expression can for example minimize the difference between the largest and the smallest subsets or directly the size of the largest subset as seen in the following example:

Problem MULTI-WAY NUMBER-PARTITIONING (OPTIMIZATION)

Item: A multi-set S of integers and number k.

Question: How to find S_1 to S_k such that $\biguplus_{i=1}^{k} S_i = S$, while minimizing $\max_{1<i\leq k} \sum_{s \in S_i}$?

Another branch of optimization problems are the *packing* problems. Instead of minimizing the sum of the subsets, the number thereof is minimized here. This however, first needs a small change in the underlying decision version of the problem leading to the following definition of the basic packing problem:

Problem BIN-PACKING (DECISION)

Item: A multi-set S of integers and a set of bins B, each with an integer capacity m.

Question: Is it possible to assign each element $s \in S$ to a bin $b \in B$ such that $\sum_{s \in b} s \leq m$ for every $b \in B$?

We renamed the subsets to bins and changed the equality requirement for an upper bound. This definition is still able to solve the decision version of multi-way number-partitioning by simply computing m accordingly. Therefore Garey and Johnson [GJ79] already noted that bin-packing is NP-hard. However, we are able to choose m differently making bin-packing a generalization. Based on this decision version, we can define the optimization version which now minimizes the number of bins needed for a fixed capacity:

Problem BIN-PACKING (OPTIMIZATION)

Item: A multi-set S of integers and an integer capacity m.

Question: How to assign each element $s \in S$ to a bin $b \in B$ such that $\sum_{s \in b} s \leq m$ for every $b \in B$ while $|B|$ is minimized?

The problems we define and solve generalize the multi-set of integers S to a set of items which can have additional properties. Since we optimize in such a property while fixing both the number and the size of the subsets/bins, our problems belong to neither scheduling nor packing directly. However, we still consider them to be packing problems since they share the decision version with bin-packing.

2 Area Distribution

Before we are able to place rooms in certain positions within a floor we first need to know which floor each room should be placed in. Since this is a complex question we first study a simplified version where all rooms have the same size. In Chapter 3 we will then use the results and insights we find here in order to generalize the instances we can solve.

For the reduced set of instances we want to solve in this chapter we may assume without loss of generality that all rooms have size 1 and instead of having a size for each room we can collapse the problem such that each colour has a size. We define this problem in the following way:

Problem AREA-DISTRIBUTION

Item: A set C of colours, a size function $s\colon C \to \mathbb{N}$ and a set F of floors, each with an integer capacity m.

Question: How to find a distribution of the colours items to floors $a\colon (C, F) \to \mathbb{N}$ that is a valid bin-packing solution while optimizing an objective function $\text{cost}\colon a \to \mathbb{R}$?

Note that a colour can be distributed to multiple floors, while a room can not. The size of a colour can be interpreted as the number of equal sized rooms of that colour or as the total area required for the rooms of that colour. The objective function we use here minimizes the number of floors a colour is present in summed over all colours, which is the same objective as in BPMCF. The purpose of the auxiliary variables $x_{c,f}$ is to determine whether a colour c is present in floor f.

$$\text{minimize } \text{cost}(a, \gamma) = \sum_{c \in C} \sum_{f \in F} x_{c,f} \tag{2.1}$$

$$x_{c,f} = \begin{cases} 1 & a(c, f) \geq 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

Theorem 2.1. *Solving area-distribution is NP-hard.*

Proof. We show this by reducing instances of number-partition to area-distribution instances in polynomial time. For each number from the multi-set S, we create a colour and the size of that colour is the number. We use two bins of size $|S|/2$ each to represent S_1 and S_2. The resulting instance of area-distribution is always solvable but the objective function can have two different values. If it is $|C|$ none of the colours got split into

both bins and hence we have a partition. For instances that do not have a partition, the solution will be $|C| + 1$ since one colour needs to be present in both bins. □

While this proof only shows that we are able to solve number-partitioning using area-distribution, we may also observe that the problem definition is similar to that of multi-way number-partitioning which we defined in Section 1.3. The decision version of area-distribution is a relaxed version of the decision version of multi-way number-partitioning, where we are allowed to split items among different sets. However, both problems optimize into different dimensions.

2.1 Properties

A solution to area-distribution is a mapping of rooms to floors that satisfies the size-constraints of all floors. Neither the floors nor the rooms within them are ordered in a solution. To support the observations in the following sections, we are first going to define a few properties.

2.1.1 Sequence Model

In the *sequence* model there is a total order on the bins as well as on the items within each bin. Hence it is possible to view a solution as a single sequence of items with delimiters between the bins. Note that there are multiple sequences representing the same solution as shown in Fig. 2.1. Each sequence however, maps to exactly one solution. The space left empty within the bins may be ignored in this model.

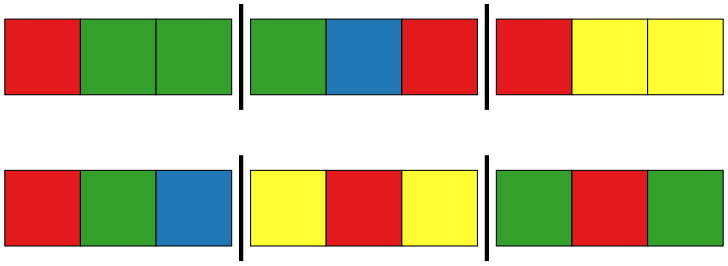


Fig. 2.1: Two sequences representing the same solution.

Furthermore we call a sequence to be *nice* if the items of every colour form an interval. Neither of the sequences in Fig. 2.1 is nice, since there are items of other colours between the red items in the left and right bin. A nice sequence for the same instance, but a different solution, can be seen in Fig. 2.2.

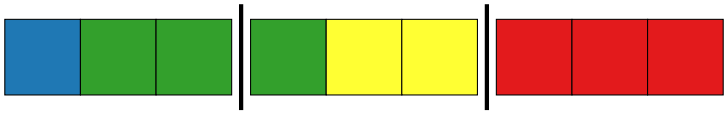


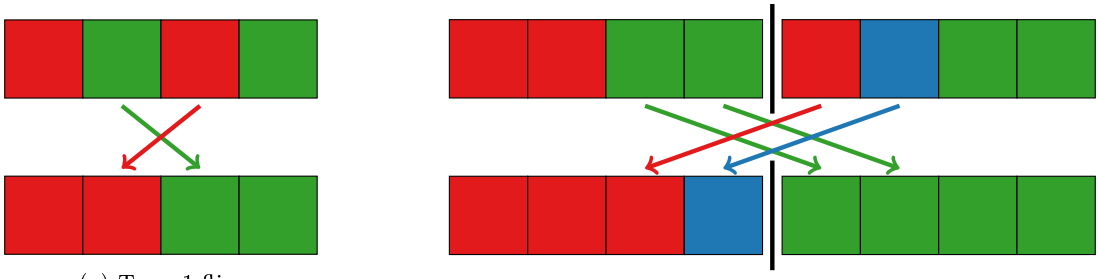
Fig. 2.2: A nice sequence.

2.1.2 Fragmentation

We define a colour's *fragmentation* to be the number of multi-coloured bins it is present in and call it *fragmented* if this number is at least two. In Fig. 2.1 red and green are fragmented colours, while blue and yellow are unfragmented. Note that this is independent of the sequence used and hence a feature of the solution. In the solution represented by the nice sequence in Fig. 2.2 for instance only green is fragmented.

2.1.3 Flips

We define a *flip* to exchange two equal-sized blocks of items with each other. The blocks can have any length but may not stretch across the boundaries of bins. There are two different basic types of flips. Fig. 2.3a shows a type 1 flip altering the sequence only within a bin and hence preserving the solution and Fig. 2.3b shows a type 2 flip across the boundary of two bins and therefore also altering the solution. While we may use type 1 flips at any time we need to be careful when using type 2 flips since the two solutions may not be equally good. In our example the value of the solution according to the colour fragmentation changes from 5 to 3.



(a) Type 1 flip.

(b) Type 2 flip.

Fig. 2.3: Two major types of flips.

2.2 Nice Sequences

While we may have the intuition that nice sequences might represent good solutions, this section is about proving the following theorem:

Theorem 2.2. *For any instance of floor-distribution there is an optimal solution admitting a nice sequence.*

Proof. We start with any optimal solution, use flips to transform it into another optimal solution with good properties and finally construct a nice sequence for it. The properties we seek are that no colour has a fragmentation of more than two and no bin has more than two fragmented colours in it.

First we remove all single-coloured bins and every bin without any fragmented colour from the solution we start with. They will remain unchanged in the following operations

and will therefore be reinserted later. Within the remaining bins we use flips of type 1 exhaustively to make each colour a block locally. This operation is visualized in Fig. 2.4. We also keep up this property in other operations by using type 1 flips.

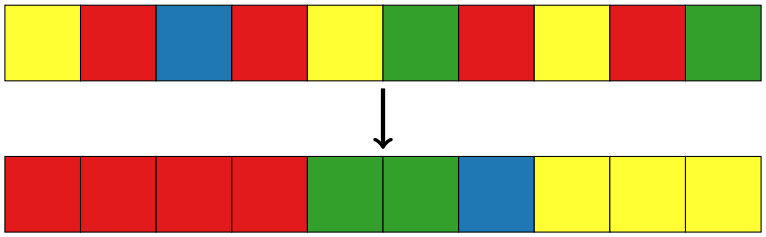
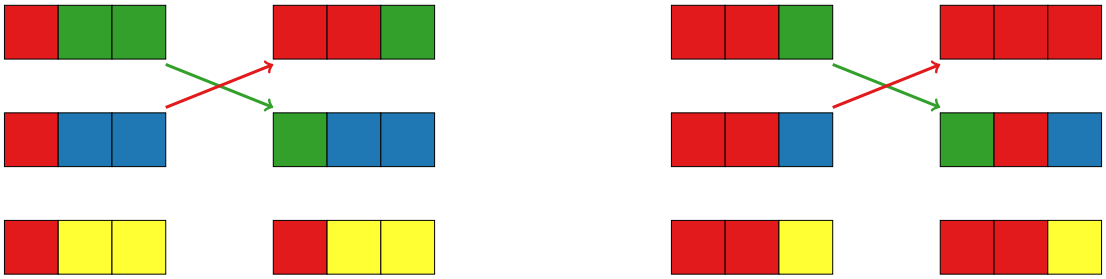


Fig. 2.4: Sorting by colour within a bin.

At this point colours may still have a high fragmentation. For any colour being present in at least three bins, we apply the following steps recursively until it is present in at most two different bins. Suppose there are two blocks of the same colour that would actually fit into one bin. In this case we can flip one of them into the bin of the other by exchanging it with a block of other colours in that bin. Note that this flip must fragment an unfragmented colour, since otherwise the solution used is not optimal. Fig. 2.5a shows an example where the fragmentation of the red items is reduced by paying with the fragmentation of the green items.

If no two red blocks fit into a single bin, we can simply fill up one of the bins with red items from another bin as seen in Fig. 2.5b. This does not change the fragmentation of either colour directly, but we are then able to remove one bin and the fragmentation of the red items is reduced by one. Once again, this flip cannot reduce the value of the solution since the second bin may not already have a green item unless the solution already was not optimal.



(a) case 1. (b) case 2.

Fig. 2.5: Reducing the fragmentation of the red items.

After applying these two cases exhaustively, every colour is present in either one or two (multicoloured) bins. Therefore we can now define a graph where the bins are vertices and the fragmented colours are edges. Such graphs can be seen in Figs. 2.6 to 2.8. Since every edge corresponds to a colour being present in two bins, the contribution to the objective function of those bins still present now equals $|C| + |E|$. Hence the number of

edges may not change throughout the following operations.

The graph may not contain a cycle since it would be possible to shift the items around the cycle until at least one colour becomes unfragmented. This operation can be seen in Fig. 2.6 and since it removes an edge the used solution was not optimal. The same argument also applies to multi-edges and therefore the graph is a forest.

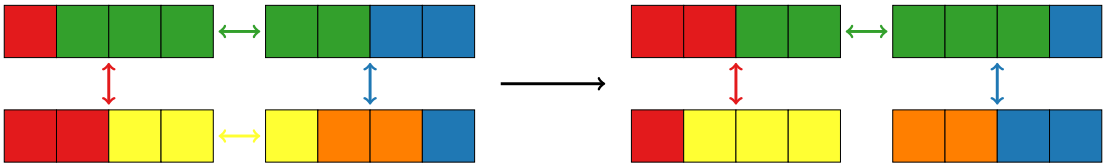


Fig. 2.6: Shifting items in a graph with cycle.

The graph may still have high degree vertices which we need to remove in order to construct a nice sequence from the graph. To accomplish this we introduce an operation that can move edges one-sidedly, meaning that one endpoint remains the same while the other end is reconnected to another vertex. Internally this is done by a flip that can be seen in Fig. 2.7. The high degree vertex in the example has three adjacent edges, so we need to move one of them. We choose to move the green edge and can then choose a new endpoint among the other neighbouring vertices. Since we choose the neighbour along the blue edge, we perform a flip along that edge. The green block from the first bin is flipped with a blue block of equal size from the second bin.

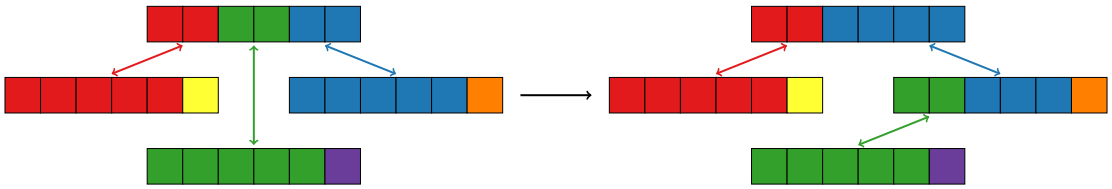


Fig. 2.7: Moving an edge by flipping item blocks case 1.

Since the second bin may not have enough blue items as seen in Fig. 2.8, the flipped block can be filled up with other items of unfragmented colours. If there are not enough of such items, the solution used is not optimal. In this case the flip changes the colour of the edge we flip along, but can neither remove it nor make it a multi-edge. Note that the solution used for showing this case is not optimal.

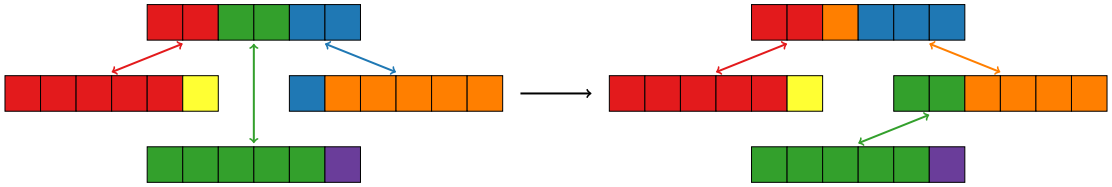


Fig. 2.8: Moving an edge by flipping item blocks case 2.

Since we are able to choose which edge to move and where to move it, we can use this operation to move an edge along a path until it reaches a leaf. When it reaches a leaf there is one vertex of high degree less. After applying this technique to all such vertices, each component of our graph is a path. Now we can reinsert the single-coloured bins at the edge corresponding to their colour and bins without fragmented colours as degree 0 vertices. Reinserting a single-coloured bin of an unfragmented colour can be done by splitting the bin that colour is present in as seen in Fig. 2.9.

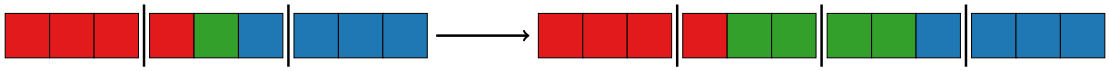


Fig. 2.9: Reinserting a green bin.

A sequence can now be given by putting the components of the graph together in any order and orientation. The resulting sequence for the graph from Fig. 2.7 can be seen in Fig. 2.10. Starting with any optimal solution, we computed a nice sequence and hence Theorem 2.2 holds. □

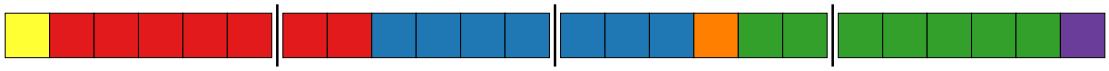


Fig. 2.10: A nice sequence for the graph from Fig. 2.7.

2.3 Approximation

In this section we use nice sequences for a 2-approximation of area-distribution.

Theorem 2.3. *For any instance of area-distribution we can compute a nice sequence in* $O(|C| + |F|)$ *time.*

Proof. Pick any permutation of the colours. Starting with an empty bin and the first colour, check whether the bin has enough space for all items of the colour. If that is the case put all items of that colour into the bin and continue with the next colour. Otherwise fill the bin completely with the colour and continue with the next bin. For such a step we only need a constant amount of time for comparing the remaining number of items the colour has and the remaining capacity of the bin. In each such step we either reduce the number of unfinished colours, or the number of empty floors by 1. Hence there may only be $|C| + |F|$ steps until all colours got assigned. □

Theorem 2.4. *For any instance of area-distribution the solution represented by any nice sequence is a* 2*-approximation.*

Proof. The number of bins each colour uses in a nice sequence is at most one higher than the minimum required, because we may only shift the coloured block but cannot scatter

it. Since each colour uses at least one bin in any optimal solution, the number of used bins per colour can at most double. These observations can be formalized as follows:

$$\frac{\mathrm{OPT} + |C|}{\mathrm{OPT}} \leq \frac{2|C|}{|C|} = 2 \tag{2.3}$$

□

Theorem 2.5. *The approximation factor of* 2 *is asymptotically sharp for using any nice sequence to approximate area-distribution.*

Proof. We first show the existence of an example with $m = 2$ and then generalize it. For this example we need two colours with one item each, $|C| - 2$ colours with two items each and $|F| = |C| - 1$. As we can see in Fig. 2.11 the approximation can have a value of at most $2|C| - 2$.

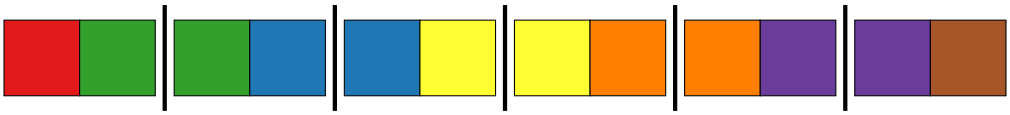


Fig. 2.11: Constructing a sharp instance for the approximation.

An optimal solution, however, would put the two single items into the same bin, resulting in a value of $|C|$. Hence we can show, that our analysis is sharp:

$$\lim_{|C| \to \infty} \frac{2|C| - 2}{|C|} = 2 \tag{2.4}$$

We can further construct a family of instances by giving the $|C| - 2$ colours m items each, for any $m \geq 2$.

□

For a finer analysis we need to find a better lower bound for OPT. While every colour needs to be in at least one bin, we may also observe that every bin has items of at least one colour. Therefore we know, that $\mathrm{OPT} \geq \max(|C|, |F|)$. We can further observe that every colour uses exactly one transition between two bins for each bin it is present in apart from the first. In other words, all bins have two fragmented colours at most:

$$\frac{|C| + |F|}{\mathrm{OPT}} \leq \frac{|C| + |F|}{\max(|C|, |F|)} = 1 + \min\left(\frac{|F|}{|C|}, \frac{|C|}{|F|}\right) \tag{2.5}$$

Therefore the approximation performs better than with a factor of 2 if $|F| \neq |C|$. Nice sequences are hence a very good approximation for instances with either $|F| \ll |C|$ or $|C| \ll |F|$.

2.4 Exact

However, we might be interested in solving area-distribution exactly:

Theorem 2.6. *Area-distribution can be solved exactly using nice sequences in* $O(|C|! \cdot 2^{\min(|C|,|F|)})$ *time.*

Proof. Among all nice sequences an instance admits, there must be at least one representing an optimal solution. To enumerate them we try all $|C|!$ permutations of the colours. Additionally at each of the $|C| - 1$ transitions between two colours there may be a gap of empty space. Since each gap may only stretch till the end of its bin, there are only two possibilities per transition. There also may not be more than $|F|$ gaps in total and therefore the overall runtime is $O(|C|! \cdot 2^{\min(|C|,|F|)})$. Here $|C|$ is the size of the input because the items do not have properties of their own and can be represented by a single number per colour. □

2.5 Distance Measures

As already mentioned, we want to additionally include a function $\delta\colon (F, F) \to \mathbb{R}$ measuring the distance of floors. For BPMCF this function would simply be $\delta(f_1, f_2) = 1$ since all bins are equal. Within a building, a reasonable distance measure may be $\delta(f_1, f_2) = |\ell(f_1) - \ell(f_2)|$ where $\ell(f)$ denotes the level of f in the building. In a multi-building setting, a graph may be used to define distances with $\delta(f_1, f_2)$ then being the length of a shortest f_1-f_2-path.

The introduction of a distance measure also requires different objective functions. The goal may then be to minimize the maximum distance of two bins having items of the same colour (Eq. 2.6). If we want to take this distance into account for each colour, we can sum the distances up (Eq. 2.7) making it equally important to lower the maximum distances for all colours. Alternatively we can define a $|C|$-dimensional vector with the values for each colour in descending order. Two vectors can then be compared lexicographically to first minimize the highest distance and subsequently the lower distances. If we do not only want to consider greatest distances, we can also minimize over all floors a colour is present in by summing all those distances up (Eq. 2.8).

$$\max_{c \in C} \max_{f_1, f_2 \in F} x_{c,f_1} \cdot x_{c,f_2} \cdot \delta(f_1, f_2) \tag{2.6}$$

$$\sum_{c \in C} \max_{f_1, f_2 \in F} x_{c,f_1} \cdot x_{c,f_2} \cdot \delta(f_1, f_2) \tag{2.7}$$

$$\sum_{c \in C} \sum_{f_1, f_2 \in F} x_{c,f_1} \cdot x_{c,f_2} \cdot \delta(f_1, f_2) \tag{2.8}$$

To illustrate the difference these cost functions have in practice, there is an optimal solution for each of them for the same instance in Fig. 2.12.

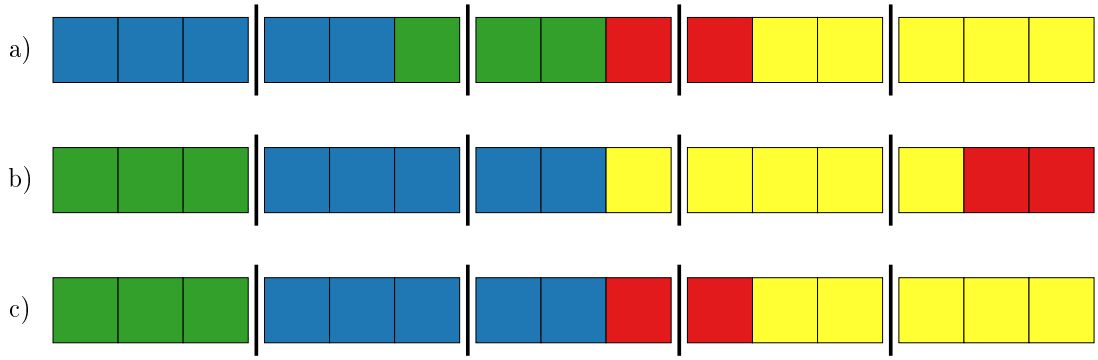


Fig. 2.12: Optimal solutions according to different measures.

Solution a) is optimal for Eq. 2.6 because no colour has a maximum distance of 2 and 0 is not possible. But this comes at the price of fragmenting all colours. Using Eq. 2.7 we can for example get the optimal solution b) where only two colours are fragmented. This, however, comes at the price of having one colour in three different floors. To avoid this behaviour we can use Eq. 2.8 and get an optimal solution like the one seen in c).

The values for each combination of cost-function and solution can be seen in the following table:

	Eq. 2.6	Eq. 2.7	Eq. 2.8
a)	1	4	4
b)	2	3	5
c)	1	3	3

We conclude that Eq. 2.6 does not discriminate well and we may not want to use it. The question of whether Eq. 2.7 or Eq. 2.8 is the right choice depends on how we want to weigh collective interest against individual interest.

2.6 Buildings

In this section we introduce buildings into our model in order to properly account for the distance between floors. Section 2.6.1 is about the simple case with only one building. In Section 2.6.2 we may have any number of independent buildings and in Section 2.6.3 those buildings are also allowed to be connected.

All these setups have in common, that they use a distance function $\delta\colon (F, F) \to \mathbb{R}$ to describe how far a pair of floors f_1 and f_2 are apart from each other. Accordingly we also need another objective function to incorporate the distance function. Here we want to minimize the maximum distance of same-coloured rooms summed over all colours:

$$\text{minimize } \operatorname{cost}(a,\delta) = \sum_{c\in C}\left[1 + \max_{f_1,f_2\in F} x_{c,f_1}\cdot x_{c,f_2}\cdot\delta(f_1,f_2)\right] \tag{2.9}$$

$$x_{c,f} = \begin{cases} 1 & a(c,f)\geq 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.10}$$

2.6.1 Single Building

Here we consider a single building with all its floors stacked and equally spaced. Therefore we can use the level $\ell(f)$ of a floor f to determine its height in the building and $\delta(f_1,f_2) = |\ell(f_1)-\ell(f_2)|$ to measure the distance of two floors f_1 and f_2. We can formalize the problem in the following way:

Problem AREA-DISTRIBUTION IN SINGLE BUILDING

Item: A set C of colours, a size function $s\colon C\to\mathbb{N}$, a set F of floors, each with an integer capacity m and a distance function $\delta\colon (F,F)\to\mathbb{R}$.

Question: How to find a distribution of the colour's items to floors $a\colon (C,F)\to\mathbb{N}$, that is a valid bin-packing solution while optimizing an objective function $\operatorname{cost}\colon (a,\delta)\to\mathbb{R}$?

We observe that the maximum distance for items of the same colour c is minimal when all its rooms are placed in the least possible number of adjacent floors. Using the level of the floors as a total order on our bins, this observation means we can rearrange our colour's items within their bins to let the colour form an interval in the sequence model. Since this can be done for all colours we can conclude:

Theorem 2.7. *Any optimal solution for an instance of area-distribution in single building can be represented by a nice sequence.*

Proof. Given an optimal solution, we remove all single-coloured bins. Each colour can then only be in two of the remaining bins and those need to be adjacent in the ordered stack of bins as we observed before. Therefore no bin can have more than two fragmented colours and the graph is a set of paths aligned with the order of the bins. From this we can get a nice sequence by only rearranging the items within their bins with type 1 flips. Hence we can find a nice sequence without changing the solution. □

Theorem 2.7 and Theorem 2.2 have a similar statement for their respective problem with the only difference, that the latter does not hold true for any optimal solution. Therefore Theorem 2.7 is the strictly stronger statement and we may use any results we obtained using Theorem 2.2 for the single-building setup. This includes both the approximation algorithm from Section 2.3 and the exact algorithm from Section 2.4.

2.6.2 Disjoint Buildings

Here we have a set of buildings B with equally sized floors, but possibly with different numbers thereof. We do further require that every working group is placed completely within one building. This is particularly important if the buildings are in different locations or even in different cities. Therefore we set the distance of two floors of different buildings to infinity, but the distance of two floors of the same building is still measured by their levels. We can formalize the problem as follows:

Problem AREA-DISTRIBUTION IN DISJOINT BUILDINGS

Item: A set B of buildings, a set C of colours, a size function $s\colon C \to \mathbb{N}$, a set F of floors, each with an integer capacity m, a function $b\colon F \to B$ indicating which building a floor belongs to and a distance function $\delta\colon (F, F) \to \mathbb{R}$.

Question: How to find a distribution of the colours items to floors $a\colon (C, F) \to \mathbb{N}$, that is a valid bin-packing solution and does not distribute any colour's items to floors of multiple building while optimizing an objective function cost$\colon (a, \delta) \to \mathbb{R}$?

The distance function we use according to this definition is the following:

$$\delta(f_1, f_2) = \begin{cases} |\ell(f_1) - \ell(f_2)| & b(f_1) = b(f_2) \\ \infty & \text{otherwise} \end{cases} \tag{2.11}$$

In this setting a solution can be represented by a set of disjoint sequences, where each such sequence represents a building. The length of a sequence is hence bound from above through the number of floors its building has. Since we do not allow colours to be in multiple sequences we can use the result from before to show the following:

Theorem 2.8. *Any optimal solution for an instance of area-distribution in disjoint buildings can be represented by a set of disjoint nice sequences.*

Proof. Every colour gets assigned to a single building in any solution. Therefore we can apply Theorem 2.7 to every building independently. □

Using the results concerning single buildings, however, needs some additional work. Since we do not know which colour should belong to which building, we may try all possible assignments. Whether an assignment is possible is determined for each building by its number of floors and the sum of items its colours have. This, however, results in an additional factor of $O(|B|^{|C|})$ for the runtime of those algorithms. In particular this means, the approximation algorithm from Section 2.3 does not run in polynomial time for this setup.

Theorem 2.9. *Area-distribution in disjoint buildings is NP-hard to approximate.*

Proof. Recall the decision version of bin-packing as given in Section 1.3. Model the fixed number of bins as buildings and the items as colours. Since floor-distribution in disjoint buildings does not optimize on the number of buildings but on the colour fragmentation within the buildings, it may only make imperfect decisions within the buildings for approximation. The assignment of colours to buildings, however, needs to stay perfect because the objective function would otherwise return infinity as defined in the distance function. Since no approximation algorithm may return an arbitrarily bad solution for any instance, we are not allowed to weaken the requirement of no colour being present in more than one building. This, however, results in any approximation algorithm assigning the colours to buildings and hence solving the bin-packing instance we reduced in the beginning. Since bin-packing is NP-hard, we can conclude that area-distribution in disjoint buildings is NP-hard to approximate. □

To solve the problem exactly, we can use an algorithm enumerating bin-packing solutions for assigning the colours to building and then use the algorithm from Section 2.4 for each building in each enumerated solution. However, this results in an additional factor exponential in $|C|$.

Theorem 2.10. *Area-distribution in disjoint buildings can be solved exactly in* $O(|B|^2 \cdot |C|! \cdot 2^{\min(|C|,|F|)})$ *time.*

Proof. We can directly enumerate nice sequences for the whole instance as seen in Section 2.4 and then try to subdivide them into the buildings we need. We can do this with an unwrapping algorithm. In each step we need to cut off a part of the sequence from either end in the length of one of the remaining buildings. Trying a combination of one building and one end of the sequence can be done in constant time since we only need to check whether the two bins we cut through share a colour. However we do not know the order in which we can cut off the buildings. Therefore we need $O(|B|^2)$ additional time and are able to solve this version of the problem exactly in $O(|B|^2 \cdot |C|! \cdot 2^{\min(|C|,|F|)})$ time. □

2.6.3 Connected Buildings

In this setup we have multiple buildings and rooms of the same colour which may be placed within different buildings. While we handle distances within a building like before, we additionally need a distance function $\delta\colon (B, B) \to \mathbb{R}$ to measure the distance of buildings. To avoid the need of further context about how the buildings are located we require this function as input. The corresponding problem definition is:

Problem AREA-DISTRIBUTION IN CONNECTED BUILDINGS

Item: A set B of buildings, a set C of colours, a size function $s\colon C \to \mathbb{N}$, a set F of floors, each with an integer capacity m, a function $b\colon F \to B$ indicating which building a floor belongs to and the distance functions $\delta\colon (F,F) \to \mathbb{R}$ and $\delta\colon (B,B) \to \mathbb{R}$

Question: How to find a distribution of the colour's items to floors $a\colon (C,F) \to \mathbb{N}$, that is a valid bin-packing solution while optimizing an objective function cost$\colon (a,\delta) \to \mathbb{R}$?

Assuming the buildings are all connected on ground level the distance function between floors can then be given in the following way:

$$\delta(f_1,f_2) = \begin{cases} |\ell(f_1) - \ell(f_2)| & b(f_1) = b(f_2) \\ \delta(b(f_1),b(f_2)) + \ell(f_1) + \ell(f_2) & \text{otherwise} \end{cases} \tag{2.12}$$

Theorem 2.11. *Any optimal solution for floor-distribution in connected buildings can be represented by a set of possibly connected nice sequences.*

Proof. First split every colour into one separate colour for each building it is present in. Since no colour is still present in multiple buildings any more, we have a solution with disjoint buildings and may apply Theorem 2.8 to show that every building can now be represented by a nice sequence. Now we merge the colours we split before again, which does not change the sequences within the buildings, but only adds connections between buildings. □

Note that a building may have multiple connections to other buildings and that a colour may be split among more than two buildings. Using the nice sequences of each building as nodes and the connections between them as edges, we can define a graph to represent optimal solutions for connected buildings. Such a graph of connected nice sequences can be seen in Fig. 2.13. The first floors are in the centre of the figure and the buildings grow towards the left or right respectively. Note that the purpose of the example is to illustrate many possible ways a colour can be split up in this setting. The used solution therefore may not be optimal for all possible distances between the buildings.

Theorem 2.12. *Area-distribution in connected buildings is NP-hard to approximate.*

Proof. Area-distribution in disjoint buildings is a special case of area-distribution in connected buildings with $\delta(b_1,b_2) = \infty$. Hence the theorem follows directly from Theorem 2.9. □

Theorem 2.13. *Area-distribution in connected buildings can be solved exactly.*

Proof. The nice sequences of connected buildings can not necessarily be concatenated to a single nice sequence. Therefore we may not enumerate nice sequences and then split them up here. Instead we first determine which building a room shall be assigned

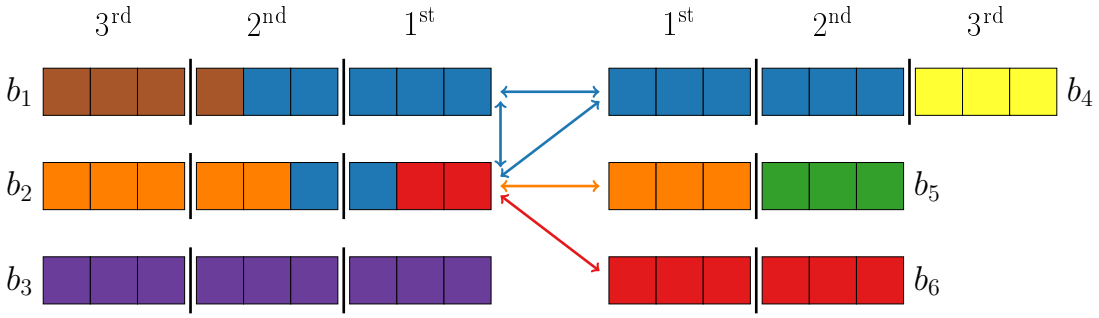


Fig. 2.13: A graph of (possibly) connected nice sequences.

to. This can once again be done with an algorithm enumerating bin-packing solution in $|B|^{|C|}$ time.

For each such distribution and within each building we enumerate nice sequences and try all combinations. The number of nice sequences has an upper bound of $|C|! \cdot 2^{\min(|C|,f(b))}$, where $f(b)$ is the number of floors in building b, because there is no trivial bound on how many colours may be present within each building.

Doing this for every building and every possible distribution of the items to buildings results in a runtime of $O(|B|^{|C|} \cdot \prod_{b \in B} |C|! \cdot 2^{\min(|C|,f(b))})$. □

For the relatively small example in Fig. 2.13 this formula already evaluates to the order 10^{32}. Hence the given algorithm is infeasible even for small instances. The problem here clearly is $|C|!^{|B|}$, which is that the order of the colours may be different in every building. Hence we may only use this approach for instances with a very low number of buildings and colours.

3 Room Assignment

In Chapter 2 we saw how to solve the special case of room-assignment with unit-sized items but already introduced buildings into the model. In this chapter we generalize those results to allow rooms of differing sizes. However, we mainly discuss approximations here, because we introduce a different approach in Section 4.4 which is able to solve room-assignment as a special case.

The basic version of room-assignment without a building giving an order to the floors can be defined in the following way:

Problem ROOM-ASSIGNMENT

Item: A set R of n rooms, a size function $s\colon R \to \mathbb{N}$, a function $\gamma\colon R \to \{1, \ldots, |C|\}$ assigning each room a colour (group), and a set F of floors, each with an integer capacity m

Question: How to find an assignment of rooms to floors $a\colon R \to F$, that is a valid bin-packing solution while optimizing an objective function $\text{cost}\colon (a, \gamma) \to \mathbb{R}$?

When using the following objective function, room-assignment minimizes the floor's colour fragmentation in the same way as BPMCF. The purpose of the auxiliary variables $x_{c,f}$, again, is to tell whether any room in floor f has colour c.

$$\text{minimize } \text{cost}(a, \gamma) = \sum_{c \in C} \sum_{f \in F} x_{c,f} \tag{3.1}$$

$$x_{c,f} = \begin{cases} 1 & \bigvee_{r \in R} \Big[\Big(a(r) = f \Big) \wedge \Big(\gamma(r) = c \Big) \Big] \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

Since we are also able to use different objective functions, room-assignment is a generalization of BPMCF.

3.1 Nice sequences

For area-distribution we used nice sequences to model solutions and design algorithms. While room-assignment is closely related to area-distribution we can show the following:

Theorem 3.1. *There are instances of room-assignment without any optimal solution that is representable by a nice sequence.*

Proof. The sequence shown in Fig. 3.1 belongs to the only possible solution for its instance and hence is optimal. Since the solutions graph has a cycle, no sequence for that solution can be nice. □

Fig. 3.1: The only optimal solution for an instance not admitting any nice sequence.

We just showed that Theorem 2.2, which we used for all results about the different versions of area-distribution, does not work for room-assignment. Therefore we may not use any of the results for area-distribution directly for room-assignment.

However, if we are able to guarantee, that at least one third of the space within the bins will remain empty, we can always find a nice sequence representing an optimal solution like in Fig. 3.2.

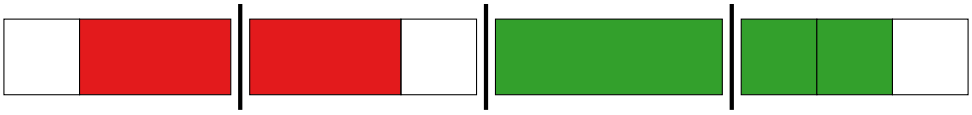


Fig. 3.2: A nice sequence representing an optimal solution for the instance from Fig. 3.1 with an additional floor available.

This however, means we are able to use only two thirds of the actual space. This might be optimal in terms of our cost function, but certainly not in any other terms. Therefore we will not use this approach.

3.2 Relaxation

In this section we define a relaxation of room-assignment instances to area-distribution instances.

Given an instance of room-assignment we group the rooms by their colour and for each colour we sum the sizes of its rooms. We then use this sum as the size of the colour in an instance of area-distribution. All other properties of the new instance are inherited from the room-assignment instance.

The relaxation hence solves the same instance while dropping the requirement, that a room can not be split into different floors. Therefore the set of valid solutions of any room-assignment instance is a subset of the corresponding relaxed area-assignment instance. Furthermore we can show:

Theorem 3.2. *The cost of an optimal solution for any instance of room-assignment cannot rise by relaxing it to an instance of area-distribution.*

Proof. Subdividing all rooms in a solution of a room-assignment instance gives a valid solution for its relaxation with the same cost. However, it does not need to be optimal. □

Therefore we can use optimal solutions for the relaxation of a room-assignment instance as a lower bound for the unrelaxed version.

3.3 Approximability

Here we will show that there is no algorithm approximating room-assignment in polynomial time with an argument similar to that we already used in Section 2.6.2.

Theorem 3.3. *Room-assignment is NP-hard to approximate.*

Proof. We show this with a polynomial time reduction from bin-packing. Given an instance of the decision version of bin-packing as defined in Section 1.3 we create an instance of room-assignment by giving every room the same colour. Apart from this we let the constructed instance inherit all properties from the bin-packing instance. Any solution for the room-assignment instance solves the underlying bin-packing instance regardless of the cost-functions value. Therefore any algorithm solving room-assignment with any quality also solves bin-packing and hence cannot run in polynomial time. □

3.4 Bi-criteria Approximation

In this section we try to parametrize the difficulty of room-assignment instances in order to define a *bi-criteria approximation* algorithm. That is a generalization of approximation algorithms, where we are allowed to violate some constraint that defines a proper solution. The degree of violation must be bound by a factor β along with a solution quality bound by a factor α as usual. Here α still refers to OPT of the version without constraint-violation. The result then is an (α, β)-approximation running in polynomial time.

In room-assignment we require every room to be assigned to a single floor where it gets the area $s(r)$ it needs. Here, however, we will violate this requirement by only guaranteeing a room gets $s(r)/\beta$ space, for $\beta > 1$. Hence β is the factor by which we allow rooms to be smaller than their requirement.

For this purpose we add an additional constraint to the room's sizes. Instead of $s\colon R \to \mathbb{N}$ we require $s\colon R \to \{1, \ldots, k\}$ for some $k \in \mathbb{N}$. This does not actually reduce the set of instances, but instead provides the parameter k we need here.

Starting with an instance of floor-assignment we first compute its relaxation to an instance of area-distribution. Then we apply the 2-approximation algorithm from Section 2.3 in $O(|C| + |F|)$ time. Due to Theorem 2.2 we can use the given solution to find another equally good solution that we can represent by a nice sequence. This nice sequence, however, still represents a solution to the area-distribution instance and needs to be transformed back to the original room-assignment instance. Through the transformation we will keep the structure of the nice sequence. Therefore we can apply the first step to each colour independently.

The space a colour gets assigned to forms an interval within the nice sequence. Therefore the space is distributed among two partial floors at most and any number of complete floors in between them. The task now is to subdivide these spaces into the rooms we need. If the colour gets assigned to only a single floor, we are always able to do the subdivision exactly. Otherwise we can model the spaces as bins and the rooms we seek as items of a bin-packing instance. However, this instance is unlikely to be solvable and therefore we allow slightly over-packing the bins and use heuristics for the distribution.

3.4.1 Greedy Heuristic

The instance we want to solve heuristically here is actually similar to multi-way number partitioning as defined in Section 1.3. The main difference is that we do not want our sets to be filled uniformly but according to the size of the bin they represent. Nevertheless, we can design a greedy heuristic based on an algorithm Graham [Gra66] designed for multi-way number-partitioning in the context of minimizing the longest processing time in multiprocessor setups.

First we sort the rooms in order of decreasing size. Then we iterate greedily through the rooms, always placing the largest room into the bin with the largest remaining space. At this point, Graham instead chooses the set with the lowest current sum. If the largest remaining space is too small for the room, we place it there anyway over-packing the corresponding bin.

Since the sum of empty spaces left is always at least as large as the sum of sizes of rooms left, we never need to place a room into an already (over)full bin. Hence after placing all rooms we may have overpacked bins by at most one room. Since there must have been at least one unit of space left before placing the last room, no bin can be overpacked by more than $k - 1$. The same bound also holds for under-packing.

The space m a floor provides can be split among any number of colours, but since we started with a nice sequence, at most two of them can be fragmented. Since unfragmented colours can be subdivided into rooms exactly, at most two colours per floor need to use the heuristic. Therefore we need to fit no more than two possibly overpacked bins into one floor and the over-/under-packing limit gets doubled. For each floor we then scale all rooms by the same factor β in order to fit all of them back into the floor.

Due to the upper bound on over-packing the bins we can also give an upper bound for the scaling factor:

$$\beta \leq 1 + \frac{2k - 2}{m} \tag{3.3}$$

The goal of the used heuristic, however, is to minimize the deviation from perfectly fitted rooms and therefore also minimizes this scaling factor. The given upper bound is only achieved by few constructed examples and we may expect something a lot better in practice.

Since all the additional work for transforming the instances back and forth can be done in linear time, sorting the rooms is the dominating contributor here. Therefore this

approach results in an $(2, 1 + (2k-2)/m)$-approximation algorithm for room-assignment running in $O(|C| + |F| + n \cdot \log n) \in O(n \cdot \log n)$ time.

3.4.2 Improved Heuristic

We can further improve the described heuristic to get a better bi-criteria approximation:

Theorem 3.4. *There is a $(2, 1 + \frac{k-1}{m})$-approximation for room-assignment, where k is the size of the largest room and m is the area each floor provides.*

Proof. Within a nice sequence the space a colour gets assigned to forms an interval and hence we have a total order for the colours. For the fragmented colours we additionally have a total order on the floors they got assigned to. At this point we once again first handle each colour independently.

For fragmented colours we first decide which of its rooms get assigned to the first floor it has space in. We greedily assign rooms to that floor in order of decreasing size, but do not allow using more space than the colour has in that floor. Hence we will at most leave $k-1$ space unused. Then we distribute the remaining rooms among the other floors in the same way we used before, which was greedily while allowing to over-pack the bins that represent the space within each floor. The over-packing is once again upper bounded by $k-1$ space.

After we have done this for every colour, we can put the partial solutions back together again. Since we made sure that the space each colour has in its first bin is not exceeded, the transition between two colours may only feature one over-packed bin at most, instead of two. Therefore we do not need to double the over-packing limit and hence we achieve a better bound for β:

$$\beta \leq 1 + \frac{k-1}{m} \tag{3.4}$$

□

3.4.3 Karmarkar-Karp Heuristic

As already mentioned the problem we try solve is closely related to multi-way number-partitioning. Another popular algorithm for this problem was defined by Karmarkar and Karp [KK83]. We will shortly outline the idea for the special case of two-way number partitioning but the algorithm works for any number of subsets.

First sort the numbers in decreasing order and then iteratively replace the largest two numbers by their absolute difference while keeping the list sorted. The last number remaining corresponds to the quality of the solution we have found. To construct the solution, place the remaining number in either set and iterate backwards through the steps we did before. In each step replace the inserted number by the two numbers it is the difference of. After all differences are resolved we have two sets with the original numbers and the difference we computed in the first step.

The optimization here is minimizing the difference of the sets. We are instead interested in minimizing the deviation of the set's difference from the difference in size our bins

have. However, it remains unclear how to generalize the differencing heuristic to aim for a difference other than zero.

3.4.4 Buidlings

Here we want to reintroduce a distance function for the floors. We do this in the same way we have already seen in Section 2.6.

For room-assignment in single building we can use the same relaxation approach we used already. This is because the only requirement we had for being able to fit the rooms back into their floor was, that no floor has more than two fragmented colours. This is true for any nice sequence and the solutions for approximating area-distribution in single buildings were nice sequences.

For room-assignment in disjoint buildings we may not find a bi-criteria approximation since we showed that we can not approximate area-distribution in disjoint building with Theorem 2.9. The same also holds true for room-assignment in connected buildings.

3.5 Heuristic

We can also use the relaxation approach to find solutions with exact room sizes. To achieve this, we do not directly solve the relaxation, but instead solve an instance with floors of size $m' = m - k + 1$ that is otherwise identical. Computing the 2-approximation for this new instance is forced to leave at least $k - 1$ space in every floor. Refitting the over-packed bins from the improved heuristic is therefore always possible without scaling.

This is of particular interest if the given room sizes are not meant to be used as estimates, but as minimums. Then we can just scale the rooms within a floor up to use all the remaining space or leave it empty for future use. However, this comes at the cost of losing the approximation factor, since there is no way to bound the cost of such a solution relative to an optimal solution. Therefore we just defined a heuristic for solving room-assignment.

3.6 Exact

If we tolerate the β-violation of rooms sizes as seen in Section 3.4, we can also use the same approach for an exact algorithm. Compute the relaxation and run the exact algorithm from Section 2.4 in $O(|C|! \cdot 2^{\min(|C|,|F|)})$ time. Run the heuristic for every colour and scale the rooms in order to fit them back into their floors.

The same approach can also be used for room-assignment in single building, disjoint buildings and connected buildings because the algorithms for solving their relaxation use nice sequences. For exact algorithms we can also use exact approaches instead of the improved heuristic. However, this only minimizes the constraint violation but cannot eliminate it.

While this approach gives us optimal solutions a constraint violation is something, we might not tolerate in an exact algorithm. Therefore we will instead look at a different

way to solve room-assignment exactly without violating constrains. This can be achieved through an integer linear program.

3.6.1 ILP

The ILP follows directly from our problem definition. The variables in Eq. 3.5 correspond to the assignment function stating whether a room r gets assigned to the floor f. Eq. 3.6 then makes sure that every room gets assigned exactly once and Eq. 3.7 ensures that no floor gets overpacked.

$$a_{r,f} \in \{0,1\} \quad \forall r \in R, f \in F \tag{3.5}$$

$$\sum_{f \in F} a_{r,f} = 1 \quad \forall r \in R \tag{3.6}$$

$$\sum_{r \in R} a_{r,f} \cdot s(r) \leq m \quad \forall f \in F \tag{3.7}$$

For the optimization part we need a few additional equations. Like before $x_{c,f}$ in Eq. 3.8 tells whether a floor f has colour c. To ensure this behaviour, we use Eq. 3.9, where $\gamma(r,c)$ is 1 if $\gamma(r) = c$ and 0 otherwise. Eq. 3.10 then minimizes the colour fragmentation as usual.

$$x_{c,f} \in \{0,1\} \quad \forall c \in C, f \in F \tag{3.8}$$

$$x_{c,f} \geq a_{r,f} \cdot \gamma(r,c) \quad \forall c \in C, r \in R, f \in F \tag{3.9}$$

$$\text{minimize} \quad \sum_{c \in C} \sum_{f \in F} x_{c,f} \tag{3.10}$$

While modelling room-assignment as an ILP seems to be pretty straight forward we will not generalize it with distance functions here, because we will later see a more sophisticated approach to this problem in Section 4.4 embedded into solving the floor-planning problem. There, for example, we do not require all floors to have the same size, but instead we are even able to model them far more realistic.

4 Floor Planning

In the preceding chapter we have seen how to assign rooms to floors. In this chapter we place the rooms within each floor as non-intersecting rectangles. For this purpose a floor is given as a simple orthogonal polygon P together with a matching hallway H. By matching we mean, that for each edge in P there is a corresponding parallel edge in H and vice versa. Note that P may also have holes, each requiring the hallway to have a cycle around it. We may assume without loss of generality that P and H are both axis-aligned.

The area we can use to place rooms in is $P \setminus H$ and can be subdivided into sections enclosed between two edges and sections between two corners. Therefore we refer to the first as edges of the usable area and to the latter as vertices of it. Such a subdivision of an example floor can be seen in Fig. 4.1. We let $w(e)$ denote the width of an edge e, measured as the distance between the two parallel edges defining e. Accordingly $l(e)$ denotes the length of the corresponding area of the subdivision and $s(e)$ its area.

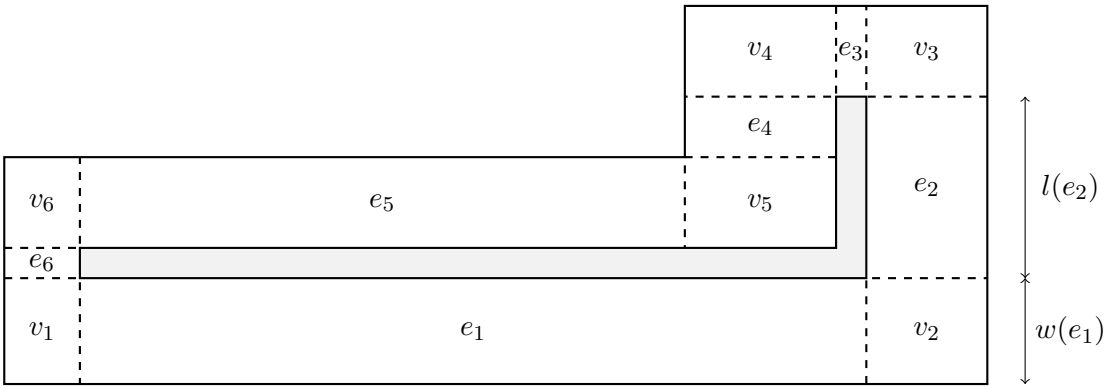


Fig. 4.1: The subdivision of a floor.

For simplicity we require every room to have a connection to both the hallway and the outer face. To be able to use those connections for doors and windows we require the length of the common edge to have at least a length of d. Since we required every room to be placed as a rectangle, they also need to be placed axis-aligned. The rooms placed within the area along an edge can therefore be seen as a simple linear subdivision of the rectangle into smaller rectangles. Therefore we do not need to care for the order of the rooms along an edge.

The rooms placed within a corner, however, need some additional attention. They also need a common edge with both the hallway and the outer face. To accomplish this they need to cover not only the entire area of that corner but also a small part of the

area belonging to one of the two adjacent edges. Therefore once we decide into which direction such a room grows, we consider the corner to belong to the adjacent edge. In Fig. 4.2 for example any room placed in v_1 needs to expand into the area of either e_1 or e_2 and the minimal length of that extension is again d. Since the heights of e_1 and e_2 might be different, the question of whether a room can be placed in v_1 also depends on the edge it will expend into. Therefore the size of the vertex $s(v)$ only counts the area these two realizations have in common. This approach does not discriminate between the two directions of inflection a corner can be of, because both can be handled identically.

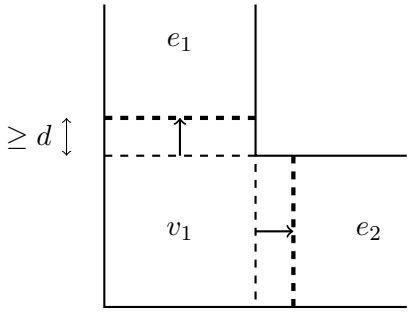


Fig. 4.2: Corner treatment.

In the following sections we will use the term neighbourhood in the following sense: The neighbourhood of an edge $N(e)$ is a set containing the two corners adjacent to the edge e. Accordingly we use $N(v)$ to denote the set containing the two edges adjacent to the corner v.

4.1 NP-hardness

Theorem 4.1. *Floor-planning is NP-hard.*

Proof. We show this by reducing instances of number-partitioning to floor-planning instances in polynomial time. That is, given a multi-set S of integers, find a partition into two subsets S_1 and S_2. We will now construct a floor-planning instance solving this problem.

We use an instance of floor-planning with the following properties: The floor is a rectangle with height $2 + \epsilon$, width $1.5|S| + 2$ and a centred hallway of width $|S|/2 + 2$ and height ϵ. The elements of S are used as the sizes of rooms and we add another four rooms of size $|S|/2 + 1$ each. This forces the corners of our floor plan to be occupied by the additional rooms since no other room is large enough to fill any corner for $0 < d \leq 1$.

The remaining areas above and below the hallway are then both of size $|S|/2$. The left and right edges both only have a width of ϵ and are therefore too small to be used by any room if we require $\epsilon < 1/a$. Since the size of the area along the upper and lower edges together is equal to the size needed by all rooms from the partitioning instance, that instance is solvable if and only if floor-planning is able to fit the rooms. □

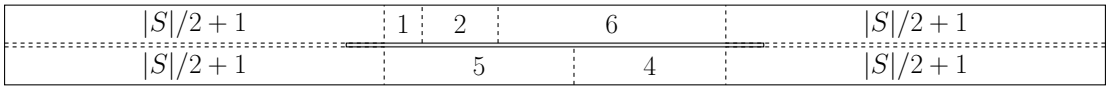


Fig. 4.3: Floor plan for partitioning the set $S = \{6, 5, 4, 2, 1\}$.

An example can be seen in Fig. 4.3.

In the following section we will see, that the feasibility of a solution can be checked through a polynomial number of constraints. Hence floor-planning $\in NP$ and therefore also NP-complete.

4.2 ILP

To solve floor-planning we first define a function $g\colon (R, E) \to \{0, 1\}$. Its purpose is to determine whether a room r can be placed along an edge e. If for example a small room is placed along an edge with high width the result would be a very thin rectangle. To circumvent such a placement, we are using an upper bound α for the aspect ratio of any rooms realization. This can be achieved through the following definition:

$$g(r, e) = \begin{cases} 1 & \alpha \geq \frac{s(r)}{w(e)^2} \geq \frac{1}{\alpha} \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

Assuming that no room is smaller than $\alpha \cdot d$ this also means we do not need to check whether rooms placed along an edge are long enough to have a door and a window. For rooms placed in a corner, however, we need to check that. For that purpose we define the function $q\colon (R, E, V) \to \{0, 1\}$ in a similar way to precompute whether a room r can be placed in the corner v expanding into the area of edge e. It needs to check whether the room is large enough to cover the whole corner and additionally enough of the adjacent edge to allow for a door or window, depending on the corner type:

$$q(r, e, v) = \begin{cases} 1 & s(r) \geq s(v) + w(e) \cdot d \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

Using these two function, we can now define the following ILP without an objective function to solve floor-planning. First Eq. 4.3 defines a variable for every combination of room and edge which is true if and only if the room is placed along that edge.

$$x_{r,e} \in \{0, 1\} \quad \forall r \in R, e \in E \tag{4.3}$$

In a similar way Eq. 4.4 defines a variable for every combination of room and corner, but also encoding which of the two adjacent edges the room will expend into.

$$y_{r,e,v} \in \{0, 1\} \quad \forall r \in R, e \in E, v \in N(e) \tag{4.4}$$

With Eq. 4.5 we make sure that every room is assigned to either one edge or one corner. In the latter case this includes the edge it belongs to.

$$\sum_{e \in E} \left[x_{r,e} + \sum_{v \in N(e)} y_{r,e,v} \right] = 1 \quad \forall r \in R \tag{4.5}$$

With Eq. 4.6 we make sure that no room is assigned to an edge it cannot be placed at according to the precomputed function g.

$$x_{r,e} + \sum_{v \in N(e)} y_{r,e,v} \leq g(r,e) \quad \forall r \in R, e \in E \tag{4.6}$$

Eq. 4.7 guarantees that every corner is occupied by one room at most and also encodes which edge it belongs to.

$$\sum_{r \in R} \sum_{e \in N(v)} y_{r,e,v} \leq 1 \quad \forall v \in V \tag{4.7}$$

We use Eq. 4.8 to check whether the rooms placed in corners can be placed in that corner according to the function q.

$$y_{r,e,v} \leq q(r,e,v) \quad \forall r \in R, e \in E, v \in N(e) \tag{4.8}$$

To ensure no edge gets overpacked, Eq. 4.9 checks for every edge whether the sum of the sizes of the rooms assigned to it fit into the given area. For this purpose the corners assigned to the edge need to be considered as well.

$$\sum_{r \in R} \left[x_{r,e} \cdot s(r) + \sum_{v \in N(e)} y_{r,e,v} \cdot \big(s(r) - s(v) \big) \right] \leq s(e) \quad \forall e \in E \tag{4.9}$$

4.3 Optimization

The ILP will simply produce any feasible solution, but, as in Chapter 3, the rooms still belong to work groups identified by the colour function $\gamma \colon R \to \{1, \ldots, |C|\}$. Hence the goal is once again to minimize the fragmentation of the set of rooms belonging to the same colour. We can achieve this by introducing an objective function to the ILP.

For this purpose we first pre-compute the values of the colour-function into a binary table $\gamma \colon (R, C) \to \{0, 1\}$ to tell whether a room r has colour c. It can simply be computed in the following way:

$$\gamma(r,c) = \begin{cases} 1 & \gamma(r) = c \\ 0 & \text{otherwise} \end{cases} \tag{4.10}$$

Then we need to adjust the ILP by adding another type of variable for each combination of edge and colour with Eq. 4.11.

$$z_{e,c} \in \{0,1\} \quad \forall e \in E, c \in C \tag{4.11}$$

Those variables are then constrained by Eq. 4.12 to be 1 if any of the rooms placed at the edge e or a corner belonging to that edge has the colour c.

$$\left(x_{r,e} + \sum_{v \in N(e)} y_{r,e,v}\right) \cdot \gamma(r,c) \leq z_{e,c} \quad \forall r \in R, e \in E, c \in C \tag{4.12}$$

By using Eq. 4.13 as our objective function, we can then minimize the number of edges a colour is present at over all colours. Note that this objective function is similar to that of BPMCF as seen in Chapter 3.

$$\text{minimize} \sum_{c \in C} \sum_{e \in E} z_{e,c} \tag{4.13}$$

This objective function does well in keeping the number of colours low within edges, but does not discriminate at all on a larger scale. The example shown in Fig. 4.4 for instance is optimal in terms of Eq. 4.13, but in hardly any others.

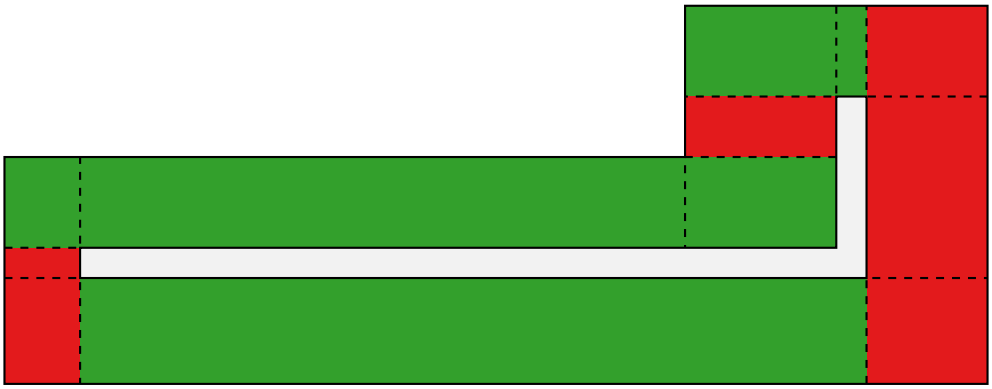


Fig. 4.4: An optimal solution with alternating colours.

To circumvent this behaviour, we need to design another measure. For this purpose we need a distance function $\delta\colon (E,E) \to \mathbb{R}$ to measure the distance of two edges. It can either be computed or given as another input. For computing it we propose the following algorithm.

The hallway can be reduced to a graph indicating walking-distances. Since the edges we use can be mapped to edges of that graph, we can use it to determine a rough distance measure through a shortest-distance algorithm. If a more sophisticated measure is needed, we can map the endpoints of each edge onto the graph to get a range on the graphs edge.

Using such a distance measure, we can design a new measure for how well colours are distributed in the floor. Eq. 4.14 sums the distances of edges with same colours, but is not a linear function and can therefore not be used in an ILP.

$$\text{minimize} \sum_{c \in C} \sum_{e_1 \in E} \sum_{e_2 \in E} z_{e_1,c} \cdot z_{e_2,c} \cdot \delta(e_1, e_2) \tag{4.14}$$

But since the variables we multiply are both boolean we are actually interested in the boolean *and* operation. To model it, we can use another type of variable for each pair of edges and colour in Eq. 4.15. Those are then constrained by Eq. 4.16 to be 1 if the edges e_1 and e_2 share the colour c.

$$u_{e_1,e_2,c} \in \{0,1\} \quad \forall e_1 \in E, e_2 \in E, c \in C \tag{4.15}$$
$$u_{e_1,e_2,c} + 1 \geq z_{e_1,c} + z_{e_2,c} \quad \forall e_1 \in E, e_2 \in E, c \in C \tag{4.16}$$

Using these variables, we can now define a new objective function for the same purpose but without multiplication in Eq. 4.17.

$$\text{minimize} \sum_{c \in C} \sum_{e_1 \in E} \sum_{e_2 \in E} u_{e_1,e_2,c} \cdot \delta(e_1, e_2) \tag{4.17}$$

Using it, we can finally get a solution like in Fig. 4.5 for the same input from Fig. 4.4.

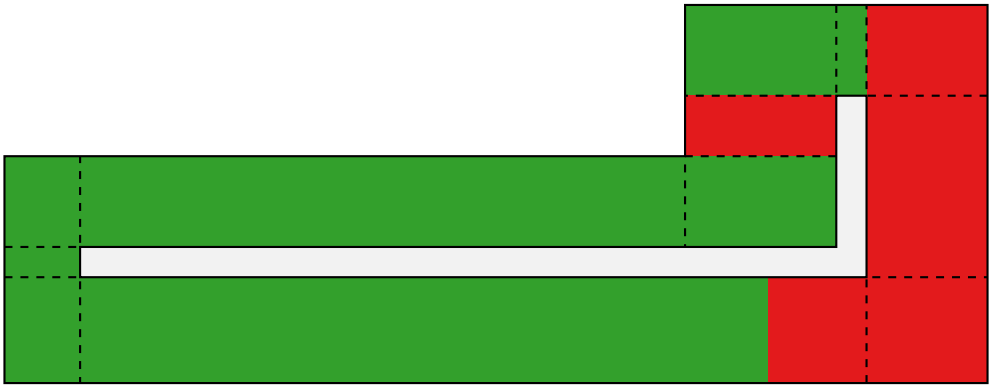


Fig. 4.5: A less fragmented solution.

However, Eq. 4.17 does not distinguish between the solutions shown in Figs. 4.5 and 4.6. In order to circumvent getting the latter solution, we need to include the corners into our distance measure.

For this purpose we define another type of variable for each combination of corner and colour in Eq. 4.3 and split up Eq. 4.12 into Eqs. 4.19 and 4.20.

$$z_{v,c} \in \{0,1\} \quad \forall v \in V, c \in C \tag{4.18}$$
$$x_{r,e} \cdot \gamma(r,c) \leq z_{e,c} \quad \forall r \in R, e \in E, c \in C \tag{4.19}$$
$$y_{r,e,v} \cdot \gamma(r,c) \leq z_{v,c} \quad \forall r \in R, v \in V, e \in N(v), c \in C \tag{4.20}$$

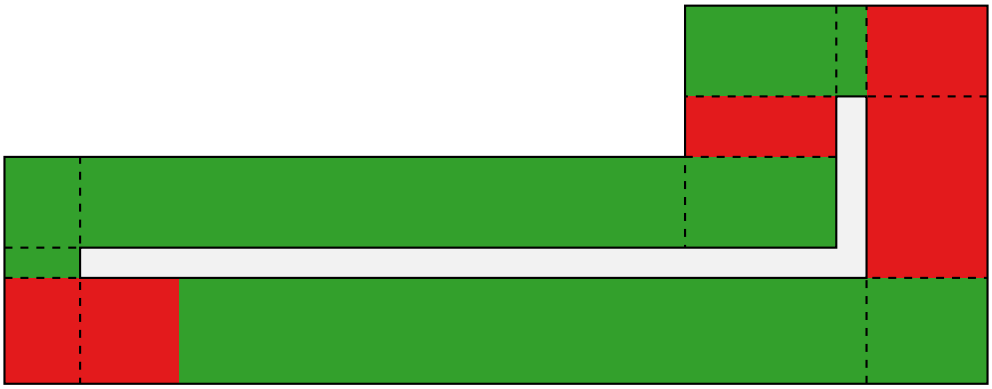


Fig. 4.6: A less likeable solution.

Additionally using a distance function $\delta(E, V) \to \mathbb{R}$, we can define Eq. 4.21 as our final objective function. Note that it again only uses multiplication on boolean values and can hence be linearised automatically like demonstrated before.

$$\text{minimize} \sum_{c \in C} \sum_{e_1 \in E} \left[\sum_{e_2 \in E} z_{e_1,c} \cdot z_{e_2,c} \cdot \delta(e_1, e_2) + \sum_{v \in V} z_{e_1,c} \cdot z_{v,c} \cdot \delta(e_1, v) \right] \tag{4.21}$$

4.4 Multiple Floors

While we designed the ILP to place rooms within a single floor it is actually capable of solving multi-floor setups. In the given form of the ILP, the neighbourhood relation between edges and corners already does not consider whether all of them are connected. For example the floor plan in Fig. 4.7 is already solvable in the current definition of the ILP.

The rooms in this example are physically connected through the hallway, but the ILP does not actually have that information. It only knows about two independent cycles of edges it can use to place rooms in. The situation for the ILP is therefore already much like having two independent floor plans.

The only exemption is the distance function. In the courtyard example from Fig. 4.7 the distance of two edges can still be computed like before. For edges not sharing a common hallway we need to use the distance of their floors instead. Since the ILP itself already needs the distance function as a pre-computed matrix, we can compute the distances when creating the ILP. Since that matrix only has a quadratic number of entries we can compute all entries independently in polynomial time. To use the distance of floors as the distance of edges we just need to use a proper scaling factor for the two measures.

While we are now able to measure the distance of any two edges, the actual distance might differ a lot. This is due to an edge's distance to the staircase. To include it into our model we first need to make sure, the ILP can handle it. Fig. 4.8 shows a floor plan with

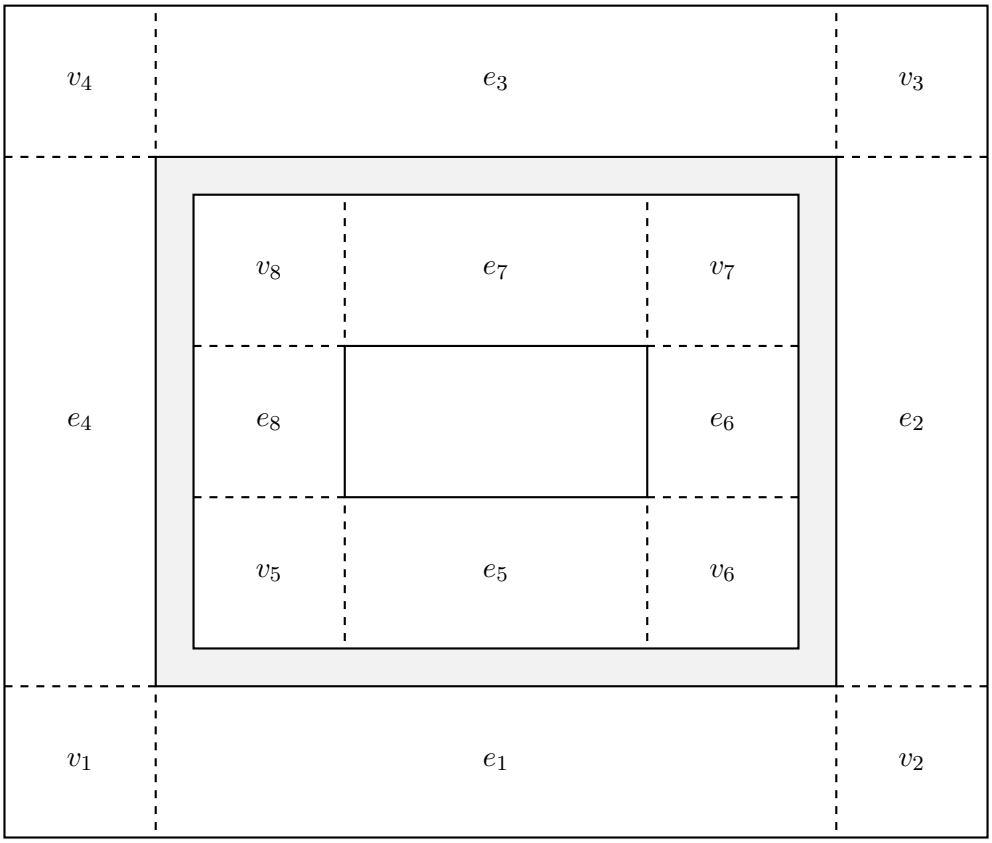


Fig. 4.7: Floor plan with courtyard.

the area occupied by the staircase crossed out. The corresponding edge is divided into two partial edges. Up to this point every edge and corner always had two neighbours, but the ILP does not actually depend on this property and hence the two half-edges do not pose a problem. Note that we do not require the blocked area for the staircase to be the same in every floor, nor do we at all require the floors to be similar. However, what we need is the distance function between floors to measure the distance within the staircase. We may also use the same method to incorporate an elevator or to block areas from usage by the ILP.

With the staircase incorporated into our model we can compute the distance of two edges from different floors in a more sophisticated way. First we compute the distance of both edges to the staircase in their floor and sum them up together with the distance of the two floors involved. Now a colour fragmented across at least two floors contributes less to the objective function if the edges it is placed in are closer to the staircase of their floors.

The generalization of the floor-planning ILP is also able to solve room-assignment instances. We simply need to model the floors with a single edge and set the distance of that edge to the staircase to zero. For the distance of two edges we can therefore simply

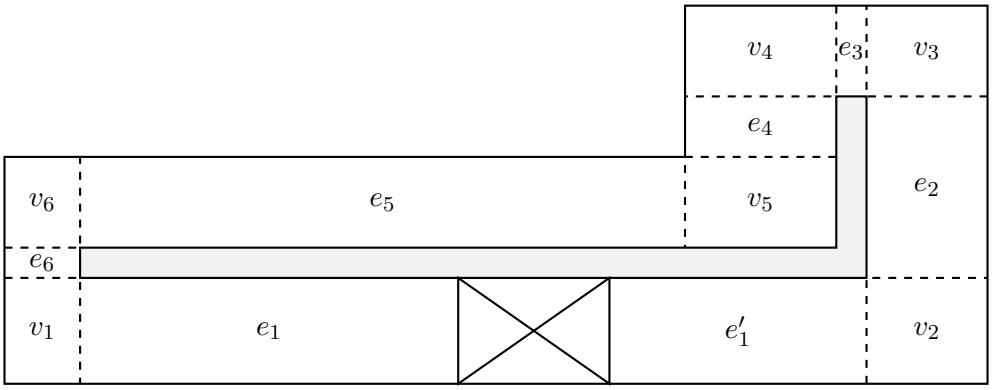


Fig. 4.8: Floor plan with staircase.

use the difference in level of their floors like before. However, we may not want to use this insight since we are already able to solve both problems in one step as seen in this section.

5 People Assignment

The problems we defined so far are designed to help in the pre-construction phase of buildings. However, requirements may change over time since work groups can change in size, cease or arise. While our goals may stay the same after a change of requirements, we are further constrained since the construction is already completed.

Therefore this chapter is about a slightly different problem, where we want to place people belonging to work groups into already existing rooms. We define it in the following way:

Problem PEOPLE-ASSIGNMENT

Item: A set P of n people, a size function $s\colon P \to \mathbb{N}$, a function $\gamma\colon P \to \{1, \ldots, |C|\}$ assigning each person a colour (group), a set R of rooms, a size function $s\colon R \to \mathbb{N}$ and a distance function $\delta\colon (R, R) \to \mathbb{R}$.

Question: How to find an assignment of people to rooms $a\colon P \to R$, that is a valid bin-packing solution while optimizing an objective function $\text{cost}\colon (a, \gamma, \delta) \to \mathbb{R}$?

We use the following cost-function here:

$$\text{minimize } \text{cost}(a, \gamma, \delta) = \sum_{c \in C} \sum_{r_1 \in R} \sum_{r_2 \in R} x_{c,r_1} \cdot x_{c,r_2} \cdot \delta(r_1, r_2) \tag{5.1}$$

$$x_{c,r} = \begin{cases} 1 & \bigvee_{p \in P} \left[\Big(a(p) = r\Big) \wedge \Big(\gamma(p) = c\Big) \right] \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

The distance of two rooms can be precomputed in the same way as we did for edges in Section 4.4.

Theorem 5.1. *People-assignment is NP-hard.*

Proof. We can reduce bin-packing to people-assignment by modelling all the items as people of the same group and the bins as equidistant rooms. Solving such an instance also solves the decision version of bin-packing. □

Theorem 5.2. *People-assignment is NP-hard to approximate.*

Proof. An approximate solution of any quality is still a valid bin-packing solution. □

Theorem 5.3. *People-assignment can be solved trough floor-planning.*

Proof. Model each of the rooms in an instance of people-assignment as a floor that consists of only one edge. The area of that edge is determined by the size of the room we used to create it and we do not introduce aspect-ratio bounds. The people are then modelled as rooms in the floor-planning instance, including their colour-function. □

5.1 ILP

Instead of using floor-planning to solve people-assignment we can define a dedicated ILP. However, it works in a similar fashion. First we model valid assignments:

$$a_{p,r} \in \{0,1\} \quad \forall p \in P, r \in R \tag{5.3}$$

$$\sum_{r \in R} a_{p,r} = 1 \quad \forall p \in P \tag{5.4}$$

$$\sum_{p \in P} a_{p,r} \cdot s(p) \leq s(r) \quad \forall r \in R \tag{5.5}$$

For the optimization part we again need auxiliary variables to circumvent the multiplication within our objective function:

$$x_{c,r} \in \{0,1\} \quad \forall c \in C, r \in R \tag{5.6}$$

$$a_{p,r} \cdot \gamma(p,c) \leq x_{c,r} \quad \forall p \in P, c \in C, r \in R \tag{5.7}$$

$$u_{c,r_1,r_2} \in \{0,1\} \quad \forall c \in C, r_1 \in R, r_2 \in R \tag{5.8}$$

$$u_{c,r_1,r_2} + 1 \geq x_{c,r_1} + x_{c,r_2} \quad \forall c \in C, r_1 \in R, r_2 \in R \tag{5.9}$$

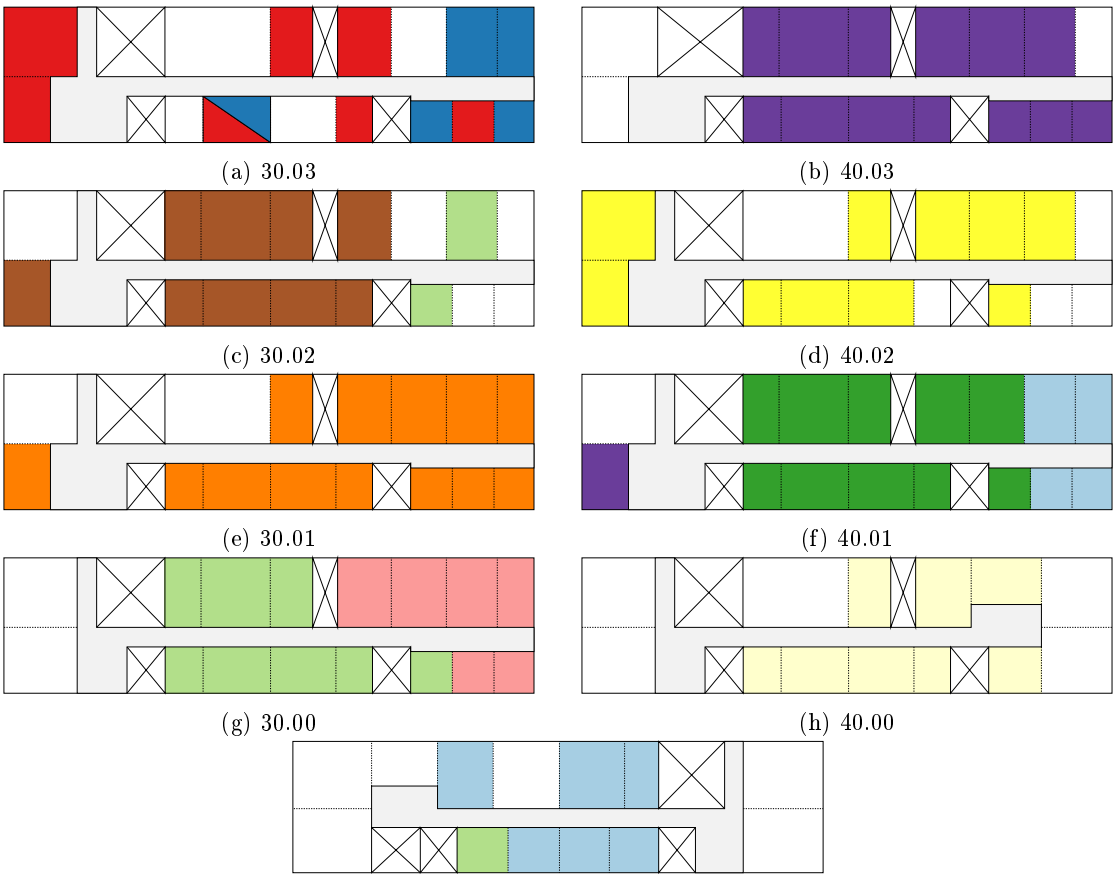
$$\text{minimize} \quad \sum_{c \in C} \sum_{r_1 \in R} \sum_{r_2 \in R} u_{c,r_1,r_2} \cdot \delta(r_1, r_2) \tag{5.10}$$

Furthermore some people might necessarily get a room of their own, while others might not. This information is required as an input function $t\colon R \to \{0,1\}$. To let the ILP plan accordingly we adjust the constraints counting the space within a room from Eq. 5.5 in the following way:

$$\sum_{p \in P} a_{p,r} \cdot s(p) \cdot (1 - t(r)) + s(r) \cdot t(r) \leq s(r) \quad \forall r \in R \tag{5.11}$$

6 Benchmark

To evaluate how well the proposed algorithms work we use real data from the Institute of Mathematics at the University of Würzburg. They are currently structured into 10 chairs, which we will use as colours, along with another colour for rooms associated with the institute itself. The rooms they use are currently distributed among 9 floors in 3 different buildings. Assigning each room the colour corresponding to the people working in it results in the schematic seen in Fig. 6.1. Note that short-term personnel was omitted due to being assigned to shared rooms on an institute basis.



(a) 30.03 (b) 40.03 (c) 30.02 (d) 40.02 (e) 30.01 (f) 40.01 (g) 30.00 (h) 40.00 (i) 31.00

Fig. 6.1: Current floor plan

The floors are identified by two digits for the building and two digits for the level in that building, separated by a dot. While the floors are all slightly different, they follow

the same general structure. The blocked areas in the lower part are stairs and those in the upper part are service and rest rooms.

All ILPs tested in this chapter have been built using OPL and solved with CPLEX version 12.10 in the following environment:

- Windows 10 1903,
- Intel i7-9700 @4.4GHz and
- 32GB DDR4 RAM @2133MHz

6.1 People Assignment

Here we use the ILP defined in Section 5.1 to find a distribution of the people working at the institute to the same floors they already use and compare it with the actual distribution. Note that most of the unassigned rooms are used as seminar or meeting rooms and therefore occupied but not associated with a chair. Those rooms are exempt from the test run described in the this section.

Since the structure of the floors is already given, we can compute distances between rooms in the following way. First we model the hallway within each floor as a simple line and project each rooms door onto that line. Distances of rooms within a floor can then be computed as the distance along that line. For rooms in different floors, distances from and to a staircase in addition to a penalty for the distance of the floors are used. Since there are two staircases, we compute the distance for using either and use the better result. For rooms within different buildings we use the distances of both rooms to the main door of their building (within the left staircase for building 30 and 40, and within the right staircase for building 31) along with an extra penalty for the distance of the buildings.

After modelling the rooms and distances between them, we still need to model the people we want to assign. According to the current plan we identified 3 major groups of people working in the institute. Their requirements are also formulated in regard to the current plan:

- Secretaries and other administrative personnel get their own room of at least $15m^2$,
- long-term personnel get their own room of at least $18m^2$ and
- other personnel may be assigned to a shared room which has at least $8m^2$ per person assigned.

Using these groups of personnel we can formalize the requirement for each chair by classifying all current personnel and vacancies. The result can be seen in Table 6.1, where chair 0 is the institute.

chair	0	1	2	3	4	5	6	7	8	9	10
$15m^2$	3	1	1	1	1	1	1	1	1	1	1
$18m^2$	2	3	3	1	2	5	3	3	3	4	3
$8m^2$	3	4	8	5	9	11	16	8	4	7	5

Tab. 6.1: Demand of different sized spaces per chair.

Solving the ILP, however, turned out to be very computationally challenging. The given example could only be solved within 12 hours after weeks optimizing CPLEX-parameters. The optimal solution obtained has a cost of 6290.04 and can be seen in Fig. 6.2.

While the solution obtained still has a few flaws, its quality is already comparable to that of the plan currently in use. Avoiding the clustering in floor 40.01 for instance, might be solvable by using a higher penalty for the distance of floors.

Compared to finding an optimal solution it turned out to be relatively easy to find very

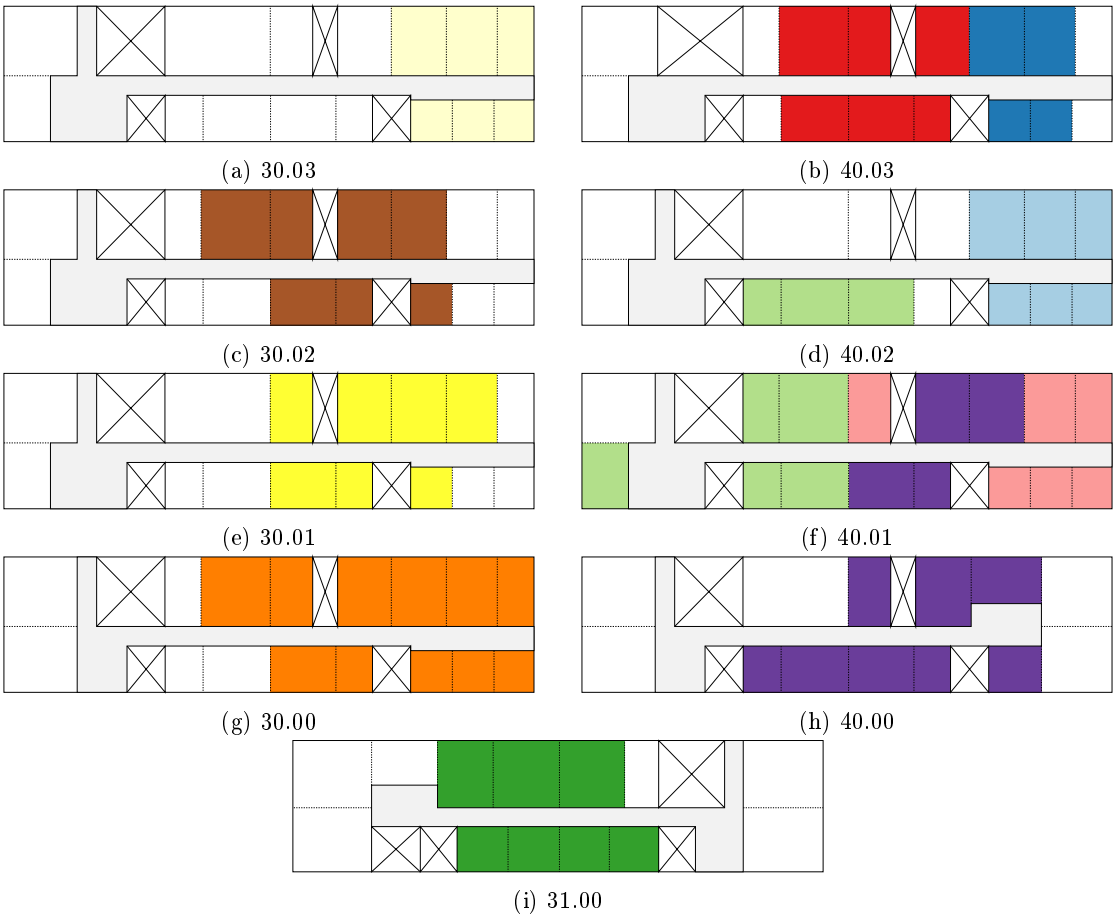
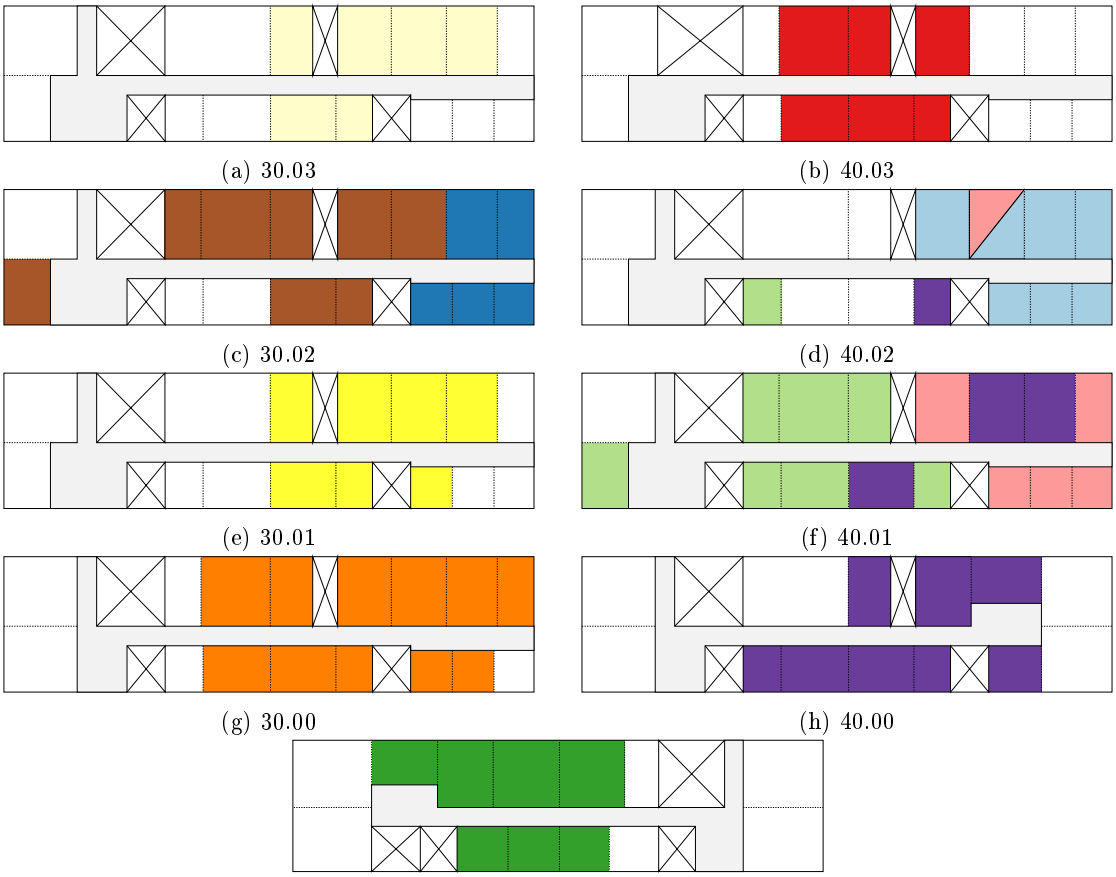


Fig. 6.2: Optimal new assignment to current floor plan

good solutions fast. Therefore Fig. 6.3 additionally shows the best solution obtainable within a more reasonable time-limit of one hour, while not using knowledge about the instance from previous attempts. The solution has a cost of 7194.96 and is hence already relatively close to OPT, while using only a fraction of the time.



(a) 30.03 (b) 40.03 (c) 30.02 (d) 40.02 (e) 30.01 (f) 40.01 (g) 30.00 (h) 40.00 (i) 31.00

Fig. 6.3: New assignment to current floor plan with 1h time-limit

6.2 Floor Planning

To test our floor-planning algorithms, we first need to generate unfinished floor plans. We could do so by simply removing the walls separating the rooms in Fig. 6.1, but this approach results in floors consisting only of edges but no corners. To properly test the capabilities of floor-planning, we therefore use the plan given in Fig. 6.4 for every floor. For simplicity we also leave out all additional rooms that were not part of testing people-assignment either.

In order to be able to compare the performance of the ILP defined in Sections 4.2 and 4.3 for the whole instance and separately for each floor after using the bi-criteria

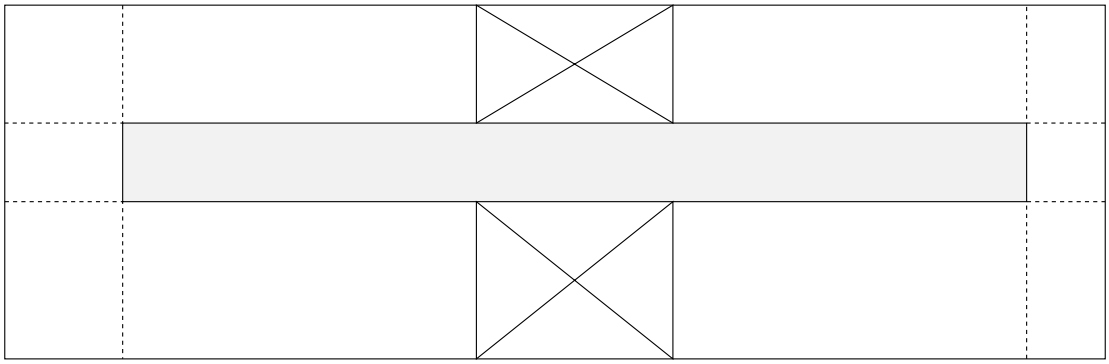


Fig. 6.4: Floor plan of test data.

approximation from Section 3.4, we may not use a multi-building setup. Therefore we use one building with 9 identical floors according to Fig. 6.4. Distances between edges and vertices are measured along the hallway much like in Section 6.1 but with less precision due to the fact, that doors are not yet placed.

For the rooms required for each chair, we still use the actual demand like formulated in Section 6.1. Since we already forced secretary and long-term personnel into rooms of their own there, we can simply translate their demand to a demand of a rooms of the same size. For other personnel we may do the same, but have the option to aggregate multiple such rooms when placed along the same edge. Instead it would also be possible to predefine into how many rooms of how many people each they should be separated.

With this testing-approach of using the actual demand, but a hypothetical building, we simulate the use-case of testing a design for a new building where an existing institute shall be moved to.

6.2.1 ILP

Solving the ILP defined in Sections 4.2 and 4.3 with the described example proved impractical. While the tested instance is smaller than the one solved for people-assignment, it might be computationally harder due to having a higher density of constraints.

Since solving the problem to optimality not being completed within a week with optimized parameters, Fig. 6.5 instead shows the best solution computed within a time-limit of one hour and without using knowledge about the instance from previous attempts. This solution of value 4738, however, still looks quite far from optimal and therefore another solution obtained with a time-limit of 12 hours is displayed for comparison. The latter has a value of 2313 and is hence reasonably closer to the optimal solution.

While it is easy to see, that the 12-hour solution is less fragmented than the 1-hour solution, it still does not look like a plan good enough to actually be used in practice. Whether this is due to not being optimal or due to the model remains an open question.

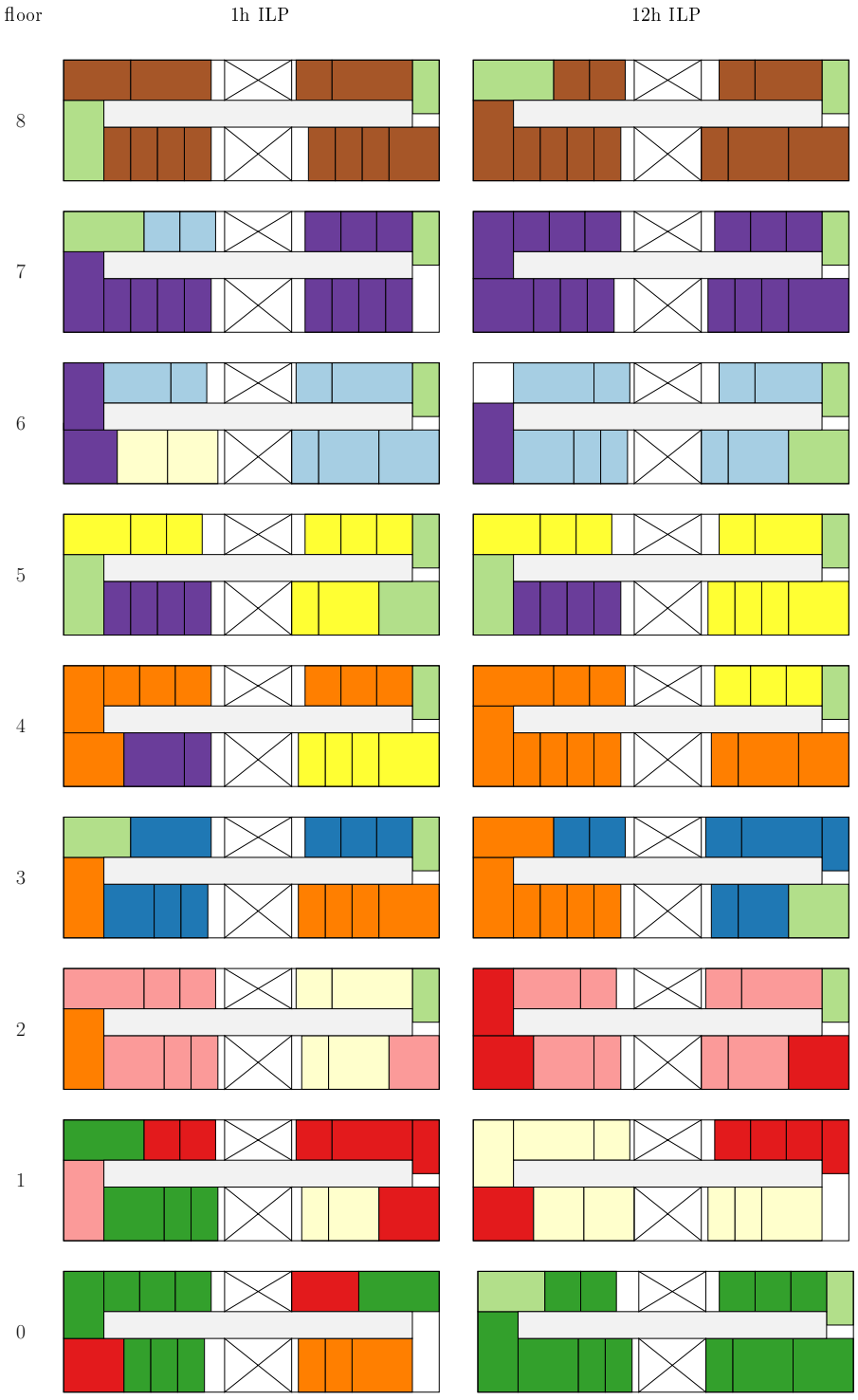


Fig. 6.5: New floor plans: first column 1h ILP, second column 12h ILP

6.2.2 Approximation

In this section we first solve an instance of room-assignment with the bi-criteria approximation from Section 3.4 and then use the floor-planning ILP separately for each floor. While this approach is only a heuristic when viewing the solved problem as a whole, it might still prove useful due to faster solving.

For the bi-criteria approximation we first need to reduce the floors to one-dimensional bins. The capacity of a floor according to Fig. 6.4 is $171m^2$. Then we need to relax the room-assignment instance to an instance of area-distribution by summing up the area-requirements for every chair. The result can be seen in Table 6.2

chair	0	1	2	3	4	5	6	7	8	9	10
area	105	101	133	73	123	193	197	133	101	143	109

Tab. 6.2: Total area required per chair.

The next step is finding a nice sequence which we can use as a 2-approximation for the area-distribution instance as shown in Theorem 2.7. We use the nice sequence shown in Fig. 6.6 which is arguably not a very good choice, but nonetheless a 2-approximative solution. The area each chair got assigned to in which floor can be seen in Table 6.3.

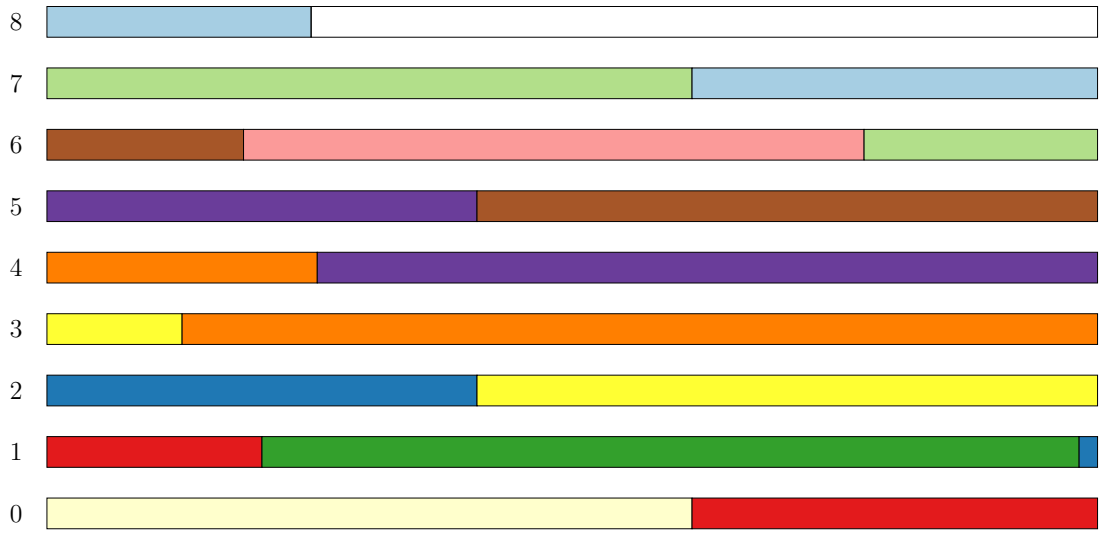


Fig. 6.6: Nice sequence distributing the area requirement of the chairs to the floors.

Nonetheless, we use the given nice sequence as input for the improved heuristic described in Section 3.4.2. Since no chair got their space scattered into more than two floors, assigning the rooms to the floors is straight forward. First fill the lower floor greedily with rooms in descending order of size and then put all remaining rooms into the other floor. This results in the assignment of rooms to floors shown in Table 6.4.

Most notably the provided area is exceeded by at most $2m^2$, giving us a scaling factor of $\beta = 173/171$, while the upper bound was only $\beta \leq 1+(k-1)/m = 208/171$. Using the

chair/floor	0	1	2	3	4	5	6	7	8	9	10
8	0	0	0	0	0	0	0	0	0	0	43
7	0	0	0	0	0	0	0	0	0	105	66
6	0	0	0	0	0	0	0	32	101	38	0
5	0	0	0	0	0	0	70	101	0	0	0
4	0	0	0	0	0	44	127	0	0	0	0
3	0	0	0	0	22	149	0	0	0	0	0
2	0	0	0	70	101	0	0	0	0	0	0
1	0	35	133	3	0	0	0	0	0	0	0
0	105	66	0	0	0	0	0	0	0	0	0

Tab. 6.3: Area-distribution per chair.

floor	chair	0	1	2	3	4	5	6	7	8	9	10	sum
8	$15m^2$											1	$47m^2$
	$18m^2$											0	
	$8m^2$											4	
7	$15m^2$										1	0	$169m^2$
	$18m^2$										2	3	
	$8m^2$										7	1	
6	$15m^2$								0	1	0		$169m^2$
	$18m^2$								0	3	2		
	$8m^2$								4	4	0		
5	$15m^2$							0	1				$173m^2$
	$18m^2$							0	3				
	$8m^2$							9	4				
4	$15m^2$						0	1					$173m^2$
	$18m^2$						0	3					
	$8m^2$						6	7					
3	$15m^2$					0	1						$169m^2$
	$18m^2$					0	5						
	$8m^2$					3	5						
2	$15m^2$				1	1							$172m^2$
	$18m^2$				1	2							
	$8m^2$				5	6							
1	$15m^2$		1	1	0								$172m^2$
	$18m^2$		0	3	0								
	$8m^2$		3	8	0								
0	$15m^2$	3	0										$167m^2$
	$18m^2$	2	3										
	$8m^2$	3	1										

Tab. 6.4: Distribution of rooms to floors per chair.

given distribution of rooms to floors, we now solve a separate instance of floor-planning for each floor using the ILP defined in Sections 4.2 and 4.3. The result can be seen left in Fig. 6.7.

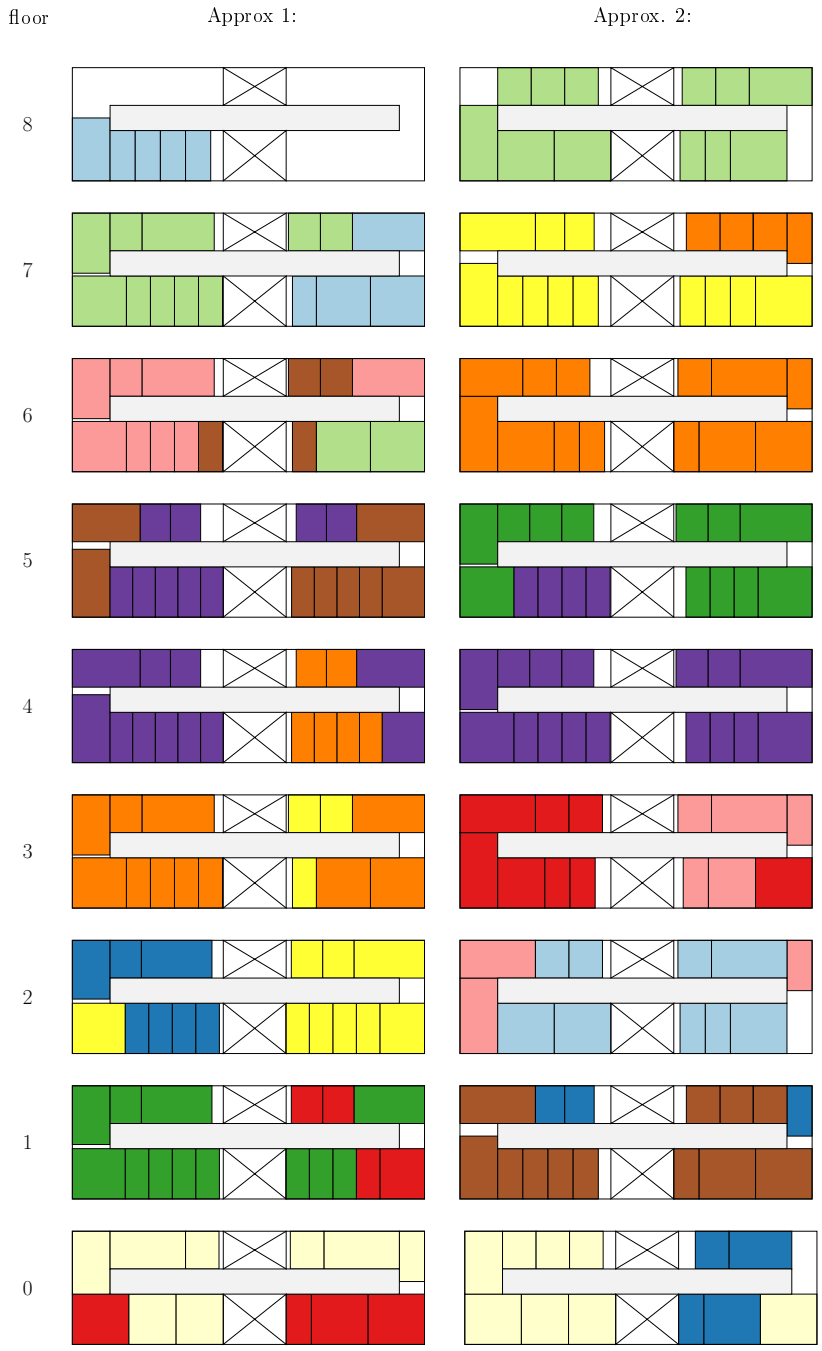


Fig. 6.7: New floor plans: first column with arbitrary nice sequence, second column with chosen nice sequence.

While every floor was solved in less than a second, it was not always possible to directly use the scaling factor from the bi-criteria approximation within the room-assignment ILP. As can be seen in the resulting floor plan, most floors do not use the edge between the two right corners. This is due to the constraints resulting in a required room of either 8 to $12m^2$ if using a corner, or of at most $4m^2$ otherwise. When using a scaling factor slightly above 1, no such room exists and hence most floors cannot use the space. Along with similar, but more complicated constraints this results in the necessity of a higher scaling factor for most floors. Floors 0 and 8 did not need to scale their rooms, but the other floors needed up to $\beta = 190/171$ for a feasible solution, which is, however, still less than the theoretical bound for room-assignment alone.

Another feature of the solution is that some parts might look non-optimal like in floor 2. While there are several reasons for such behaviour, in this case it is due to the ILP running with only slightly more available space than required space which leads to feasibility being the major task instead of optimizing the cost-function. Another reason is how we modelled the third (smallest) type of room. While we used them separately, in reality most of them are grouped into double or triple rooms. In the tested way the ILP loses the ability to put such rooms into corner areas since any single room is too small, but with pre-grouped rooms the ILP loses flexibility in distributing those rooms. Since the model cannot decide this question on its own it is too simplified to capture this use-case properly.

In floor 4 and 5 we can also see another problem concerning the greedy approach of assigning the rooms of a colour to the floors. It leads to most of the larger rooms of any colour to be within the same floor, while most of the smaller rooms get assigned to another floor. Therefore the large rooms within a floor may have all the same colour, while the floor is not single-coloured. Since small rooms may not be placed in corners, this leads to suboptimal behaviour.

This approach already works quite well compared to the ILP if considering, that it runs at least 5 orders of magnitude faster on the used example. However, putting some more work into carefully selecting the used nice sequence might result in far better floor plans without needing a lot more runtime. Also using a nice sequence where the empty space is distributed among the floors might not need to scale rooms down.

To test whether this claim holds, we use the same approach and simply use a chosen nice sequence instead of an arbitrary one. Fig. 6.8 shows the used nice sequence, which has less fragmented colours as well as a better distribution of the empty space compared to Fig. 6.6. The corresponding area-distribution is displayed in Table 6.5 and the results of the improved heuristic for the distribution can be seen in Table 6.6.

Using this assignment of rooms to floors again to solve an ILP-instance for each floor independently, results in the floor-plan displayed on the right in Fig. 6.7. Floor 4 and 5 still needed a scaling factor of $\beta = 179/171$ due to their large number of small rooms. All other floors were solvable without scaling which already shows how much of a difference an informed choice of the nice sequence makes. The resulting floors also appear less fragmented since the ILP has more empty space it can use for optimizing the cost-function.

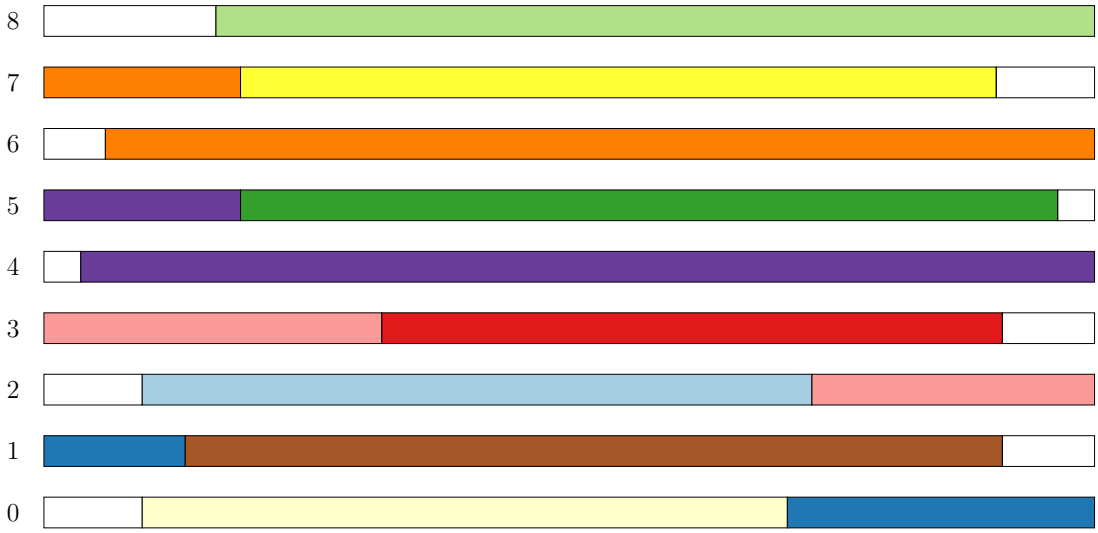


Fig. 6.8: A better nice sequence distributing the area requirement of the chairs to the floors.

chair/floor	0	1	2	3	4	5	6	7	8	9	10
8	0	0	0	0	0	0	0	0	0	143	0
7	0	0	0	0	123	32	0	0	0	0	0
6	0	0	0	0	0	161	0	0	0	0	0
5	0	0	133	0	0	0	32	0	0	0	0
4	0	0	0	0	0	0	165	0	0	0	0
3	0	101	0	0	0	0	0	0	55	0	0
2	0	0	0	0	0	0	0	0	46	0	109
1	0	0	0	23	0	0	0	133	0	0	0
0	105	0	0	50	0	0	0	0	0	0	0

Tab. 6.5: Area-distribution per chair of new nice sequence.

floor	chair	0	1	2	3	4	5	6	7	8	9	10	sum
8	$15m^2$										1		$143m^2$
	$18m^2$										4		
	$8m^2$										7		
7	$15m^2$					1	0						$155m^2$
	$18m^2$					2	0						
	$8m^2$					9	4						
6	$15m^2$						1						$161m^2$
	$18m^2$						5						
	$8m^2$						7						
5	$15m^2$			1				0					$165m^2$
	$18m^2$			3				0					
	$8m^2$			8				4					
4	$15m^2$							1					$165m^2$
	$18m^2$							3					
	$8m^2$							12					
3	$15m^2$		1							1			$158m^2$
	$18m^2$		3							1			
	$8m^2$		4							3			
2	$15m^2$									0		1	$153m^2$
	$18m^2$									2		3	
	$8m^2$									1		5	
1	$15m^2$				0				1				$157m^2$
	$18m^2$				0				3				
	$8m^2$				3				8				
0	$15m^2$	3			1								$154m^2$
	$18m^2$	2			1								
	$8m^2$	3			2								

Tab. 6.6: Distribution of rooms to floors per chair for new nice sequence.

6.3 CPLEX-Remarks

The two ILPs for people-assignment and floor-planning implemented and tested in this chapter behaved pretty similar. Since their formulations have similarities, this does not surprise much, although people-assignment is less constrained and tested on a larger instance. In this section we want to give some brief remarks on how CPLEX works with them.

- CPLEX handles both ILPs in their simpler IQP formulations and automatically linearises the multiplication of boolean variables.
- The branch & bound algorithm used in CPLEX generates a huge search tree with a similarly huge amount of memory consumption if not run with proper parameters.
- Using an estimate for OPT as an upper bound within the search, can dramatically reduce memory and time consumption through pruning, but since it requires additional knowledge about the instance, we did not use it for benchmark.
- Finding good solutions and working towards a proof of optimality are both done in parallel, but optimizing them requires mostly contrary parameter settings.
- Finding OPT can therefore be done by two separate runs, where the first is optimized to find good solutions in order to provide a good upper bound on OPT for the second run, optimized in proving optimality. If running the second without the first, the search-algorithm cannot prune much of the search-tree and therefore requires several hundred GB of memory.
- The test-data does involve a few rooms/people with identical constraints which results in symmetries within the generated ILPs. This could be exploited with a different formulation of the ILPs where identical entities are grouped and assigned with integer instead of boolean decision variables. While this most likely results in a model that can be solved much faster, it does only work if such groups exist in the instance to solve.

7 Conclusion

In this work we defined, implemented and tested two different approaches to automating the process of floor-planning. One approach formulates the whole problem as an ILP, while the other first splits the problem into sub-problems for each floor and then solves them individually with the same ILP. While both approaches work in principle, there is still additional work required to use them in practice.

For the ILP, further work into optimizing the runtime is necessary. However, with the current formulation and parameters it may already be used for instances with only a few floors. Depending on the instance and goals, different objective functions may be necessary. To avoid even stronger implications on the runtime, only convex functions should be considered. Due to the tested instance not being solved optimally, it also remains unclear whether choosing a higher penalty for the distance of floors might be able to improve results.

The algorithmic approach looks promising, but also needs some further adjustments to produce usable floor-plans. For example, it might prove useful not to fill the first bin in the heuristic greedily, but with an approach that distributes large and small rooms more balanced among the floors. This would also reflect the nature of subgroups within the chairs better, which we did not consider at all yet. Furthermore we already tested using nice sequences with additional desirable properties which proved very successful. However, good selection criteria still need to be identified and formalized along with an algorithm for automatically finding them, while avoiding to brute-force all possibilities.

In general the level of abstraction used to formulate the given problem definition was higher than anticipated but necessary for a first proof of concept. Further adjustments to the model might therefore still be necessary for solving instances in a more sophisticated way. For instance, we modelled the chairs as a simple assignment of people or room requirements to colours. However, the real structure does include multiple layers, for example with different professorships within the same chair. There are also more constraints like a secretary's and the associated professor's rooms might have to be placed with a common wall to allow for a door in between. Some people might also work for multiple groups which can not be represented in the current model. While we may adjust the models level of detail to better cope with such cases, it will not likely be able to fully represent all details of real instances. A lower level of abstraction than the one we used here, might, however, still be desirable and achievable.

Bibliography

[BCM19] David Bergman, Carlos Cardonha, and Saharnaz Mehrani: Binary decision diagrams for bin packing with minimum color fragmentation. In Louis-Martin Rousseau and Kostas Stergiou (editors): *Proceedings of the Sixteenth International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*, volume 11494 of *Lecture Notes in Computer Science*, pages 57–66. Springer, 2019. https://doi.org/10.1007/978-3-030-19212-9_4.

[Bor13] Andreas Bortfeldt: A reduction approach for solving the rectangle packing area minimization problem. *European Journal of Operational Research (EJOR)*, 224(3):486–496, 2013. https://doi.org/10.1016/j.ejor.2012.08.006.

[GJ79] Michael R. Garey and David S. Johnson: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[Gra66] Ronald L. Graham: Bounds for certain multiprocessing anomalies. *The Bell System Technical Journal (BSTJ)*, 45(9):1563–1581, 1966. https://doi.org/10.1002/j.1538-7305.1966.tb01709.x.

[HK13] Eric Huang and Richard E. Korf: Optimal rectangle packing: An absolute placement approach. *Journal of Artificial Intelligence Research (JAIR)*, 46:47–87, 2013. https://doi.org/10.1613/jair.3735.

[HK14] André Hamacher and Eirik Kjølsrud: Growing floor plans and buildings - generation and optimisation through algorithms and synthetic evolution. In Alberto T. Estévez (editor): *Proceedings of the Second International Conference of Biodigital Architecture and Genetics (BIODIG)*, 2014. https://researchgate.net/publication/311455480.

[HYC11] Wenqi Huang, Tao Ye, and Duanbing Chen: Bottom-left placement theorem for rectangle packing. *Computing Research Repository (CoRR)*, abs/1107.4463, 2011. http://arxiv.org/abs/1107.4463.

[IL03] Claudia Iturriaga and Anna Lubiw: Elastic labels around the perimeter of a map. *Journal of Algorithms*, 47(1):14–39, 2003. https://doi.org/10.1016/S0196-6774(03)00004-X.

[Kar72] Richard M. Karp: Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher (editors): *Proceedings of a symposium on the Complexity of Computer Computations*, volume 40 of *The IBM Research*

Symposia Series, pages 85–103. Plenum Press, New York, 1972. https://doi.org/10.1007/978-1-4684-2001-2_9.

[KK83] Narenda Karmarker and Richard M. Karp: The differencing method of set partitioning. Technical Report UCB/CSD-83-113, EECS Department, University of California, Berkeley, 1983. http://www2.eecs.berkeley.edu/Pubs/TechRpts/1983/6353.html.

[Kne11] Katja Knecht: Generierung von Grundriss-Layouts mithilfe von Evolutionären Algorithmen und K-dimensionalen Baumstrukturen. Technical report, Chair of Computer Science in Architecture, University of Weimar, Germany, 2011. https://doi.org/10.25643/bauhaus-universitaet.2666.

[Kor03] Richard E. Korf: Optimal rectangle packing: Initial results. In Enrico Giunchiglia, Nicola Muscettola, and Dana S. Nau (editors): *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS)*, pages 287–295. AAAI, 2003. http://www.aaai.org/Library/ICAPS/2003/icaps03-029.php.

Erklärung

Hiermit versichere ich die vorliegende Abschlussarbeit selbstständig verfasst zu haben, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben, und die Arbeit bisher oder gleichzeitig keiner anderen Prüfungsbehörde unter Erlangung eines akademischen Grades vorgelegt zu haben.

Würzburg, den 4. September 2020

.....
Felix Klesen