Optimal Carpet Cutting

Andreas Schutt[†], Peter J. Stuckey[†], and Andrew R. Verden[⋆]

[†] National ICT Australia, Department of Computer Science & Software Engineering, The University of Melbourne, Victoria 3010, Australia
{aschutt,pjs}@csse.unimelb.edu.au
[⋆] National ICT Australia, Kensington, NSW 2052, Australia
andrew.verden@nicta.com.au

Abstract. In this paper we present a model for the carpet cutting problem in which carpet shapes are cut from a rectangular carpet roll with a fixed width and sufficiently long length. Our exact solution approaches decompose the problem into smaller parts and minimise the needed carpet roll length for each part separately. The customers requirements are to produce a cutting solution of the carpet within 3 minutes, in order to be usable during the quotation process for estimating the amount of carpet required. Our system can find and prove the optimal solution for 106 of the 150 real-world instances provided by the customer, and find high quality solutions to the remainder within this time limit. In contrast the existing solution developed some years ago finds (but does not prove) optimal solutions for 30 instances. Our solutions reduce the wastage by more than 35% on average compared to the existing approach.

1 Introduction

The carpet cutting problem is a two-dimensional cutting and packing problem in which carpet shapes (also called items or objects) are cut from a rectangular carpet roll with a fixed roll width and a sufficiently long roll length. The goal is to find a non-overlapping placement of all carpet shapes on the carpet roll, so that the waste is minimised or in other words the utilisation of used carpet material is maximised while meeting all additional constraints. In our case the objective is to minimise the carpet roll length.

In this paper the carpet shapes are rectilinear polygons of up to 12 sides that can be made up of non-overlapping rectangles, that must be placed orthogonal on the carpet roll, i.e., their edges must be parallel to the borders of the roll. Before the placement of a carpet shape a rotation may be allowed by 90°, 180°, or 270°, i.e., it can be put onto the roll in one of four cardinal directions 0°, 90°, 180°, or 270°. But depending on the pile direction of the carpet there may be restrictions on the which cardinal directions can be used: perhaps only 0°, 90° or perhaps fixed to 0°.

Normally, a carpet shape is cut as a single piece from the carpet roll, but carpet shapes for covering stairs or filling up the remainder of a room are allowed to be cut in several pieces provided that the partition of these carpet shapes

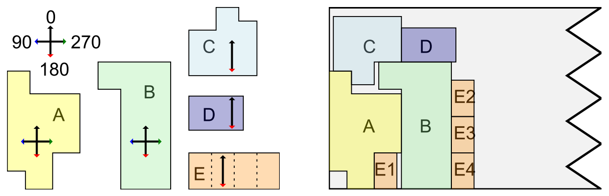


Fig. 1: Example of a carpet cutting instance.

satisfy additional constraints which are described later. The joint of carpets for stairs that is then introduced between two adjacent pieces can be hidden between the tread and the riser of the stairs once they are laid. The resulting seams of carpets filling up a room are hidden at the edge of a room. Moreover, these carpet shapes are simple rectangles.

Another complexity of the problem is that carpets have a *pile direction* that may constrain the orientations of some carpet shapes to be dependent on one or another. Clients may also prefer to have the pile direction fixed to ensure an even colour of the carpet when laid relative to a window. Where two carpets join, e.g. at a door way, the pile direction becomes visible if the two pieces are not laid with a similar pile direction. Therefore, all carpet shapes that are joined together must be arranged pile aligned in the plan. Carpet shapes for stairs must be pile aligned with the pile direction being up the stairs, for safety reasons this ensures that it is less easy to slip down the stairs.

Example 1. Figure 1 shows an example of a carpet cutting instance. On the left side the five carpet shapes A, B, C, D, and E are shown and on the right side their placement on the carpet roll (gray area). The roll is laid out from the left to the right, i.e., its width is the vertical edge and its length the horizontal one.

On the left-hand side each object contains arrows displaying in which direction the object can be placed where an arrow pointing to the top, left, bottom, or right stands for the direction 0°, 90°, 180°, and 270° respectively. As shown the object A, and B can placed in any direction, but not the objects C, D, and E which must be pile aligned. Moreover, the object E is a carpet for covering four stair steps. The vertical dotted line shows the edge between the tread and riser of two steps. The object E can be split at those edges.

In the placement shown on the right-hand side the objects A and B are placed in the 0° direction whereas the other objects are rotated by 180°. The object E is partitioned in four parts in order to minimise the needed roll length.

The carpet retailer uses a solution as a base of an on-site cost estimation and ordering process, to submit an offer to customers. The offer should be made in a timely manner and a three-minutes runtime limit is given to the cut-plan optimisation process and a nice graphic is displayed during the computation.

The carpet cutting problem can be characterised as an extension of a two-dimensional orthogonal strip packing (Osp) problem (referred to as a two-dimensional orthogonal open dimension problem by [21]) with additional constraints in which a packing of rectangles with minimal waste is sought. The extensions are the placement constraints between rectangles belonging to the same carpet shape and the partition constraints for carpet shapes covering stairs. The side effect of the first constraints is that for those carpet shapes a rotation by 180° and 270° may be not symmetric to a rotation by 0° and 90° respectively.

For Osp and related cutting and packing problems different methods have been applied, a survey can be found in [9]. The different methods can be roughly categorised in these groups: (1) positional placement/reasoning and (2) relational placement/reasoning. The first category includes methods such as the bottom-left rule [7,10] and the discretisation of the large rectangle [3]. The second category includes methods that determines the relations (above, under, left, and right) of each pair of rectangles [13] and the graph-theoretical models [5]. Our approach includes features of both categories.

A two-dimensional cutting and packing problem can be relaxed into two scheduled problems, once the problem is projected on the length-axis of the large rectangle and the other on the width-axis of the large rectangle. These relaxation can be used in order to infer more about possible positions of the items to be laid on the large rectangle and detect infeasibilities of partial solution earlier.

Constraint programming methods include the global constraints `cumulative` [1] that models a cumulative scheduling problem, the sweep pruning technique for k-dimensional objects [3] and the geost constraint [2] (modelling k-dimensional objects that can take different shapes). Moreover, special pruning algorithms exists for the `cumulative` constraint in the case of non-overlapping rectangles [4]. The sweep algorithm and the geost constraint are specifically designed to model non-overlapping object with at least 2-dimensions. These algorithms demonstrate very good results if the *slack* (the unused space) is small. If the slack is not small then the additional computational effort may not rewarded by the reduction of the search space.

The existing fielded solution [19,20] uses a combination of heuristic search and dynamic programming in a series of optimisation steps. The algorithm incrementally selects carpet shapes that are placed across the roll considering all alternatives and reduces the overall length of material in a branch-and-bound backtracking search. The algorithm is complex and can be subject to reduced performance when certain rare combinations of heuristic choice lead to inefficiencies of placement. It is not exact and often uses the full 3 minutes of runtime but considerably less for smaller problems. It was designed to run on 100MHz tablet PCs with considerably less computing power available than todays processors.

We define two new exact approaches to the carpet cutting problems. The first approach decomposes the problem into multiple instances where all the carpets have fixed dimension and orientation. These subproblems are solved sequentially maintaining the best solution found overall. Since all dimensions are fixed the constraint propagation is strong. But a disadvantage is there may be many instances for a single problem. The second approach models the orientation of the carpet as a variable and hence reduces the number of instances required for each problem. It can handle problems that the first approach cannot.

The subproblems are solved by the lazy clause generation (Lcg) [12] which is a hybrid of a Boolean satisfiability (Sat) solving and finite domain (Fd) solving. The Lcg lazily transforms an Fd problem into an Sat problem during the progress of a search where the conflict analysis only takes the Sat part into account. At the moment Lcg is one of the best exact solution approaches for tackling the basic resource-constrained project scheduling problem [15] and its extension with minimal and maximal time lags [16] in which an optimal schedule minimising the project duration is demanded. These problems involve an explaining version of the global constraint `cumulative` in their model which is also used to solve carpet cutting.

2 The Carpet Cutting Problem

In the carpet cutting problem there are three different types of carpet shapes: (***i***) *room carpets* that cover rooms which are made up of a number of rectangular pieces which are constrained to align; (***ii***) *stair carpets* that cover stairs which can be cut into regular pieces and are always rectangular; (***iii***) *edge filler carpets* that cover the remainder of a room that is only slightly wider than the width of the carpet. The remainder of the room is covered with multiple narrow pieces cut at any point providing each piece is of a minimum length.

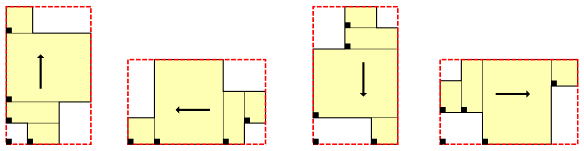
A room carpet is characterised by its set of (possible) orientations and offsets from its origin to the origin of its rectangles for each orientation. The origin of a room carpet is the bottom left corner of the smallest rectangle that encloses all its rectangles in each orientation. Each rectangle has a width and a length which are given for the 0° orientation. The carpet *origin* is the bottom left corner in each orientation. Where a room is larger in both directions than the width of the carpet, a choice of where the full roll width is aligned is made by the user in advance of the placement optimisation.

Example 2. Figure 2a shows the room carpet laid out in each orientation. Its smallest enclosing rectangle is displayed with a red-dotted line. The small black squares in each rectangle indicates the origin for the carpet and its rectangles. These pictures show how the offsets from the origin differ for each orientation.

Stair and edge filler carpets are characterised by their width and length. Each of them may be allowed to be cut in several pieces. Stair carpets are cut with regular breaks between the tread and the riser of two or more steps hence each single piece must cover an integral number of steps.

Edge filler pieces may be cut arbitrarily with irregular length breaks. These shapes can be divided at any position so long as their length is not smaller than a minimal given length. The resulting seam(s) is hidden at the edge of a room. Significant savings in material wastage occur for certain single room carpet orders using this approach. For both kinds of breaks a maximal number of pieces and minimal length of sub-pieces can be given.

Example 3. Figure 2b shows a stair carpet with 4 pieces and possible partitions, with a maximum of three pieces allowed. Figure 2c shows possible partitions for


(a)

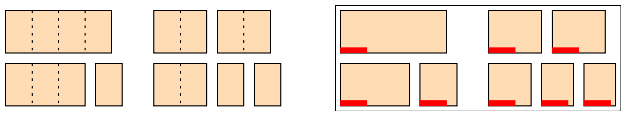

(b) (c)

Fig. 2: The origin of a room carpet and its rectangles in each orientation (a). Possible partitions for a stair carpet (b) and an edge filler carpet (c).

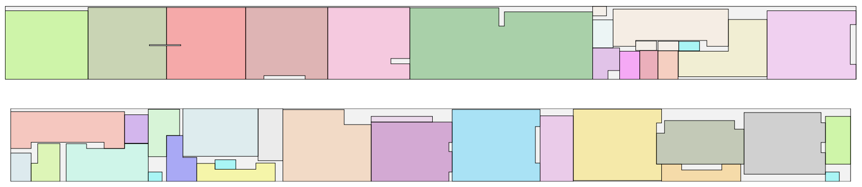


Fig. 3: A solution (split into two parts) for Cc instance with 34 room carpets (involving 74 rectangles) and 2 stair carpets (involving 7 rectangles). The roll length is about 93m to a granularity of 1cm.

an edge filler carpet with length 200 units, with a minimal length of 50 units (indicated by the bar in the bottom left corner) and a maximum of two cuts.

A formal specification of an instance I of the carpet cutting problem is defined as follows. We are given 3 sets of disjoint objects: (***i***) $Room$ is a set of room carpets. Each $c \in Room$ is defined by a set of rectangles $c.rect$. For each rectangle $r \in c.rect$ we have a length $r.len$ and width $r.wid$ (in the $0°$ orientation) together with an offset $(r.ox, r.oy)$ from the origin of the room carpet (in the $0°$ orientation). Moreover, each $c \in Room$ is also given a set of allowable orientations $c.ori \subseteq \{0°, 90°, 180°, 270°\}$. (***ii***) Str is a set of stair carpets. For each $c \in Str$ we have a width $c.wid$, step length $c.step$ and number of steps $c.n$ as well as a maximum number of pieces $c.pcs$ and minimum steps per piece length $c.min$. (***iii***) Edg is a set of edge filler carpets. For each $c \in Edg$ we have a width $c.wid$, length $c.len$ as well as a maximum number of pieces $c.pcs$ and minimum length per piece length $c.min$. The remaining part of the model is a set $Pile \subseteq Room$ which determines which carpets must be pile aligned, *i.e.* $c.ori = \{0°, 180°\}$ for each $c \in Pile$, and a roll width RW. Hence, $I = (Room, Str, Edg, Pile, RW)$. Note that all stair and edge filler carpets must be pile aligned, but this constraint can be neglected, since the pile orientations are symmetrical for rectangles as it is for parts of these carpets.

The aim is to find an allowable partitioning $c.part$ of each carpet $c \in Str \cup Edg$ into rectangles, and position (x, y) and allowed orientation for each rectangle r appearing in a room carpet such that: none of the rectangles overlap; each of the rectangles in a room carpet are correctly offset from the origin of the carpet; all pile aligned carpets are aligned in the same orientation, and the roll length RL is minimised.

Figure 3 shows the best solution found by our method for a large instance. It reduces the wastage by about 33% in comparison to the current method.

3 Static Model

The first model we present, the *static* model, splits the original problem into instances where the orientations and dimensions of each of the rectangular pieces are fixed in advance (*statically known*). This is achieved by fixing rotations of room carpets and fixing the partitions for stair carpets. The advantage of the static model is that it reduces the number of variables required to specify the problem, and gives stronger initial propagation. It reduces the requirements of the global constraints needed to model non-overlap, since dimensions are fixed. It also improves the strength of preprocessing. The obvious disadvantage of the static model is that the number of instances required to specify one original problem may become prohibitive.

To apply the static model we wish to fix the orientation and dimensions of all the rectangles in the problem. To do so we have to split the problem into multiple instances. For many problems in the customer data the number of instances required is not too large since they are often reasonably constrained.

3.1 Dealing with Orientations

Every carpet $c \in Room \setminus Pile$ has an allowable set of orientations in $\{0°, 90°, 180°, 270°\}$. We can split an instance I to remove possibilities of different orientations for a carpet c by creating the set of instances $I_o, o \in c.ori$ that are each identical to I except that $c.ori = \{o\}$, and for room carpets we swap the length $r.len$ and width $r.wid$ of the component rectangles if $o \in \{90°, 270°\}$, and update the offsets $(r.ox, r.oy)$ to reflect them from the new origin in this orientation.

If pile aligned carpets are involved in an instance then the instance is split in two instances. In one instance all pile aligned carpets c are fixed to the orientation 0° and in the other to 180°.

Note that before doing this we preprocess instances for reducing the possible orientations of carpets: (***i***) For room carpets consisting of one rectangle the orientations 0° and 180° (90° and 270°) are symmetric. If both orientations are given then one of them is removed. For square carpets the orientation is fixed to 0°. (***ii***) Some room carpets are too wide for the carpet roll if they are placed in a certain orientations. All those orientations are removed. (***iii***) Finally, if all room carpets in one instance that are made of more than one rectangle must be pile aligned then the pile-aligned constraint is removed from all of them and their orientation is fixed to $0°$, since each solution for the direction $0°$ is a solution for the direction $180°$ by rotating the carpet roll and all the placed objects by $180°$.

3.2 Stair carpets

Carpets for stairs play an important role for the difficulty of a problem because they can be partitioned in many combinations and introduce symmetries if two parts in the partition have the same length. We can ameliorate the difficulty of stair carpets by avoiding considering all possible partitions by determining "dominated" partitions.

Example 4. Suppose a stair carpet covers 15 steps and can be cut into an unlimited number of pieces where each part must consists of at least two steps. Possible partitions are $\{10, 5\}$, $\{10, 3, 2\}$, $\{5, 4, 3, 3\}$, etc. where each multiset represents a partition and the elements express the size in steps of each piece. The total number of possible partitions (incl. the partition $\{15\}$) is 41.

The partition problem is well studied in number theory. The (generating) function that counts the number of different partitions for a sum n is called the *partition function* [6]. This function grows exponentially as the value n increases. For stair carpets an important simplification of the problem arises when we realise that not all partitions need to be considered because some parts of a partition can be broken into smaller pieces which can be laid out in a way identical to the original coarser pieces.

Example 5. Consider a stair carpet with possible partitions $\{10, 5\}$ and $\{10, 3, 2\}$. Given a layout for the first partition, the piece of length 5 steps in the first partition can be split into two parts in which one part covers three steps and the other one two steps, thus giving a layout for the second partition. Hence we need not consider laying out the first partition, the partition $\{10, 5\}$ is *dominated* by the partition $\{10, 3, 2\}$.

Definition 1. Let P_1 and P_2 be two different partitions of n ($i.e \sum P_1 = \sum P_2 = n$). We say $P_1 = \{p_{11}, \ldots, p_{1m}\}$ is *dominated* by $P_2 = \{p_{21} \ldots, p_{2k}\}$ iff there is a mapping $\sigma : 1..m \to 1..k$ such that $\forall i \in 1..k : p_{2i} = \sum_{j \in 1..m \text{ where } \sigma(j)=i} p_{1j}$. That is we can further partition P_2 to obtain P_1. Given a set of partitions $\mathbf{P}$ we say $P \in \mathbf{P}$ is *dominating* if it is not dominated by any $P' \in \mathbf{P} - \{P\}$.

It follows that only dominating partitions must be considered during the solution process. We now construct a recursive definition of the number $nd(n, p, k)$ of dominating partitions for a stair carpet of length n steps with maximum number of pieces p and minimal step length k as follows:

$$nd(n, p, k) = nd(n, p, k, k)$$

$$nd(n, p, l, k) = \begin{cases} 0 & \text{if } 0 < n \wedge n < l \text{ or } 0 \wedge p > 0 \wedge l \geq 2k \\ 1 & \text{if } n = 0 \wedge p = 0 \text{ or } n = 0 \wedge p > 0 \wedge l < 2k \\ \sum\limits_{l \leq i \leq n} nd(n - i, p - 1, i, k) & \text{otherwise.} \end{cases}$$

Table 1: All dominating partitions for various lengths n where the minimal step length k is 2, and maximal pieces is n (so effectively no limit on pieces).

n	partitions	n	partitions	n	partitions	n	partitions
2	$\{2\}$	8	$\{3,3,2\}$, $\{2,2,2,2\}$	12	$\{3,3,3,3\}$,	14	$\{3,3,3,3,2\}$,
3	$\{3\}$	9	$\{3,3,3\}$, $\{3,2,2,2\}$		$\{3,3,2,2,2\}$,		$\{3,3,2,2,2,2\}$,
4	$\{2,2\}$	10	$\{3,3,2,2\}$,		$\{2,2,2,2,2,2\}$		$\{2,2,2,2,2,2,2\}$
5	$\{3,2\}$		$\{2,2,2,2,2\}$	13	$\{3,3,3,2,2\}$,	15	$\{3,3,3,3,3\}$,
6	$\{3,3\}$, $\{2,2,2\}$	11	$\{3,3,3,2\}$,		$\{3,2,2,2,2,2\}$		$\{3,3,3,2,2,2\}$,
7	$\{3,2,2\}$		$\{3,2,2,2,2\}$				$\{3,2,2,2,2,2,2\}$

The function $nd(n, p, l, k)$ returns the number of dominating partitions for a carpet of length n, maximal pieces p, minimum length l and minimum original length k. The definition captures the following reasoning. The first case is where there is carpet left but it is smaller that the minimal required length, or there is no carpet left but there are pieces remaining and one of the earlier pieces (which is at least size l) could be split in two. The second case is where there is no carpet and no pieces left, or there is no carpet left, and more pieces possible but the longest piece is not big enough to split. The recursive case adds up the possibilities of selecting a piece of size i in the range l to n from a carpet of size n, and determine how many ways to partition the remaining carpet. The remaining subproblem is for a carpet of length $n-i$, with one less piece possible, and a minimum length of i (so we pick pieces in increasing order). The function can be easily modified to return the dominating partitions.

In the customer data the parameter k is either 1 or 2 and the number of steps n in a stair carpet ranges from 1 to 18 and 2 to 15 for $k = 1$ and $k = 2$ respectively. For most of the customer data the number of cuts constraint is not constraining ($\geq n$ when $k = 1$ and $\geq \lfloor n/2 \rfloor$ when $k = 2$), and the total number of dominating partitions is small. This means we can separate the problem into different instances with different fixed (dominating) partitions. Table 1 shows the dominating partitions for stair carpets up to 15 steps for $k = 2$. If $k = 1$ then the partition with n parts "1", i.e., $\{1, \ldots, 1\}$ is the only dominating partition for stair carpets covering n steps.

We can split a carpet cutting instance I involving a stair carpet c as follows. For a stair carpet c we determine the set of dominating partitions $\mathbf{P}$ of c and create a new instance $I_P, P \in \mathbf{P}$ where $P = \{p_1, \ldots, p_m\}$ which is identical to I except that the partition function for carpet c is fixed so that $c.part = \{r_1, \ldots, r_m\}$ and the rectangular pieces r_i are constrained as follows: $r_i.wid = c.wid$, $r_i.len = p_i \times c.step$.

Too many dominating partitions For some cases in the customer data, for example $n = 18$, $k = 1$ and $p = 7$, there are 49 dominating partitions. Splitting into different instances becomes prohibitive when we have to consider other reasons for splitting such as multiple stair carpets, and different room carpet orientations.

When the number of dominating partitions is too large, we modify the partitioning as follows. We consider the partitioning problem with no limit on the number of pieces (or equivalently limit n). For the customer data, the maximal number of dominating partitions that arise with this weakening is 3 (as illustrated by Tab. 1). We split into instances using these dominating partitions. This model of course can create a carpet cutting with too many carpet pieces for a regular carpet c. For each rectangle $r \in c.part$ we add a Boolean variable $r.last$ to the model.

We constrain $r.last$ to holds if the rectangle does not have another rectangle $r' \in c.part$ directly to the right (1) and ensure that there are at most $c.pcs$ last parts (2). These constraints are posted for all carpets $c \in Str$:

$$\forall r \in c.part: \quad r.last \leftrightarrow (\forall r' \in c.part \setminus \{r\} : r.x + r.len \neq r'.x \vee r.y \neq r'.y) \quad (1)$$

$$\sum_{r \in c.part} r.last \leq c.pcs \ . \quad (2)$$

3.3 The Model

After handling rotations and stair carpets our original instance I is transformed into a set of *static instances* $\mathbf{I}$ in which all rectangles are fixed in orientation and length and width. If the splitting process created too many instances $\mathbf{I}$ or involved edge filler carpets then we will have to handle the original problem using the dynamic model defined in the next section.

We can now model each static instance $I' \in \mathbf{I}$ reasonably straightforwardly. Let a variable tuple $(r.x, r.y)$ be defined for each rectangle in the instance $Rect = (\bigcup_{c \in Str} c.part) \cup (\bigcup_{c \in Room} c.rect)$ which gives the position of the rectangle on the roll, and variable tuples $(c.x, c.y)$ for each room carpet $c \in Room$. We introduce variable RL to hold the roll length. The constraints of the model are (1–2) if required, together with:
Each rectangle must be on the roll

$$\forall r \in Rect : 0 \leq r.x \wedge r.x + r.len \leq RL \wedge 0 \leq r.y \wedge r.y + r.wid \leq RW \ . \quad (3)$$

Each rectangle in a room carpet must be placed correctly relative to the carpet

$$\forall c \in Room, \forall r \in c.rect : r.x = c.x + r.ox \wedge r.y = c.y + r.oy \ . \quad (4)$$

No rectangles overlap.

$$\texttt{diff2}([r.x \mid r \in Rect], [r.y \mid r \in Rect], [r.len \mid r \in Rect], [r.wid \mid r \in Rect]) \quad (5)$$

For the solver we make use of there is no global definition of `diff2`, instead it is decomposed into a disjunction of possibilities.

$$\begin{aligned}\forall r_1, r_2 \in Rect \text{ s.t. } r_1 < r_2 : r_1.x + r_1.len \leq r_2.x \vee r_2.x + r_2.len \leq r_1.x \\ \vee\, r_1.y + r_1.wid \leq r_2.y \vee r_2.y + r_2.wid \leq r_1.y \ . \quad (6)\end{aligned}$$

This decomposition is very weak, and only propagates if three inequalities are unsatisfiable and the remaining one undecided. In order to get a stronger propagation on the involved variables two global `cumulative` constraints are used, i.e., one for the roll length and the other one for the roll width. We hence enhance the model with the redundant constraints

$$\texttt{cumulative}(([r.x \mid r \in \mathit{Rect}], [r.\mathit{len} \mid r \in \mathit{Rect}], [r.\mathit{wid} \mid r \in \mathit{Rect}], \mathit{RW}) \ , \tag{7}$$

$$\texttt{cumulative}(([r.y \mid r \in \mathit{Rect}], [r.\mathit{wid} \mid r \in \mathit{Rect}], [r.\mathit{len} \mid r \in \mathit{Rect}], \mathit{RL}) \ . \tag{8}$$

The `cumulative` constraints are implemented as global constraints with explanation [15]. They provide much stronger propagation than the decomposed `diff2`. Equation (8) also provides strong lower bound reasoning on the objective RL.

In order to find the optimal solution to an original problem instance I using the static model we must find the minimal roll length solution for any of the instances $\mathbf{I}$ it was split into.

4 Dynamic Model

The static model splits the problem into multiple instances to fix the dimensions of the rectangles. But this can be prohibitive when an original problem splits into very many instances, and it does not give an approach to edge filler carpets. The dynamic model models the problem more directly.

Orientation For each room carpet c we model its orientation with variable $c.\mathit{vori}$ which takes a value in $c.\mathit{ori}$. We introduce two Boolean variables $c.0or180$ which is true if the carpet is oriented at 0° or 180°, and similarly $c.0or90$.

For each rectangle r we introduce a variable $r.\mathit{vlen}$ to hold its length (after orientation), and similarly a variable to hold its width $r.\mathit{vwid}$, and x offset $r.\mathit{vox}$ and y offset $r.\mathit{voy}$ from the carpet origin. For each carpet c and rectangle $r \in c.\mathit{rect}$ we precalculate two arrays of offsets of r from the carpet origin and each orientation $o \in \{0°, 90°, 180°, 270°\}$ given by $ox_{c,r}[o]$, and $oy_{c,r}[o]$.

The model includes the following constraints for each carpet $c \in \mathit{Room}$: Enforcing agreement of the orientation and Boolean variables

$$c.0or180 = c.\mathit{vori} \in \{0°, 180°\} \ \wedge \ c.0or90 = c.\mathit{vori} \in \{0°, 90°\} \ . \tag{9}$$

Setting length, width and offsets of each rectangle depending on orientation

$$\begin{aligned} \forall r \in c.\mathit{rect} : \quad & r.\mathit{vox} = ox_{c,r}[c.\mathit{vori}] \ \wedge \ r.\mathit{voy} = oy_{c,r}[c.\mathit{vori}] \\ \wedge \quad & r.\mathit{vwid} = r.\mathit{len} + (r.\mathit{wid} - r.\mathit{len}) \times c.0or180 \qquad (10) \\ \wedge \quad & r.\mathit{vlen} = r.\mathit{wid} + (r.\mathit{len} - r.\mathit{wid}) \times c.0or180 \ . \qquad (11) \end{aligned}$$

Note that the offset calculation constraints are examples of `element` constraints.

Edge filler carpets Given an edge filler carpet $c \in \mathit{Edg}$ we model this with a set of $c.\mathit{pcs}$ different rectangles $c.\mathit{part}$ (so $|c.\mathit{part}| = c.\mathit{pcs}$). We have to ensure that these pieces either 0 length (and hence only really pseudo pieces) or reach the minimal length.

$$\forall c \in \mathit{Edg}, \forall r \in c.\mathit{part} : r.\mathit{vwid} = c.\mathit{wid} \wedge (r.\mathit{vlen} = 0 \vee r.\mathit{vlen} \geq c.\mathit{min}) \tag{12}$$

And the sum of the lengths must equal the irregular break length

$$\forall c \in \mathit{Edg} : \sum_{r \in c.\mathit{part}} r.\mathit{vlen} = c.\mathit{len} \ . \tag{13}$$

We can also reason about dominating partitions for irregular breaks. Any partition with a piece r where $r.vlen \geq 2c.min$ and one piece of zero length will be dominated by a partition where r is broken in two. Hence we can add

$$\forall c \in Edg: \; (\exists r \in c.part.r.vlen = 0) \rightarrow (\forall r \in c.part. \; r.vlen < 2c.min) \; . \qquad (14)$$

If $c.len \geq 2(c.pcs - 1) \times c.min$ then there can be no zero length pieces since the right hand side of the implication in (14) cannot be satisfied at the same time as (13), hence in this case we can simplify (12).

The Model The set of rectangles is $Rect = \bigcup_{c \in Room} r.rect \cup \bigcup_{c \in Str \cup Edg} c.part$. We assume that for each stair piece $r.vlen = r.len$ and $r.vwid = r.wid$. The constraints of the model are: (1–2) if required, (3–8) with $r.len$ replaced by $r.vlen$ and $r.wid$ replaced with $r.vwid$, (9–11) and (12–14) if required.

5 Refining the Models

The basic model can be further enhanced in order to improve the propagation, reduce the model size, and strengthen the reasoning and the conflict-driven search in the LCG solver.

Variable views Variable views [14] are a form of variable aliasing. Suppose $y = ax + c$ where a and c are constants, then rather than creating a new variable for y use a view to compute information about the (view) variable y from the real variable x. This refinement (views) is particularly useful for LCG solvers since it improves learning. For a fixed orientation room carpet c we can replace the variables $r.x$ and $r.y$ by views on $c.x$ and $c.y$ for all $r \in c.rect$ using (4). For non-fixed orientation carpets c we can use views to define $r.vlen$ and $r.vwid$ for $r \in c.rect$ using (10) and (11).

Disjunction and Better `diff2` decomposition In all carpet cutting problems the roll width is narrow in comparison to some carpets, so that no other carpet can be positioned below or above to those carpets. We say these carpets are *in disjunction*. Carpets that are in disjunction with all others can be placed at the beginning of the roll. We denote this as the disj refinement.

We can use disjunction to improve the `diff2` decomposition (diff2). Assume function $not\_par(r_1, r_2)$ holds if r_1 and r_2 cannot overlap horizontally on the role. For pairs r_1, r_2 with this holds we replace the body of (6) by $r_1.x + r_1.len \leq r_2.x \;\vee\; r_2.x + r_2.len \leq r_1.x$. The simplest definition of not_par just $r_1.wid + r_2.wid > RW$, but it can be improved by considering the compulsory parts [8] and possible y coordinates of r_1 and r_2 to determine if there is insufficient space for them to overlap.

Symmetry breaking constraints In the model symmetries can occur between rectangles that have the same size, i.e., length and width. The most common case for symmetries occurs for pieces of stair carpets. We assume a function $same(r, r')$ which (statically) tests if two rectangles have the same dimensions, are not rotatable and are not part of a room carpet with more than one rectangle. For refinement sym we add a lexicographic ordering on $(r.y, r.x)$ for rectangles that are the same. Symmetry breaking can also considerably simplify the definition of $r.last$ for stair carpets $c \in Str$ since we only need to consider the lexicographically least member of each symmetric group that appears in the partition $c.part$. Finally we can enforce that the pieces of an edge filler carpet are ordered in length.

Forbidden gaps *Forbidden gaps* [17] are areas between a rectangle and a long edge (either from another rectangle or a boundary) that are too small to accommodate any part of other rectangles. In this paper, we forbid these gaps between rectangles that have fixed orientation and do not belong to room carpets with multiple rectangles, and the borders of the carpet roll as follows.

Let *gap* be the minimal width of any rectangle. In the y direction (fbg y) We consider how many rectangles (multiples of *gap*) might fit between the considered object edge and the border of the carpet roll: (***i***) none, (***ii***) one, and (***iii***) two or more. In case (***i***) the y coordinate is set to 0. In case (***ii***) the object is aligned with either the top or the bottom. In case (***iii***) constraints are added to forbid placements of the object that creates a smaller gap than *gap* with either the top or bottom of the roll. Similarly, we impose forbidden gaps (fbg x) for the left and right border of the roll.

6 Search

To solve a carpet cutting problem instance I in our approaches we need to solve a series of instances $\mathbf{I}$ determined by splitting. The generic algorithm first attempts to find a good solution for each $I' \in \mathbf{I}$ and then uses the best solution found as an upper bound on roll length, and searches for an optimal solution of each $I' \in \mathbf{I}$ in the order of how good a first solution we found for them. During this process the upper bound is always the best solution found so far.

The two phase approach has two benefits. First it means that domain sizes of variables in the optimisation search are much smaller. Because lazy clause generation generates a Boolean representation of the size of the initial domain size this makes the optimisation search much more efficient. Second the first phase ranks that split instances on likelihood of finding good solutions, so usually later instances in the optimisation phase are quickly found to be unable to lead to a better solution.

First solution generation The goal of the first search is to quickly generate a first solution that gives a good upper bound on the carpet roll length. We examine each split instance in $\mathbf{I}$ in turn. We order the split instances by the partitions of regular stair carpets examining partitions with fewer pieces before partitions with more pieces, and otherwise breaking ties arbitrarily.

We use a simple sequential search on each split instance. We treat the room carpets first, in decreasing order of total area. First we assign a horizontal or vertical orientation for all room carpets by fixing the $c.0or180$, which fixes the dimensions of each rectangle. Then we fix the orientation by fixing $c.0or90$. We then fix the lengths of edge filler carpets. We next determine $c.x$ for all room carpets c, and then determine each $c.y$ again in decreasing area order. Finally we place each stair carpet rectangles by fixing $r.x$ and then $r.y$ treating each rectangle in input order.

Minimisation A hybrid sequential/activity based search is used to find optimal solutions. We first fix the orientations of each room carpet as we did in the first-solution search. Then we switch to the activity-based search (a variant of VSIDS [11]) which concentrates on variables which are involved in lots of recent failures. Activity-based search is tightly tied to the learning solver we use, but is acknowledged from the SAT community to be very effective.

For the activity-based search, we use a geometric restart policy on the number of node failures in order to make the search more robust. The restart base and factor are 128 failures and 2.0, respectively.

7 Experiments

The experiments were carried out on a 64-bit machine with Intel(R) Pentium(R) D processing with 3.4 GHz clock and Ubuntu 9.04. For each original problem instance I an overall 3 minutes runtime limit was imposed for calculating carpets that are in disjunction with all other carpets if the refinement disj is used, finding a first solution and minimising the roll length for all split instances $\mathbf{I}$.

The G12/FDX solver from the G12 Constraint Programming Platform [18] was used as the LCG solver. We also experimented with the G12 FD solver using search more suitable for FD (placement of the biggest carpets at first). It could only optimally solve 7 instances compared to 76 for LCG using the same search. This shows that LCG is vital for solving the problem to prune substantial parts of the search space.

Dynamic versus static model Table 2 compares the static and dynamic model as well as the current solution approach on the instances which the static model can handle (126 of 150). It shows the number of instances solved optimally ("opt."), the sum of the best first solutions found for each instance ("init. ΣRL"), the sum of the best solutions found for each instance ("ΣRL") and the area of wastage ("wast."), i.e. for one instance $RL \times RW - \sum_{c \in Rect} c.len \times c.wid$, relatively to the wastage created by the current method as well as the total runtime to solve all instances ("Σrt."). The static approach solves one more problem and its first solutions are better than for the dynamic approach. In total, a better first solution was generated for 55 instances. Where applicable the static approach is preferable.

The existing method outperforms manual approaches using a manual grapical editing tool. It finds, but does not prove, 27 optimal solutions. It was tested by IF Computer GmbH on a Dell Latitude D820 with a Intel(R) Core(TM) Duo processor T2400 processing with 1.86 Ghz clock. The times marked (†) for the existing approach are the sum of times when the best solution was found. Since it cannot prove optimality for the majority of instances the existing method uses

Table 2: Comparison between dynamic and static approach.

approach	opt.	init. ΣRL	ΣRL	wast.	Σrt.
dynamic	92/126	171,645	160,536	66.5%	6,247s
static	93/126	168,270	160,399	65.9%	6,946s
Current method	27/126	-	167,668	100%	7,450s†

Table 3: Results of different refinements

disj	views	diff2	sym	fbg x	fbg y	opt.	init. ΣRL	ΣRL	wast.	Σrt.
						86/150	232,181	221,542	67.9%	12,721s
×						88/150	232,075	221,521	67.8%	12,360s
	×					89/150	232,181	221,248	66.9%	11,999s
		×				89/150	232,181	221,240	66.9%	11,980s
			×			99/150	232,181	221,344	67.2%	9,933s
				×		88/150	232,181	221,596	68.1%	12,295s
					×	88/150	232,181	221,399	67.4%	12,302s
				×	×	89/150	232,181	221,060	66.3%	12,385s
×	×	×	×	×	×	**106/150**	232,075	**220,775**	**65.2%**	**9,290s**
Current method						30/150	-	230,795	100%	8,988s†

the whole 3 minutes. The new approach results in an improvement of wastage of over 33%.

Refinements Table 3 presents the impact of different refinements on the dynamic models. The entry × means that the refinement was used. We compare the different refinements with the same features as before.

The change in number of optimally solved instances clearly illustrates the important of symmetry breaking for proving optimality. Variable views and forbidden gaps have a minor impact on proving optimality.

We can see a tradeoff in the refinements. Most make it harder to find solutions, but reduce the search space required to prove optimality. When applying all refinements we solve the most instances, and generate solutions with minimal total length, since the new optimal solutions make up for unsolved problems where we found worse solutions.

8 Conclusion

We have created an approach to carpet cutting that can find and prove the optimal solution for typical problems instances within 3 minutes. The power of the approach comes from the combination of careful modelling of the stair breaking constraints to eliminate symmetries and dominated solutions, and the use of lazy clause generation to drastically reduce the time to proof of optimality.

Acknowledgements We are indebted to IF Computer GmbH for providing us not only real world data, but also sharing their knowledge and results with us.

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council.

References

1. Aggoun, A., Beldiceanu, N.: Extending CHIP in order to solve complex scheduling and placement problems. Math. Comput. Model. 17(7), 57–73 (1993)
2. Beldiceanu, N., Carlsson, M., Poder, E., Sadek, R., Truchet, C.: A generic geometrical constraint kernel in space and time for handling polymorphic k-dimensional objects. In: CP 2007. LNCS, vol. 4741, pp. 180–194 (2007)
3. Beldiceanu, N., Carlsson, M.: Sweep as a generic pruning technique applied to the non-overlapping rectangles constraint. In: CP 2001. pp. 377–391 (2001)
4. Beldiceanu, N., Carlsson, M., Poder, E.: New filtering for the cumulative constraint in the context of non-overlapping rectangles. In: CPAIOR 2008. pp. 21–35 (2008)
5. Fekete, S.P., Schepers, J., van der Veen, J.C.: An exact algorithm for higher-dimensional orthogonal packing. Oper. Res. 55(3), 569–587 (2007)
6. George, A.E.: The Theory of Partitions. Cambridge University Press (1998)
7. Hadjiconstantinou, E., Christofides, N.: An exact algorithm for general, orthogonal, two-dimensional knapsack problems. Eur. J. Oper. Res. 83(1), 39–56 (1995)
8. Lahrichi, A.: Scheduling: the notions of hump, compulsory parts and their use in cumulative problems. C. R. Acad. Sci., Paris, Sér. I, Math. 294(2), 209–211 (1982)
9. Lodi, A., Martello, S., Monaci, M.: Two-dimensional packing problems: A survey. Eur. J. Oper. Res. 141(2), 241–252 (2002)
10. Martello, S., Vigo, D.: Exact solution of the two-dimensional finite bin packing problem. Manage. Sci. 44(3), 388–399 (1998)
11. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: DAC 2001. pp. 530–535 (2001)
12. Ohrimenko, O., Stuckey, P.J., Codish, M.: Propagation via lazy clause generation. Constraints 14(3), 357–391 (2009)
13. Pisinger, D., Sigurd, M.: Using decomposition techniques and constraint programming for solving the two-dimensional bin-packing problem. INFORMS J. Comput. 19(1), 36–51 (2007)
14. Schulte, C., Tack, G.: Views iterators for generic constraint implementations. In: CP 2005. pp. 817–821 (2005)
15. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Explaining the cumulative propagator (2010), to appear in Constraints
16. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Solving the resource constrained project scheduling problem with generalized precedences by lazy clause generation (Sep 2010), http://arxiv.org/abs/1009.0347
17. Simonis, H., O'Sullivan, B.: Search strategies for rectangle packing. In: CP 2008. pp. 52–66 (2008)
18. Stuckey, P.J., Garcia de la Banda, M., Maher, M., Marriott, K., Slaney, J., Somogyi, Z., Wallace, M.G., Walsh, T.: The G12 project: Mapping solver independent models to efficient solutions. In: ICLP 2005. pp. 9–13 (2005)
19. Verden, A., Pearson, C., Birtwistle, M.: Reducing material wastage in the carpet industry. In: PAP'98. pp. 101–112 (1998)
20. Verden, A., Pearson, C., Birtwistle, M.: Reducing material wastage in the carpet industry. In: INAP'98. pp. 76–91 (1998)
21. Wäscher, G., Haußner, H., Schumann, H.: An improved typology of cutting and packing problems. Eur. J. Oper. Res. 183, 1109–1130 (2007)