Problems, Models and Algorithms in One- and Two-Dimensional Cutting

DISSERTATION

zur Erlangung des akademischen Grades

Doctor rerum naturalium
(Dr. rer. nat.)

vorgelegt

der Fakultät Mathematik und Naturwissenschaften
der Technischen Universität Dresden

von

Diplommathematiker (Wirtschaftsmathematik) Belov, Gleb

geboren am 6. Januar 1977 in St. Petersburg/Russland

Gutachter:

Prof. Dr. Andreas Fischer	Technische Universität Dresden
Prof. Dr. Robert Weismantel	Otto-von-Guericke Universität Magdeburg
Prof. Dr. Stefan Dempe	Bergakademie Freiberg

Eingereicht am: 08.09.2003

Tag der Disputation: 19.02.2004

Preface

Within such disciplines as Management Science, Information and Computer Science, Engineering, Mathematics and Operations Research, problems of cutting and packing (**C&P**) of concrete and abstract objects appear under various specifications (cutting problems, knapsack problems, container and vehicle loading problems, pallet loading, bin packing, assembly line balancing, capital budgeting, changing coins, etc.), although they all have essentially the same logical structure. In cutting problems, a large object must be divided into smaller pieces; in packing problems, small items must be combined to large objects. Most of these problems are NP-hard.

Since the pioneer work of L.V. Kantorovich in 1939, which first appeared in the West in 1960, there has been a steadily growing number of contributions in this research area. In 1961, P. Gilmore and R. Gomory presented a linear programming relaxation of the *one-dimensional cutting stock problem*. The best-performing algorithms today are based on their relaxation. It was, however, more than three decades before the first 'optimum' algorithms appeared in the literature and they even proved to perform better than heuristics. They were of two main kinds: enumerative algorithms working by separation of the feasible set and cutting plane algorithms which cut off infeasible solutions. For many other combinatorial problems, these two approaches have been successfully combined. In this thesis we do it for one-dimensional stock cutting and *two-dimensional two-stage constrained cutting*. For the two-dimensional problem, the combined scheme provides mostly better solutions than other methods, especially on large-scale instances, in little time. For the one-dimensional problem, the integration of cuts into the enumerative scheme improves the results of the latter only in exceptional cases.

While the main optimization goal is to minimize material input or trim loss (waste), in a real-life cutting process there are some further criteria, e.g., the number of different cutting patterns (setups) and open stacks. Some new methods and models are proposed. Then, an approach combining both objectives will be presented, to our knowledge, for the first time. We believe this approach will be highly relevant for industry.

The presented investigation is a result of a three-year study at Dresden Uni-

versity, concluded in August 2003.

I would like to express my gratitude to all the people who contributed to the completion of my dissertation. Dr. Guntram Scheithauer has been a very attentive and resourceful supervisor, not only in the research. He and Professor Johannes Terno, who died in 2000 after a long illness, supervised my diploma on a cutting plane algorithm for one-dimensional cutting with multiple stock lengths.

Professor Elita Mukhacheva (Russia) introduced me to the field of cutting and packing in 1995. In her research group I studied the powerful sequential value correction heuristic for one-dimensional stock cutting.

The last two years of the study were supported by the GOTTLIEB DAIMLER- and CARL BENZ-Foundation. The conferences on different topics and student meetings organized by the foundation were very useful for interdisciplinary communication; they provided opportunities to learn, for example, how to present one's own research in an understandable way.

My frequent visits to Ufa, my home city in the southern Urals, were warmly welcomed and nutritionally supported by my family. Many thanks to my dance partner Kati Haenchen for two years of intensive ballroom dancing and for understanding when I had to give up the serious sport because of time restrictions.

My thanks go to Joachim Kupke, Marc Peeters, and (quite recently) Cláudio Alves with J.M. Valério de Carvalho for testing their branch-and-price implementations on the hard28 set; to Jon Schoenfield for providing this set, for many discussions, and for thorough proofreading of large parts of the thesis; to Vadim Kartak for some brilliant ideas about cutting planes; to Eugene Zak and Helmut Schreck, TietoEnator MAS GmbH, for their informative advice about real-life cutting; to François Vanderbeck for providing his test instances; and to the anonymous referees of Chapter 2 (which was submitted to EJOR) for the remarks on presentation.

Dresden, September 2003
Gleb Belov

Contents

Introduction

Optimization means to maximize (or minimize) a function of many variables subject to constraints. The distinguishing feature of *discrete, combinatorial,* or *integer* optimization is that some of the variables are required to belong to a discrete set, typically a subset of integers. These discrete restrictions allow the mathematical representation of phenomena or alternatives where indivisibility is required or where there is not a continuum of alternatives. Discrete optimization problems abound in everyday life. An important and widespread area of applications concerns the management and efficient use of scarce resources to increase productivity.

Cutting and packing problems are of wide interest for practical applications and research (cf. [DST97]). They are a family of natural combinatorial problems, encountered in numerous areas such as computer science, industrial engineering, logistics, manufacturing, etc. Since the definition of C&P problems as geometric-combinatoric problems is oriented on a logical structure rather than on actual phenomena, it is possible to attribute apparently heterogeneous problems to common backgrounds and to recognize general similarities. Figure 1, which is taken from [DF92], gives a structured overview of the dimensions of C&P.

A *typology* of C&P problems by H. Dyckhoff [Dyc90, DF92] distinguishes between the dimension of objects (1,2,3,N), the kind of assignment, and the structure of the set of large objects ('material') and of small objects ('products', 'pieces', 'items'). In 2- and 3-dimensional problems we distinguish between rectangular and *irregular* C&P (objects of complex geometric forms). Rectangular cutting may be *guillotine*, i.e., the current object is always cut end-to-end in parallel to an edge.

Inspired by the explosion of the research in the last 30 years, which in the last 10–15 years brought about the ability to solve really large problems, in this thesis we investigate two problems which can be effectively formulated by *Integer Linear Programming* (**ILP**) models. Effectiveness is understood as the ability to construct successful solution approaches on the basis of a model. For the problems considered, there are two main kinds of linear models: *assignment* formulations (decision variables determine which items are cut from each large object) and

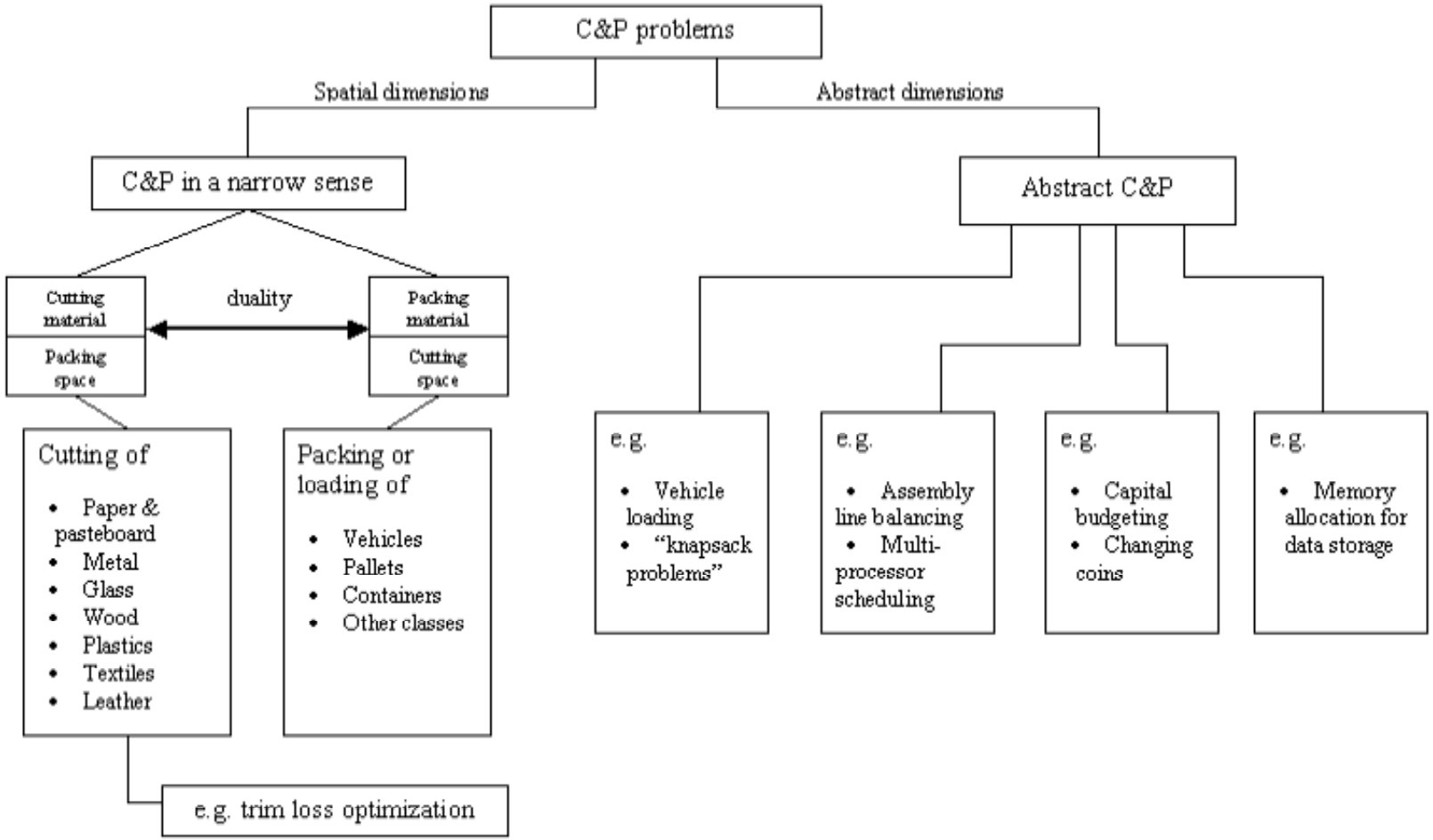


Figure 1: Phenomena of cutting and packing

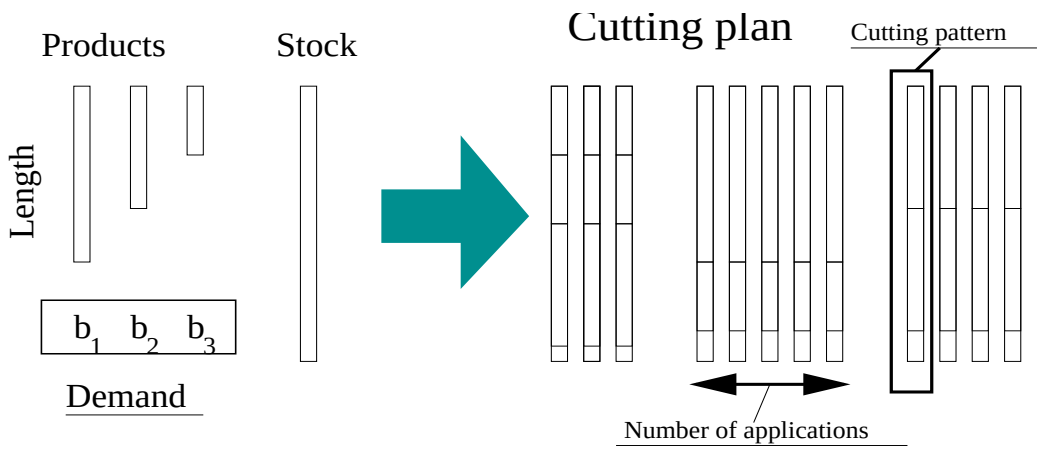


Figure 2: One-dimensional cutting stock problem with one stock type

strip/cutting pattern generation (decision variables determine which patterns are used, i.e., solutions are combined from ready patterns).

The *one-dimensional cutting stock problem* (**1D-CSP**) is to obtain a given set of order lengths from stock rods of a fixed length (Figure 2). The objective is typically to minimize the number of rods (*material input*). The *two-dimensional two-stage constrained cutting problem* (**2D-2CP**) is to obtain a subset of rectangular items from a single rectangular plate so that the total value of the selected items is maximized (Figure 3). The technology of cutting is *two-stage guillotine*, i.e., in the first stage we obtain vertical strips by guillotine edge-to-edge cuts; in the second stage the cutting direction is rotated by 90° and the cuts across the strips produce the items. Moreover, the difficulty of the *constrained* case is that the number of items of a certain type is limited; otherwise the problem is rather easy. Both 1D-CSP and 2D-2CP are classical topics of research and also of high relevance for production.

Both for cutting patterns in 1D-CSP and for strip patterns in 2D-2CP, the number of patterns for a given problem can be astronomical. Thus, in a pattern-oriented approach we employ *delayed pattern generation*.

In Chapter 1 we discuss some known models of the problems under investigation and propose a new *subpattern model* for 1D-CSP which does away with the astronomical number of complete patterns and combines them from a smaller number of subpatterns. The model has a weak continuous (LP) relaxation. Then some problems related to 1D-CSP and 2D-2CP are stated. The rest of the chapter introduces traditional solution approaches, e.g., heuristics including delayed pattern generation based on the LP relaxation and exact enumerative schemes including bounds on the objective value.

In Chapter 2 we investigate two approaches based on pattern generation, an

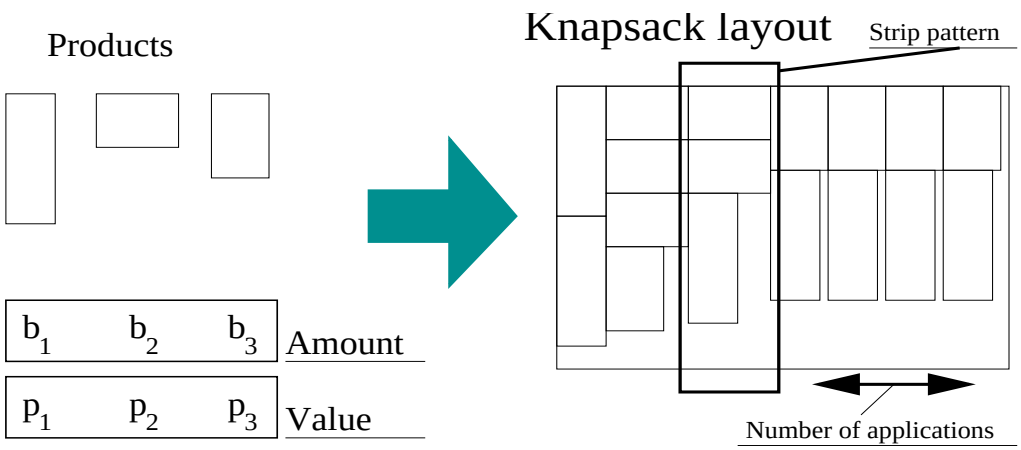


Figure 3: Two-dimensional two-stage constrained knapsack

enumerative scheme called *branch-and-price* and general-purpose cutting planes, and combine them. For branch-and-price, some widely applied tools like pseudo-costs, reduced cost bounding and different branching strategies are tested. For cutting planes, numerical stability is improved and mixed-integer cuts are integrated. The combined approach produces mostly better results for 2D-2CP than other known methods, especially on large instances. For 1D-CSP, general-purpose cuts are necessary only in exceptional instances.

In Chapter 3 we discuss the real-life conditions of 1D-CSP. Among many industrial constraints and criteria, the *number of different patterns* and the *number of open stacks* seem to add significant complexity to the optimization problem. The latter are the number of *started and not yet finished product types* at some moment during the sequence of cutting. An approach minimizing these auxiliary criteria should not relax the initial objective, i.e., the minimization of material input. A simple non-linear model is proposed for pattern minimization whose linear relaxation enables the application of an enumerative scheme. The sequential value correction heuristic used in Chapter 2 to solve *residual problems* is improved and modified to restrict the number of open stacks to any given limit; tests show only a negligible increase of material input. Then there follows a strategy to combine all three objectives, which is probably done for the first time.

In Chapter 4 we consider assignment formulations of 2D-2CP. New models with *variable strip widths* will be presented. Symmetries in the search space are eliminated by *lexicographic constraints* which are already known from the literature. However, previously known models with fixed strip widths are shown to be more effective.

Chapter 1

State-of-the-Art Models and Algorithms

1.1 One-Dimensional Stock Cutting and Bin-Packing

The one-dimensional cutting stock problem (1D-CSP, Figure 2) is defined by the following data: (m, L, $l = (l_1, \ldots, l_m)$, $b = (b_1, \ldots, b_m)$), where L denotes the length of each stock piece, m denotes the number of smaller piece types and for each type $i = 1, \ldots, m$, l_i is the piece length, and b_i is the order demand. In a *cutting plan* we must obtain the required set of pieces from the available stock lengths. The objective is to minimize the number of used stock lengths or, equivalently, trim loss (waste). In a real-life cutting process there are some further criteria, e.g., the number of different cutting patterns (setups) and open stacks (Chapter 3).

A special case in which the set of small objects is such that only one item of each product type is ordered, i.e., $b_i = 1\ \forall i$ (sometimes also when b_i are very small), is known as the *bin-packing problem* (**1D-BPP**). This special case, having a smaller overall number of items, is more suitable for pure combinatorial solution approaches.

1.1.1 The Model of Kantorovich

The following model is described in [Kan60] and in the survey [dC02]; it is called *assignment* formulation in [Pee02]. Let N be an upper bound on the number of stock lengths needed in an optimum solution and x_{ij} ($i = 1, \ldots, m$; $j = 1, \ldots, N$) be the number of items of type i in the j-th stock length. Let $y_j = 1$ if stock length j is used in the solution, otherwise $y_j = 0$.

$$z_{\text{Kant}} = \min \sum_{j=1}^{N} y_j \tag{1.1}$$

$$\text{s.t.} \quad \sum_{i=1}^{m} l_i x_{ij} \le L y_j, \qquad j = 1, \dots, N \tag{1.2}$$

$$\sum_{j=1}^{N} x_{ij} \ge b_i, \qquad i = 1, \dots, m \tag{1.3}$$

$$x_{ij} \in \mathbb{Z}_+, \quad y_j \in \mathbb{B}, \qquad \forall i, j. \tag{1.4}$$

The *continuous relaxation* of this model (when all variables are allowed to take real values) is very weak. Consider the following instance with large waste: $m = 1$ and $l_1 = L/2 + \epsilon$, $\epsilon \to +0$. The objective value of the relaxation is $\underline{z}_{\text{Kant}} = z_{\text{Kant}}(1/2 + \epsilon/L)$. Actually, $\underline{z}_{\text{Kant}}$ is equal to the *material bound* $\sum l_i b_i / L$. As stronger relaxations exist, it is not advantageous to use this model as a bound in an enumerative framework. Furthermore, the model has much symmetry: by exchanging values corresponding to different stock lengths, we obtain equivalent solutions which generally leads to an increased effort in an enumerative approach.

1.1.2 The Column Generation Model of Gilmore and Gomory

A reason for the weakness of the relaxation of model (1.1)–(1.4) is that the number of items in a stock length and the pattern frequencies y_j can be non-integer. DANTZIG-WOLFE *decomposition* [NW88, dC02], applied to the knapsack constraint (1.2), restricts each vector $(x_{1j}, \dots, x_{mj})$, $j = 1, \dots, N$, to lie inside the *knapsack polytope*, which is the set of linear combinations of all feasible cutting patterns.

A *cutting pattern* (Figure 2) describes how many items of each type are cut from a stock length. Let column vectors $a^j = (a_{1j}, \dots, a_{mj}) \in \mathbb{Z}_+^m$, $j = 1, \dots, n$, represent all possible cutting patterns. To be a valid cutting pattern, a^j must satisfy

$$\sum_{i=1}^{m} l_i a_{ij} \le L \tag{1.5}$$

(knapsack condition). Moreover, we consider only *proper* patterns:

$$a_{ij} \le b_i, \quad i = 1, \dots, m, \quad j = 1, \dots, n \tag{1.6}$$

because this reduces the search space of the continuous relaxation in instances where the demands b are small [NST99].

Let x_j, $j = 1, \dots, n$, be the frequencies (*intensities*, *multiplicities*, *activities*, i.e., the numbers of application) of the patterns in the solution. The model of Gilmore and Gomory [GG61] is as follows:

$$z_{\text{G\&G}}^{\text{1D-CSP}} = \min \sum_{j=1}^{n} x_j \tag{1.7}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} a_{ij} x_j \ge b_i, \quad i = 1, \dots, m \tag{1.8}$$

$$x_j \in \mathbb{Z}_+, \quad j = 1, \dots, n. \tag{1.9}$$

The huge number of variables/columns is not available explicitly for practical problems. Usually, necessary patterns are generated during a solution process, hence the term *column generation*. However, the number of different patterns in a solution cannot be greater than the number of stock lengths and is usually comparable with the number of piece types.

The model has a very strong relaxation. There exists the conjecture *Modified Integer Round-Up Property* (**MIRUP**, [ST95b]):

> *The gap between the optimum value $z_{G\&G}^{1D\text{-}CSP}$ and the optimum relaxation value $\underline{z}_{G\&G}^{1D\text{-}CSP}$ (obtained by allowing non-integer variable values) is always smaller than 2.*

Actually, there is no instance known with a gap greater than 7/6 [RS02]. Moreover, instances with a gap smaller than 1 constitute the vast majority. These are called *Integer Round-Up Property* (**IRUP**) instances. Instances with a gap greater than or equal to 1 are called **non-IRUP** instances.

In the decomposed model there are a huge number of variables. But no variable values can be exchanged without changing the solution, i.e., the model has no symmetry. This and the strength of the relaxation make it advantageous for an enumerative approach.

1.1.3 Other Models from the Literature

In the survey [dC02] the following models are described:

1. **Position-indexed formulations**

   Variables that correspond to items of a given type are indexed by the physical position they occupy inside the large objects. Formulations of this type have been used in two- and three-dimensional non-guillotine cutting and in scheduling ('time-indexed formulations').

   (a) *Arc flow model* [dC98]. Vertices represent possible positions of an item's start and end in a pattern. Forward arcs represent items or waste and backward arcs represent stock lengths. Thus, the variable $x_{p_1p_2}$ for $p_2 - p_1 \in \{l_1, \ldots, l_m\}$ denotes how many pieces of length $p_2 - p_1$ are positioned at p_1 in all patterns of the solution. Under flow conservation constraints, the flow weight is to be minimized. The author proposed three criteria to reduce the search space and symmetries.

   The model is equivalent to the Gilmore-Gomory formulation: solutions can be easily converted, i.e., a flow can be decomposed into paths representing patterns. The model was used in [dC98] as such.

The number of variables is pseudo-polynomial but most of them were not necessary and the author applied column generation. Moreover, for each position involved, flow conservation constraints are necessary, i.e., constraints were also generated and the model grew in two directions.

In [AdC03], the Gilmore-Gomory model was used (and called *cycle-flow formulation*) but branching was done in fact on the variables of the arc flow formulation because it is possible to distinguish patterns contributing to a specific $x_{p_1 p_2}$ variable.

(b) *A model with consecutive ones* is obtained by a unimodular transformation of the arc flow model, which sums each flow conservation constraint with the previous one.

2. **One-cut models**

   Each decision variable corresponds to a single cutting operation performed on a single piece of material. Given a piece of some size, it is divided into smaller parts, denoted as the first section and the second section of the one-cut. Every one-cut should produce, at least, one piece of an ordered size. The cutting operations can be performed either on stock pieces or intermediate pieces that result from previous cutting operations.

   Introduced by Dyckhoff [Dyc81], the models have a pseudopolynomial number of variables but much symmetry in the solution space. To our knowledge, the integer solution has not been tried. A variation of this model by Stadtler [Sta88] has an interesting structure: the set of constraints can be partitioned into two subsets, the first with a pure network structure, and the second composed of generalized upper bounding (GUB) constraints. Stadtler calls the Gilmore-Gomory model a *complete-cut* model.

3. **Bin-packing as a special case of a vehicle routing problem** where all vehicles are identical and the time window constraints are relaxed [DDI$^+$98]. The problem is defined in a graph where the items correspond to clients and the bins to vehicles.

1.1.4 A Class of New Subpattern Models

In the Gilmore-Gomory model we have to deal implicitly with a huge number of columns. The advantage is a strong relaxation and absence of symmetries. To reduce the number of different columns, we can restrict them to *subpatterns*, i.e., partial patterns which are combined to produce whole patterns (this idea was introduced by Profes-

sor R. Weismantel). Each subpattern can be a part of different resulting patterns; thus, all patterns have to be numbered, which brings much symmetry into the model. Moreover, the continuous relaxation may not only have non-integer pattern application intensities but also overcapacity patterns, resulting in a weak bound. However, it is possible to choose a small set of elementary subpatterns so that standard optimization software can be applied to the model.

Let 1D-CSP be defined by (m, L, $l = (l_1, \dots, l_m)$, $b = (b_1, \dots, b_m)$). Let $\overline{\mathcal{A}} = \{\overline{a}^j\}$ ($j = 1, \dots, \eta$) be a set of subpatterns such that each feasible pattern a can be represented as a sum of elements of $\overline{\mathcal{A}}$. For example, we could define $\overline{\mathcal{A}}$ as a set of half-patterns so that at most two are needed in a combination:

$$\left\{ \overline{a} \in \mathbb{Z}_+^m : \; l\overline{a} \in \left(\frac{L - l_{\min}}{2} - \frac{l_{\max}}{2}, \frac{L}{2} + \frac{l_{\max}}{2} \right] \right\} \tag{1.10}$$

or as a set of single-product subpatterns

$$\left\{ 2^p e^i : \; i = 1, \dots, m, \; p = 1, \dots, \left\lfloor \log_2 \left(\min \left\{ b_i, \left\lfloor \frac{L}{l_i} \right\rfloor \right\} \right) \right\rfloor \right\} \tag{1.11}$$

(e^i is the i-th unit vector). Note that when using half-patterns (1.10), we are guaranteed only *maximal* resulting patterns, i.e., $\{a \in \mathbb{Z}_+^m : \; 0 \le L - la < l_{\min}\}$. Thus, it is necessary to allow overproduction of items, so that the total amount of items of type i in the solution may be larger than b_i. In both (1.10) and (1.11), if we want the resulting patterns to be proper ($a \le b$), we need additional constraints, although this is done implicitly in the standard model.

Let $\overline{K}$ be an upper bound on the number of different patterns in an optimum solution. Let x_{kj} be the frequency with which subpattern $\overline{a}^j$ is used in the solution as a part of pattern k ($k = 1, \dots, \overline{K}$, $j = 1, \dots, \eta$). Let $y_{kj} = 1$ if $\overline{a}^j$ is a part of pattern k, otherwise $y_{kj} = 0$. Then $x_{kj} \le x_j^{\max} y_{kj}$, $\forall k, j$, where

$$x_j^{\max} = \min_{i:\, \overline{a}_{ij} > 0} \{ \lfloor b_i / \overline{a}_{ij} \rfloor \}, \quad \forall j. \tag{1.12}$$

Let q_k be the intensity of pattern k in the solution: $q_k \ge x_{jk}$, $\forall j, k$.

$$\min \textstyle\sum_k q_k \tag{1.13}$$

[SUBPAT] s.t.

$$\textstyle\sum_k \sum_j \overline{a}_{ij} x_{kj} \ge b_i, \qquad \forall i \tag{1.14}$$

$$x_{kj} \le x_j^{\max} y_{kj}, \quad \forall k, j \tag{1.15}$$

$$\textstyle\sum_j l\overline{a}^j y_{kj} \le L, \qquad \forall k \tag{1.16}$$

$$q_k \ge x_{kj}, \qquad \forall k, j \tag{1.17}$$

$$x_{kj}, q_k \in \mathbb{Z}_+, \qquad \forall k, j \tag{1.18}$$

$$y_{kj} \in \mathbb{B}, \qquad \forall k, j, \tag{1.19}$$

where (1.16) restricts the length of any resulting pattern.

Note that we allow a subpattern to be in a pattern not more than once (1.19) as otherwise we cannot linearly measure q_k. The same reason applies to having a separate variable x_{kj} for each k. Furthermore, the positive components x_{kj} for pattern k may have different values for different j. This creates several actual patterns, all of which satisfy (1.16), but the model 'knows' only about pattern k. Thus, the total actual number of patterns in a solution may be larger than $\overline{K}$.

In the solution of the continuous relaxation, (1.15) may allow that $y_{kj} < 1$, which leads to overcapacity patterns. The number of the constraints (1.15) and (1.17) is $\overline{K}\eta$, which is the number of x-variables. Thus, we have to choose a set $\overline{\mathcal{A}}$ with a sensible number of subpatterns or hope that not too many variables are needed. For example, it can be the single-product set (1.11).

To strengthen the relaxation, we may add the constraint

$$\sum_j l\overline{a}^j y_{kj} \le Lq_k, \qquad \forall k, \tag{1.20}$$

cf. (1.16). To restrict the resulting patterns to be proper, we need

$$\sum_j \overline{a}^j y_{kj} \le b, \qquad \forall k. \tag{1.21}$$

But the relaxation is weak:

Example 1.1.1 Consider the instance ($m = 3$, $L = 30$, $l = (6, 10, 15)$, $b = (100, 100, 100)$). An optimum can be easily constructed: $20 \times (5, 0, 0)$, $33 \times (0, 3, 0)$, $50 \times (0, 0, 2)$, $1 \times (0, 1, 0)$. Starting with this solution in the relaxation of (1.13)–(1.21) using subpatterns (1.11):

$$\overline{\mathcal{A}} = \begin{pmatrix} 1 & 2 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 2 \end{pmatrix}$$

$$\begin{array}{lccccccc} x_{1j} = & 20 & & 20 & & & & \\ x_{2j} = & & & & 33 & 33 & & \\ x_{3j} = & & & & & & 50 & \\ x_{4j} = & & & & 1 & & & \end{array} \qquad \begin{array}{l} q_1 = 20 \\ q_2 = 33 \\ q_3 = 50 \\ q_4 = 1, \end{array}$$

it is easy to come to a better solution by allowing 7 items of type 1 in the first pattern. Set $x_{11} = x_{12} = x_{13} = q_1 = 14\frac{2}{7}$ and $y_{11} = 0.15$, $y_{12} = 0.3$, $y_{13} = 0.6$ so that (1.15) is not violated. Hence (1.16) is fulfilled: $6(1 \cdot 0.15 + 2 \cdot 0.3 + 4 \cdot 0.6) = 6 \cdot 3.15 < 20 < L = 30$. The LP value is smaller by more than 5 units. A further reduction will be achieved by incorporating the subpattern (0,0,1), i.e., allowing 3 items of type 3 in the third pattern.

However, with the single-product subpatterns (1.11) it is obvious that no column generation is needed. Thus, standard optimizers can be applied. Anti-symmetry constraints like $q_k \geq q_{k+1}$, $k = 1, \ldots, \overline{K} - 1$, can reduce the search space. An interesting research topic would be the extension of the model to handle the number of open stacks (Chapter 3).

1.1.5 Problem Extension: Multiple Stock Sizes

A straightforward extension of the problem is the case with material of several lengths. As an option, the number of rods of each length can be limited. 1D-CSP with multiple stock sizes (**1D-MCSP**) is characterized by the following data:

m	Number of piece types	M	Number of stock types
$(l_1, \ldots, l_m)$	Piece lengths	$(L_1, \ldots, L_M)$	Stock lengths
$(b_1, \ldots, b_m)$	Piece order demands	$(B_1, \ldots, B_M)$	Stock supply
		$(\rho_1, \ldots, \rho_M)$	Stock prices

When stock prices are proportional to stock lengths, a material input (trim loss) minimization problem arises. Note that the problem can be infeasible if the bounds B_i are too small.

The following model was called a *machine balance model* by Gilmore and Gomory [GG63]. Let $\overline{m} = m + M$. A vector $a = (a_1, \ldots, a_{\overline{m}}) \in \mathbb{Z}_+^{\overline{m}}$ represents a *cutting pattern* if $\sum_{i=1}^{m} l_i a_i \leq \sum_{i=1}^{M} L_i a_{i+m}$ and $\sum_{i=1}^{M} a_{i+m} = 1$. The components a_i for $1 \leq i \leq m$ determine how many pieces of type i will be cut. The component a_{k+m} corresponding to the used stock type k equals one, all other components a_{i+m} for $i \in \{1, \ldots, M\} \setminus k$ equal zero.

Let n denote the number of all cutting patterns; this can be a very large number. The Gilmore-Gomory formulation of 1D-MCSP is as follows. Determine the vector of cutting frequencies $x = (x_1, \ldots, x_n)$ which minimizes the total material cost so that piece order demands and material supply bounds are fulfilled:

$$
\begin{array}{rrcll}
 & z_{\text{G\&G}}^{\text{1D-MCSP}} & = & \min cx & \\
\text{s.t.} & \sum_{j=1}^{n} a_{ij} x_j & \geq & b_i, & 1 \leq i \leq m \\
 & \sum_{j=1}^{n} a_{ij} x_j & \leq & B_{i-m}, & m < i \leq \overline{m} \\
 & x & \in & \mathbb{Z}_+^n, &
\end{array}
$$

where the objective function coefficients $c = (c(a^1), \ldots, c(a^n))$ with $c(a) = \sum_{i=1}^{M} \rho_i a_{i+m}$ are the prices of material used in the columns.

Holthaus [Hol02] argues that exact approaches are not suitable for 1D-MCSP and only heuristics are effective. However, the cutting plane algorithm in [BS02],

which is based on [Bel00], gave a gap under 3% of the largest stock length* between the strengthened continuous bound and the best solution with a time limit of 1 minute per instance on problems with $m = 40$.† Recently, Alves and Valério de Carvalho [AdC03] obtained still better results with a branch-and-price algorithm using branching rules derived from the arc flow formulation.

For the cutting plane approach, problems with more stock types ($M = 8, 16$) were easier than with a few types ($M = 2, 4$), which can be explained by the possibility of good material utilization. 1D-MCSP is more difficult for optimality proof because the possible objective values belong to the set of combinations of the material prices (a given solution is optimum when no better objective values are possible between it and some lower bound). However, the utilization of material is better than with a single stock type, cf. [GG63]. The gap between the relaxation value and the best known solution was always smaller than the largest stock price, similar to the IRUP property for 1D-CSP.

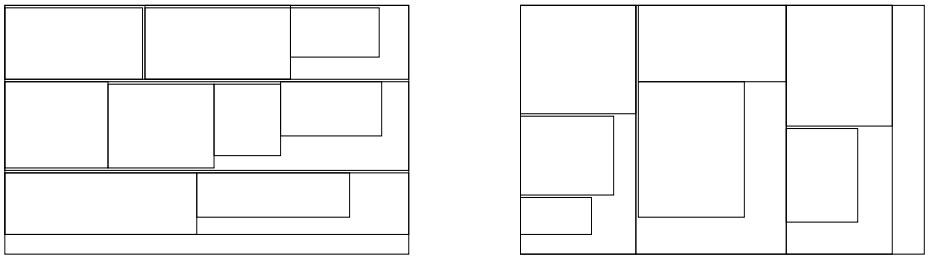
1.2 Two-Dimensional Two-Stage Constrained Cutting

The two-dimensional two-stage constrained cutting problem‡ (2D-2CP) [GG65, HR01, LM02] consists of placing a subset of a given set of smaller rectangular pieces on a single rectangular plate. The total profit of the chosen pieces should be maximized. *n-stage* cutting implies that the current plate is cut *guillotine-wise* from edge to edge in each stage and the number of stacked stages is limited to n. In two-stage cutting, strips (*strip patterns*) produced in the first stage are cut into pieces in the second stage. *Constrained* means that the pieces can be distinguished by types and the number of items of each type is bounded from above. We consider the *non-exact* case, in which items do not have to fill the strip width completely; see Figure 1.1. Pieces have a fixed orientation, i.e., rotation by 90^o is not considered.

*Note that measuring the *absolute* gap, i.e., as a percentage of the largest stock length/price, is more meaningful because for an LP-based approach the gap seems to be *absolutely* bounded; cf. the MIRUP conjecture. Indeed, the absolute gap is comparable in both large and small instances. In contrast, for heuristics the *relative* gap, i.e., as a percentage of the optimum objective value, seems more consistent, because the absolute gap increases with scale. See Chapter 3/SVC and [Yue91a].

†Pentium III, 500 MHz.

‡As it often happens, different authors use different terminology, see, e.g., [Van01, GG65]. According to the latest trend [HM03], we say *cutting problem*; *cutting stock problem* denotes more often the case when many stock pieces are considered. *Knapsack problem* is the term used, e.g., in [LM02], but it corresponds only to the logical structure and not to the physical phenomena.



a) The first cut is horizontal b) The first cut is vertical

Figure 1.1: Two-stage patterns

The problem 2D-2CP is defined by the following data: $L, W, m, (l_i, w_i, p_i, b_i)$, $i = 1, \ldots, m$, where L and W denote the length and width of the stock plate, m denotes the number of piece types and for each type $i = 1, \ldots, m$, l_i and w_i are the piece dimensions, p_i is the value, and b_i is the upper bound on the quantity of items of type i. A *two-stage pattern* is obtained by cutting strips, by default in the L-direction (1st stage), then cutting single items from the strips (2nd stage). The task is to maximize the total value of pieces obtained. If piece prices are proportional to the areas, a trim loss minimization problem arises.

The Gilmore-Gomory formulation for 2D-2CP is similar to that for 1D-CSP. Let column vectors $a^j = (a_{1j}, \ldots, a_{mj}) \in \mathbb{Z}_+^m$, $j = 1, \ldots, n$, represent all possible strip patterns, i.e., a^j satisfies $\sum_{i=1}^m l_i a_{ij} \leq L$ and $a_{ij} \leq b_i$, $i = 1, \ldots, m$ (*proper* strip patterns). Let x_j, $j = 1, \ldots, n$, be the intensities of the patterns in the solution. The model is as follows:

$$z_{\text{G\&G}}^{\text{2D-2CP}} = \min \textstyle\sum_{j=1}^n \left(\sum_{i=1}^m -p_i a_{ij}\right) x_j \tag{1.22}$$

$$\text{s.t.} \quad \textstyle\sum_{j=1}^n a_{ij} x_j \leq b_i, \quad i = 1, \ldots, m \tag{1.23}$$

$$\textstyle\sum_{j=1}^n w(a^j) x_j \leq W \tag{1.24}$$

$$x_j \in \mathbb{Z}_+, \quad j = 1, \ldots, n, \tag{1.25}$$

where $w(a^j) = \max_{i:\, a_{ij} > 0}\{w_i\}$ is the strip width. We represent the objective as a minimization function in order to keep analogy with 1D-CSP. This model can be viewed as representing a multidimensional knapsack problem.

Lodi and Monaci [LM02] propose two Integer Linear Programming formulations of 2D-2CP with a polynomial number of variables, so that no column generation is needed. Other models interpret strip pattern widths as variables. These 'assignment' formulations are investigated and compared in Chapter 4.

1.3 Related Problems

A problem closely related to 2D-2CP is *two-dimensional two-stage strip packing* (**2D-2SP**). Given a strip of fixed width and unlimited length, the task is to cut the given set of items from a minimum strip length. There are only heuristic approaches known; see [Hif99] and, for the non-guillotine case, [CLSS87]. For the two-stage guillotine case, the models and methods for 1D-CSP are straightforwardly applicable. Moreover, the same technological constraints and criteria may be of relevance (Chapter 3).

Guillotine cutting problems of three or more stages cannot be easily modeled by strip generation. Vanderbeck [Van01] did it for three stages under some special assumptions. Thus, constrained n-stage ($n \geq 3$) cutting is still a difficult problem while unconstrained is easy [GG65, Hif01].

A direct extension of 1D-CSP on two dimensions is 2D-CSP or 2D-nCSP if the n-stage cutting technology is applied. Here, a given set of rectangles must be obtained from a minimum number of stock plates. For 2D-2CSP, the column (plate layout) generation problem is exactly 2D-2CP.

In [Pee02] we find recent results and a survey on *dual bin-packing* (**1D-DBP**) and *maximum cardinality bin-packing* (**CBP**). In 1D-DBP, having the same problem data as in 1D-CSP, we fill each bin *at least* to its full capacity; the objective is to maximize the number of bins while the number of items of each type is limited. Zak [Zak02] also called 1D-DBP the *skiving stock problem* and compared it to set packing while comparing 1D-CSP to set covering. In CBP there are a limited number of bins and the objective is to maximize the number of packed items. In [Sta88] we find a simplification of 1D-CSP called 1,5-dimensional CSP: non-integer pattern multiplicities are allowed.

1.4 Solution Approaches

1.4.1 Overview

In 1961, Gilmore and Gomory proposed model (1.7)–(1.9) where each column of the constraint matrix corresponds to a feasible cutting pattern of a single stock length. The total number of columns is very large in practical instances so that only a subset of columns/variables can be handled explicitly. The continuous relaxation of this model was solved using the revised simplex method and heuristic rounding was applied to obtain feasible solutions. This approach is described in Section 1.4.3. Under conditions of cyclic regular production, the solution of the continuous relaxation is optimum for a long-term period because fractional frequencies can be transferred/stored to the next cycle. Heuristic rounding of

continuous solutions produces an integer optimum in most cases, which can be explained by the MIRUP conjecture described above.

For the general 1D-CSP, where order demands are large, LP-based exact methods have been more effective than pure combinatorial approaches. In the last several years many efforts have succeeded in solving 1D-CSP exactly by LP-based enumeration [DP03, Kup98, ST95a, dC98, Van99]. In [dC98] the arc flow formulation was used; in the others the Gilmore-Gomory formulation. Both models employ generation of necessary columns, also called *pricing*; hence, the enumerative scheme is called *branch-and-price*. A cutting plane algorithm for 1D-CSP was tested in [STMB01, BS02]. The instance sets where each method has difficulties are different; both are investigated in Chapter 2. See also a general scheme and terminology of branch-and-bound below.

The *unconstrained* two-dimensional n-stage cutting problem ($n \geq 2$) was introduced in [GG65, GG66], where an exact dynamic programming approach was proposed. The problem has since received growing attention because of its real-world applications. For the constrained version, a few approximate and exact approaches are known: [PPSG00, Bea85, HR01, Hif01, HM03, MA92, LM02]. Among exact algorithms, the most successful are LP-based enumeration without column generation [LM02] and non-LP-based enumeration [HR01].

Approaches for reduction of patterns and open stacks are surveyed in Chapter 3.

In [GS01], algorithm portfolio design is discussed. For a quasigroup completion problem, running 20 similar randomized algorithms, each on a separate processor, produced results 1000 times faster than with a single copy. In [Kup98], a branch-and-price algorithm for 1D-CSP was restarted every 7 minutes with different parameter settings, which significantly improved results.

1.4.2 Combinatorial Heuristics

We speak about combinatorial heuristics as those heuristics which do not use any LP relaxation of the problem. It should be noted that they are effective on 1D-BPP, but with larger order demands, the absolute optimality gap increases. Many contributions consider evolutionary algorithms for 1D-CSP, also with various additional production constraints, see [Fal96, ZB03]. For 2D-2CP, a genetic approach is compared against some others (with best results) in [PPSG00].

Another group of approaches is based on the structure of the problems. To begin, we should mention the various kinds of First-, Next-, and Best-Fit algorithms with worst-case performance analysis [Yue91a]. More sophisticated are the sequential approaches [MZ93, Kup98, Hae75] for 1D-CSP (such a heuristic will be used in Chapter 2 to solve residual problems and in Chapter 3 as a standalone

algorithm) and Wang's combination algorithm [Wan83] and the *extended substrip generation algorithm* (ESGA) of Hifi [HM03] for 2D-2CP.

1.4.3 Gilmore-Gomory's Column Generation with Rounding

Consider the general form of the models (1.7), (1.22) after introducing slacks:

$$z^{IP} = \min\left\{cx : Ax = b,\ x \in \mathbb{Z}_+^n\right\}, \tag{1.26}$$

where $A \in \mathbb{Q}^{m\times n}$, $c \in \mathbb{Q}^n$, $b \in \mathbb{Q}^m$. The continuous relaxation of (1.26), the *LP master problem*, can be obtained by discarding the integrality constraints on the variables:

$$z^{LP} = \min\{cx : Ax = b,\ x \in \mathbb{R}_+^n\}. \tag{1.27}$$

1.4.3.1 LP Management

We work with a subset of columns called a *variable pool* or *restricted master problem*. It is initialized with an easily constructible start basis. After solving this restricted formulation with the primal simplex method, we look for new columns. In fact, this is what the standard simplex algorithm does but it has all columns available explicitly. Let A_B be the basis matrix, and let $d = c_B A_B^{-1}$ be the vector of simplex multipliers. The *column generation problem* (*pricing problem, slave problem*) arises:

$$z^{CG} = \min\left\{c_j - da^j : j = 1,\ldots,n\right\}. \tag{1.28}$$

If the minimum *reduced cost* z^{CG} is zero, then the continuous solution is optimum. Otherwise, a column with a negative reduced cost is a candidate to improve the current restricted formulation. Problem (1.28) is a standard knapsack problem with upper bounds for 1D-CSP (or a longest path problem in the arc flow formulation):

$$z^{CG} = 1 - \max\left\{da : la \le L,\ a \le b,\ a \in \mathbb{Z}_+^m\right\}. \tag{1.29}$$

1.4.3.2 Column Generation for 1D-CSP

In practice, when implementing a column generation approach on the basis of a commercial IP Solver such as ILOG CPLEX [cpl01], we may employ their standard procedures to solve (1.28), which was done in [LM02] for 2D-2CP. An advantage of this design is the low implementation effort. Another solution ideology for Integer Programming problems is *Constraint Programming* [cpl01], which is an enumerative solution ideology coming from Computer Science. ILOG Solver

[cpl01] implements it in a universal framework containing several search strategies. Problem-specific heuristics can be integrated easily. The coding ('programming') of a model is straightforward using the modeling language or C++/Java classes allowing non-linearities; even indices can be variables. The goal of the system is to find good solutions and not to prove optimality. Thus, powerful constraint propagation routines are built in to reduce the search domain of a node. We are not aware of application to 1D-BPP directly but on the homepage of ILOG Corporation there are industry reports about column generation with this approach considering various industrial side constraints. It is sometimes used to generate all patterns *a priori*, which is possible for small instances, and solve the now complete formulation with standard IP methods.

But even for medium-sized instances, *delayed* pattern generation becomes unavoidable in an exact method because of the astronomical number of possible patterns. Traditionally there were two methods to solve (1.28): dynamic programming, applied already in [GG61], and branch-and-bound, cf. [GG63, MT90]. Dynamic programming is very effective when we are not restricted to proper patterns ($a \leq b$) [GG61]. But such upper bounds on the amount of items can only reduce the search space of branch-and-bound. Its working scheme is basically a lexicographic enumeration of patterns (after some sorting of piece types).[§] It allows an easy incorporation of additional constraints, e.g., industrial constraints and new conditions caused by branching on the LP master. However, the bounds on the objective value used to restrict the amount of search may not be sufficiently effective when many small items are present, because of a huge number of combinations. The branch-and-bound approach becomes unavoidable when general-purpose cuts are added to the LP (Section 2.4) because (1.29) becomes non-linear and non-separable.

1.4.3.3 Rounding of an LP Solution

To thoroughly investigate the neighborhood of an LP optimum on the subject of good solutions, we need an extensive rounding procedure. Steps 1–3 of Algorithm 1.4.1 were introduced in [STMB01] already. Step 4 was introduced in [BS02]; it will be referred to as *residual problem extension*. The reasons that lead to Step 4 are the following: the residual problem after Step 1 is usually too large to be solved optimally by a heuristic; on the other hand, the residual problem after Step 2 is too small, so that its optimum does not produce one for the whole problem. The introduction of Step 4 reduced by a factor of 10 the number of 1D-CSP

[§]In [GG63], for 1D-CSP with multiple stock lengths, the multiple knapsack problem (1.28) was solved by a single enumeration process (lexicographical enumeration of patterns) for all stock sizes simultaneously.

Algorithm 1.4.1 Rounding procedure

Input: a continuous solution $\overline{x} \in \mathbb{R}^n_+$
Output: a feasible solution $\widetilde{x} \in \mathbb{Z}^n_+$
Variables: $[x]$, rounded continuous solution

S1. $\overline{x}$ is rounded down: $[x] \leftarrow \lfloor \overline{x} \rfloor$.

S2. Partial rounding up: let $\overline{x}_{j_1}, \ldots, \overline{x}_{j_m}$ be the basis components of $\overline{x}$ sorted according to non-increasing fractional parts. Then try to increase by 1 the components of $[x]$ in this order:

For $i = 1, \ldots, m$
if $a^{j_i} \leq \max(\mathbf{0}, b - A[x])$ then set $[x]_{j_i} \leftarrow [x]_{j_i} + 1$;

The reduced right-hand side $b' = \max(\mathbf{0}, b - A[x])$ defines a *residual problem* (m, L, l, b').

S3. A constructive heuristic is applied to (m, L, l, b'). Its best solution $\widetilde{x}'$ is added to $[x]$ producing a feasible result: $\widetilde{x} = [x] + \widetilde{x}'$.

S4. Decrease the last component that was changed in S2 and go to S3; thus, the rounded part $[x]$ is reduced, which enlarges the residual problem. This is done up to 10 times.

---

instances which could not be solved optimally after the root LP relaxation.[¶] The constructive heuristic SVC used to solve the residual problems will be thoroughly investigated in Chapter 3.

1.4.3.4 Accelerating Column Generation

In column generation approaches it is common [JT00] to use the Lagrange relaxation [NW88] to compute a lower bound on the LP value. This allows us to stop generating further columns if the rounded-up LP value cannot be improved any more. This criterion is vital in some problems (e.g., the multiple depot vehicle scheduling problem [LD02]) where the so-called *tailing-off* effect occurs: the last columns before the optimum is reached require a very long time to generate. For 1D-CSP a stronger criterion was found by Farley [Far90].

The subgradient method was applied in [DP03] to improve the dual multipliers, generate better columns, and to strengthen the Lagrange bound. Dual cuts were used to accelerate column generation in [dC02]. These are cutting planes valid in the dual LP formulation; in the primal formulation, they are valid columns (but not patterns).

[¶] In contrast, the effect on 1D-MCSP is negligible, see also the results for 2D-2CP.

The standard pricing principle is always to choose a column with the minimum reduced cost in (1.28). This corresponds to the classical pricing scheme for simplex algorithms with an explicit constraint matrix. Modern schemes like steepest-edge pricing would make the problem (1.29) non-linear.

A good LP solver producing numerically stable data and LP management is of importance. Ten years ago, programming under DOS prohibited working with a variable pool; only the current LP basis was stored. This led to the generation of an exponentially larger number of columns than with a variable pool.

1.4.4 Branch-and-Bound

This subsection defines the basic terminology that will be used throughout the remainder of the thesis to describe derivations of enumerative algorithms.

1.4.4.1 A General Scheme

Let us consider some minimization problem P:

$$z^* = \min\{f(x);\ x \in \Omega\}, \tag{1.30}$$

where f is the objective function and Ω is the set of *feasible solutions*. A *lower bound* on its optimum objective value z^* is some value $\mathtt{lb}$ such that $\mathtt{lb} \leq z^*$ always holds. For minimization problems, it can be obtained, e.g., by solving a *relaxation* $\underline{P}$ of P

$$\underline{z} = \min\{\underline{f}(x);\ x \in \underline{\Omega}\} \tag{1.31}$$

with $\underline{\Omega} \supseteq \Omega$ and $\underline{f}(x) \leq f(x)$, $\forall x \in \Omega$. We distinguish between the *relaxation value* $\underline{z}$ and the implied lower bound $\mathtt{lb}$; the latter is the smallest possible objective value of (1.30) not smaller than $\underline{z}$, usually the next integer above: $\mathtt{lb} = \lceil \underline{z} \rceil$. Similarly, an *upper bound* $\mathtt{ub}$ satisfies $\mathtt{ub} \geq z^*$. For minimization problems, upper bounds represent objective values of some feasible solutions.

Let us consider partitioning of the set Ω into subsets $\Omega_1, \ldots, \Omega_k$ so that $\Omega = \cup_{i=1}^{k} \Omega_i$ and $\cap_{i=1}^{k} \Omega_i = \emptyset$. Then for each subset we have a *subproblem*

$$z_i^* = \min\{f(x);\ x \in \Omega_i\}, \quad i = 1, \ldots, k \tag{1.32}$$

and its lower bound $\mathtt{lb}_i$ is called the *local lower bound*. For a given subproblem, we will denote it by $\mathtt{llb}$. One can easily see that

$$\mathtt{glb} = \min_i \mathtt{lb}_i$$

is a *global lower bound*, i.e., valid for the whole problem P. Each *local upper bound* is valid for the whole problem; thus, we speak only about the *global upper*

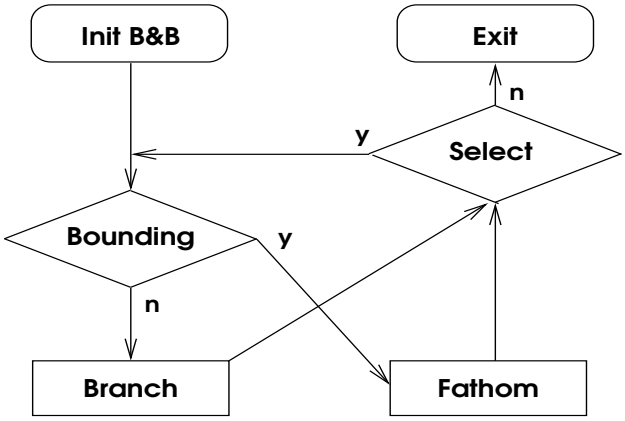


Figure 1.2: General branch-and-bound scheme

bound gub, usually the objective value of the best known feasible solution. The splitting up of a problem into subproblems by partitioning of its set of feasible solutions is called *branching*. The family of problems obtained by branching is usually handled as a *branch-and-bound tree* with the *root* being the whole problem. Each subproblem is also called a *node*. All unsplit subproblems that still have to be investigated are called *leaves* or *open nodes*.

In Figure 1.2 we see a branch-and-bound scheme that is a generalization of that from [JT00]. It begins by initializing the *list of open subproblems* $\mathcal{L}$ with the whole problem P: $\mathcal{L} = \{P\}$. Procedure BOUNDING is applied to a certain subproblem (the root node at first). It computes a local lower bound llb and tries to improve the upper bound gub. If $\texttt{llb} < \texttt{gub}$ then the node can contain a better solution (proceed by BRANCH). Otherwise the node is FATHOMed or pruned, i.e., not further considered. Note that if the root node is fathomed then the problem has been solved without branching. In this case, there are only two possibilities: either a feasible solution has been found, $\texttt{glb} = \texttt{gub} < \infty$, or the problem is infeasible, $\texttt{glb} = \texttt{gub} = \infty$. A subproblem can also be infeasible; then we obtain $\texttt{llb} = \infty$.

Procedure BRANCH splits up a given node into some subproblems and adds them to $\mathcal{L}$. As long as there are nodes in $\mathcal{L}$ with local lower bounds $\texttt{llb} < \texttt{gub}$, procedure SELECT picks one to be processed next.

1.4.4.2 Bounds

The bound provided by the Gilmore-Gomory relaxation is generally much stronger than bounds from other relaxations, see [Van00b] for 1D-CSP and [LM02] for 2D-2CP. However, for 1D-BPP other bounds are also effective and fast, e.g., in [MT90, SW99, Sch02a] and others, many combinatorial bounds are

applied which use the logic of possible item combinations. Furthermore, *problem reductions* are achieved by considering *non-dominated patterns*. In branch-and-price implementations [DP03, Van99] combinatorial pruning rules are employed in addition to the LP bound.

In 1D-CSP, the possible values of the objective function are integers and the objective value of the LP relaxation rounded up is a valid lower bound. In 2D-2CP, integer nonnegative linear combinations of piece prices p_i, $i = 1, \ldots, m$, are the possible values. Thus, the smallest combination greater than or equal to the LP value is a lower bound in 2D-2CP. A similar situation occurs in 1D-MCSP. One of the methods to construct a large set of combinations is the method with logarithmic complexity based on bit operations [Bel00].

For 2D-2CP the following non-LP bounds are used [HM03]:

1. *Geometric bound*: the one-dimensional knapsack problem considering only the areas and values of items.
2. The unconstrained problem, i.e., bounds b_i are omitted. In this case, the dynamic programming procedure of Gilmore and Gomory [GG65] produces an optimum in pseudo-polynomial time.

To strengthen a bound, we should use *effective plate sizes* $L^{\text{eff}} \leq L$ and $W^{\text{eff}} \leq W$, which are the greatest integer combinations of the corresponding item sizes not larger than L and W, respectively.

1.4.4.3 Non-LP-Based Branch-and-Bound

In an enumerative scheme which does not use an LP relaxation, both the bounds and the branching principle are different from an LP-based scheme [MT90, SW99, SKJ97, Sch02a]. In the one-dimensional problem, the schemes are effective only on 1D-BPP: increasing the order demands leads to an exponential growth of the search space [MT90]. For 2D-2CP, an exact approach is investigated in [HR01].

1.4.4.4 LP-Based Branch-and-Bound and Branch-and-Price

Here, the splitting up of the feasible set is done by adding linear constraints to the IP program (1.26) so that at each node its local relaxation is available. If cutting planes tighten the relaxation at the nodes, we speak of *branch-and-cut*. For example, using standard commercial software such as ILOG CPLEX [cpl01] we could apply branch-and-cut to the model of Kantorovich and to the subpattern model; in Chapter 4 this is carried out with some assignment models of 2D-2CP. The combination of LP-based branching with cutting planes is nowadays the most effective scheme for most (mixed-)integer programming problems [Mar01].

If columns may be generated at each node as in the model of Gilmore and Gomory, the enumeration scheme is called *branch-and-price*. A detailed description of a *branch-and-cut-and-price* (the name has been already used, see [JT00, CMLW99]) algorithm can be found in the next chapter; an example leading to branch-and-price is shown in the Appendix.

The first papers on branch-and-price appeared in the 1980's and dealt with routing problems, where the pricing problems are constrained shortest path problems [DSD84]. Applications to 1D-CSP can be found in [DP03, Kup98, ST95a, dC98, Van00a, Van99]; general overviews are in [JT00, CMLW99, VW96, Van00b]. Branch-and-price has probably not been applied to 2D-2CP before.

Chapter 2

A Branch-and-Cut-and-Price Algorithm for 1D-CSP and 2D-2CP

The one-dimensional cutting stock problem and the two-dimensional two-stage guillotine constrained cutting problem are considered in this chapter. The Gilmore-Gomory model has a very strong continuous relaxation which provides a good bound in an LP-based solution approach. In recent years, there have been several efforts to attack the one-dimensional problem by LP-based branch-and-bound with column generation (called branch-and-price) and by general-purpose CHVÁTAL-GOMORY cutting planes. When cutting planes are added to the LP relaxation, the pricing problem becomes very complex and often cannot be solved optimally in an acceptable time. Moreover, the modeling and implementation of its solution method as well as of the cutting plane apparatus for the chosen kind of cuts requires much effort. We develop a new upper bound for this pricing problem. We improve the cutting plane part of the algorithm, e.g., in terms of its numerical stability, and integrate mixed-integer GOMORY cuts. For 2D-2CP we propose a pricing procedure which enables the search for strips of different widths without repetitions. Various branching strategies and tools such as pseudo-costs and reduced cost bounding are investigated. Tests show that, for 1D-CSP, general-purpose cuts are useful only in exceptional cases. However, for 2D-2CP their combination with branching is more effective than either approach alone and mostly better than other methods from the literature.

2.1 Introduction

The basic scheme of branch-and-cut-and-price was taken from ABACUS [JT00], SIP [Mar01] and `bc-opt` [CMLW99] (there are many state-of-the-art free of charge and commercial codes implementing *branch-and-cut*, i.e., without column generation; among them are CPLEX [cpl01], MINTO, XPRESS-MP and others.) Also, branch-and-price implementations for 1D-CSP were studied, especially [Kup98, DP03, dC02]. As the basis for the development of the cutting plane part, we took [STMB01, BS02].

Note that many branch-and-price implementations employ simple cuts which do not provide finite description of the convex hull of integer solutions as is the case for CHVÁTAL-GOMORY cuts. For example, a special case of GOMORY mixed-integer cuts, where only a single previous constraint is used for generation* are applied in [Van00a]. This corresponds to widely-used practices and general beliefs expressed in the literature. Usually, in the context of an LP-based branch-and-bound algorithm, where the LP relaxation is solved using column generation, cutting planes are carefully selected in order to avoid the destruction of the structure of the pricing problem, on which its solution is based. This viewpoint is shared by numerous authors. Barnhart, Hane, and Vance [BHV00], for example, mention as one of the main contributions of their branch-and-price-and-cut algorithm "*a pricing problem that does not change even as cuts are added, and similarly, a separation algorithm that does not change even as columns are added.*" Vanderbeck [Van99] writes in the introduction of his computational study of a column generation algorithm for 1D-BPP and 1D-CSP: "*Adding cutting planes can be viewed in the same way [as branching]. However, the scheme efficiency depends on how easily one can recognize the prescribed column property that defines the auxiliary variable when solving the column generation subproblem, i.e., on the complexity of the resulting subproblem modifications.*"

Furthermore, cutting planes are commonly used to get a tight bound at a node of the branch-and-bound tree and aim to reduce the total number of nodes (e.g., [Wol98], Section 9.6). In the case of 1D-CSP, the optimum LP relaxation value of the Gilmore-Gomory formulation rounded up at the root node often equals the optimum solution. As a result, the bound is very tight, even without cuts. In 2D-2CP, the objective function has its values among integer combinations of item prices, which makes the optimality test using a lower bound much more difficult. We shall see that for 2D-2CP a combination of cuts and branching is better than either a pure cutting plane algorithm or branch-and-price.

In Sections 2.2 and 2.3 we give an overview of the branch-and-cut-and-price scheme. In Section 2.4 we discuss cutting planes, in Section 2.5 column genera-

*This corresponds to a cut-generating vector u (Section 2.4) with a single non-zero element.

tion. Computational results are presented in Section 2.6 followed by implementation details.

2.2 Overview of the Procedure

Consider again the general form of the models (1.7), (1.22) after introducing slacks (see Chapter 1):

$$z^{IP} = \min \left\{ cx : \; Ax = b, \; x \in \mathbb{Z}^n_+ \right\}, \tag{1.26}$$

where $A \in \mathbb{Q}^{m \times n}$, $c \in \mathbb{Q}^n$, $b \in \mathbb{Q}^m$. The continuous relaxation of (1.26), the *LP master problem*, can be obtained by discarding the integrality constraints on the variables:

$$z^{LP} = \min\{cx : \; Ax = b, \; x \in \mathbb{R}^n_+ \}. \tag{1.27}$$

An LP-based exact algorithm uses the LP relaxation objective value to obtain a *lower bound on the optimum objective value* (if the main problem is a minimization problem). There exist other relaxations providing lower bounds, see Chapter 1. On the other side, for the practical efficiency, we need good, possibly optimum solutions as early as possible. They provide *upper bounds on the optimum value*. The best known solution is a proven optimum when it equals some lower bound.

Let $P_{IP} = \text{conv}\{x \in \mathbb{R}^n : x \text{ is feasible for (1.26)}\}$ be the convex hull of feasible solutions. Let $\overline{x}$ be an optimum solution of the relaxation (1.27). If $\overline{x}$ is not integer, it is used to construct a feasible solution $\widetilde{x}$ of (1.26) by means of heuristic rounding (Algorithm 1.4.1). If no better value of z^{IP} is possible between $c\overline{x}$ and $c\widetilde{x}$, then $\widetilde{x}$ is an optimum. Otherwise, there exists a hyperplane $\{x \in \mathbb{R}^n : \pi x = \pi_0\}$ such that $\pi\overline{x} > \pi_0$ and $P_{IP} \subseteq \{x \in \mathbb{R}^n : \pi x \leq \pi_0\}$. The problem of finding such a cutting plane is called the *separation problem* (Section 2.4).

If we are able to find a cutting plane, we can strengthen the relaxation and iterate. This is repeated until $\overline{x}$ is a feasible solution or no more strong cuts can be found. In the latter case *branching* is initiated. This is commonly done by picking some fractional variable $\overline{x}_j$ that must be integer and creating two subproblems, one with $x_j \leq \lfloor \overline{x}_j \rfloor$ and one with $x_j \geq \lceil \overline{x}_j \rceil$. Alternatively, branching can be done on more complex hyperplanes, see the discussion below. Algorithm 2.2.1 [Mar01] summarizes, for the case of LP-based branching, the procedure introduced in Section 1.4.4.1.

The list $\mathcal{L}$ is organized as a binary tree, the so-called *branch-and-bound tree.* Each (sub)problem Π corresponds to a node in the tree, where the unsolved problems are the *leaves* of the tree and the node that corresponds to the entire problem (1.26) is the *root*. Each leaf knows the optimum LP basis and set of cuts

Algorithm 2.2.1 Branch-and-cut(-and-price)

1. Let $\mathcal{L}$ be a list of unsolved problems. Initialize $\mathcal{L}$ with (1.26).
   Initialize the global upper bound $\overline{z} = +\infty$.
2. **Repeat**
3. Choose a problem Π from $\mathcal{L}$ whose LP bound is smaller than $\overline{z}$.
   Delete Π from $\mathcal{L}$.
4. **Repeat** *(iterate)*
5. Solve the LP relaxation of Π. Let $\overline{x}$ be an optimum solution.
6. If $c\overline{x} \geq \overline{z}$, fathom Π and **goto** 12.
7. Construct feasible solutions for (1.26) using $\overline{x}$.
   (Possibly, they improve $\overline{z}$)
8. If $c\overline{x} \geq \overline{z}$, fathom Π and **goto** 12.
9. Look for violated inequalities and add them to the LP.
10. **Until** there are no strong inequalities.
11. Split Π into subproblems and add them to $\mathcal{L}$.
12. **Until** $\mathcal{L} = \emptyset$.
13. Print the optimum solution. **STOP.**

---

which were active in its parent. For the next node retrieved from storage, the optimum LP solution $\overline{x}$ of its parent is recreated. The LP is reoptimized using the dual simplex method. Then columns are generated until an LP optimum is found or infeasibility is proven. In the latter case the node is fathomed. Further, if $c\overline{x}$[†] is not smaller than $\overline{z}$ then this node can provide no better solutions and is fathomed as well. This includes the case when $\overline{x}$ is feasible for (1.26).

2.3 Features

In this section, we discuss the issues which can be found in basically every state-of-the-art branch-and-cut(-and-price) implementation and were adapted in our model. These are LP/cuts management, integer rounding, reduced cost bounding, enumeration strategy, and preprocessing. A more detailed presentation of existing features can be found in [LS99, Mar01, JT00, BP02].

[†] in 1D-CSP, $\lceil c\overline{x} \rceil$; in 2D-2CP, the next smallest integer combination of item prices.

2.3.1 LP Management

Dual simplex method is applied to the restricted master LP after adding cuts or branching constraints because dual feasibility of the basis is preserved. Primal simplex method is applied after adding columns and deleting cuts. With the best-first strategy (see below), many constraints are changed from node to node; in this case, the choice of the optimization method is left to the CPLEX LP solver, which is very powerful in fast reoptimization of a modified LP model.

To control the size of LPs and the complexity of column generation, we restrict the maximum number of added cuts and remove cuts which have not been tight for a large number of iterations at the current node. However, all cuts constructed at a node are stored and in the separation procedure we look for violated cuts among those of the current node and its parents. If no violated cuts are found, new cuts are constructed (Section 2.4).

To prevent cycling when the same deleted cuts are violated and added again iteratively, we count the number of deletions of a cut at the current node and, after each deletion, we increase the minimum number of iterations that the cut should be kept (by a factor of 1.2). But every 30 iterations we delete all inactive cuts (this number is then multiplied by 1.05) with the aim of simplifying the LP formulation and column generation. Note that only at the root node it is advisable to perform many iterations; at subnodes we allowed only 2-3 iterations because of time.

2.3.2 Rounding of an LP Solution

To construct a feasible solution of (1.26) which neighbors an optimum solution $\overline{x}$ of the relaxation, we do not take only the rounded-down part of $\overline{x}$, see, e.g., [DP03]. Some components of $\overline{x}$ with large fractional parts are also rounded up, see Algorithm 1.4.1. This can be compared to diving deeper into the branch-and-bound tree. The resulting rounded vector $[x]$ is generally not yet feasible: $b' = b - A[x]$ is the right-hand side of the *residual problem*. For 1D-CSP, the residual problem is solved by the heuristic SVC, see Chapter 3, Algorithm 3.2.1, with at most 20 iterations. SVC has already proven to yield much better performance than First-Fit in [MZ93, MBKM01]. For 2D-2CP, a variant of SVC is proposed below. Because of the importance of LP-based local search, we vary the portion of $\overline{x}$ which is taken over to $[x]$ [BS02], Step 4 of Algorithm 1.4.1.

Another way to obtain feasible solutions is to declare the variables integer and use the mixed-integer solver of CPLEX on the currently known set of columns.[‡] The rounding procedure is called at every 30th node and the MIP solver every 200 nodes with a time limit of 30 seconds.

[‡]relaxing the local branching constraints.

2.3.3 Sequential Value Correction Heuristic for 2D-2CP

For a general scheme of SVC, we refer the reader to Chapter 3. To adapt SVC for 2D-2CP, let us define a *pseudo-value* y_i of piece i so that it reflects the average material consumption in a strip. After constructing strip a containing a_i items of type i, the *current pseudo-value* y'_i is

$$y'_i = p_i \frac{w(a)}{w_i} \frac{L}{L-h},$$

where $h = L - la$ is the free length of the strip. The final pseudo-value is a weighted average of y'_i and the previous value, similar to the one-dimensional case.

After filling the plate, for those items which are not in the solution, we also distribute the space not occupied by the strips, i.e., the area $L(W - \sum_j w(a^j)x_j)$. It is divided among the piece types (added to their pseudo-values) in proportion to their free amounts b'_i. This is done in order to facilitate some exchange of items packed. The heuristic was not tested intensively because in 2D-2CP good solutions are found quickly, see the tests. The nature of SVC is to minimize trim loss (waste); value maximization, specific to 2D-2CP, could be only partially considered.

2.3.4 Reduced Cost Bounding

The idea in reduced cost bounding is to bound or fix variables by exploiting the reduced costs of the current LP solution [Mar01]. Namely, we can obtain the range in which a variable does not make the LP bound greater or equal to the upper bound. Let $\underline{z} = c\overline{x}$ be the objective value of the current LP solution, $\overline{c} = (\overline{c}_j)_{j=1,\ldots,n}$ be the corresponding reduced cost vector, and $\overline{z}^1$ be the largest possible solution value smaller than $\overline{z}$. Consider a non-basic variable x_j of the current LP solution with finite lower and upper bounds l_j and u_j, and non-zero reduced cost $\overline{c}_j$. Set $\delta = \lfloor \frac{\overline{z}^1 - \underline{z}}{|c_j|} \rfloor$. Now, if x_j is currently at its lower bound l_j and $l_j + \delta < u_j$, the upper bound of x_j can be reduced to $l_j + \delta$. In case x_j is at its upper bound u_j and $u_j - \delta > l_j$, the lower bound of x_j can be increased to $u_j - \delta$. In case the new bounds l_j and u_j coincide, the variable can be fixed to its bounds and removed from the problem. Reduced cost bounding was originally applied to binary variables [CJP83], in which case the variable can always be fixed if the criterion applies.

We did not see any improvement when using this feature. For possible reasons, see the discussion on pseudo-costs. Actually, reduced cost bounding led to a huge number of columns generated (one hundred thousand on 1D-CSP with $m = 200$)

so that the CPLEX LP solver failed. All results presented below were computed without the feature.

2.3.5 Enumeration Strategy

In the presented general outline of a branch-and-cut algorithm there are two steps which leave some choices: splitting up of a problem into subproblems and selecting a problem to be processed next. If variable j has a fractional value $\overline{x}_j$ in the local optimum LP solution, the one subproblem is obtained by adding the constraint $x_j \leq \lfloor \overline{x}_j \rfloor$ *(left son)* and the other by adding the constraint $x_j \geq \lceil \overline{x}_j \rceil$ *(right son)*. This rule is called *branching on variables*, since we need only to change bounds of variable j and be careful at column generation, see Section 2.5. The bounds can be effectively handled by simplex algorithms. Branching on more complicated inequalities or splitting up of a problem into more than two subproblems is rarely incorporated into general mixed-integer solvers. This would increase the number of constraints in the LP. In our case, this would also change the structure of the objective function in the pricing procedure because each new constraint has a dual variable. However, this produces more balanced solution trees [dC02, Van99]. Branching on variables does not produce balanced trees because right subproblems are smaller and are quickly processed. However, the latest branch-and-price algorithm for 1D-CSP [DP03, Pee02] uses this kind of branching.

2.3.5.1 Alternative Branching Schemes

In this work we investigated branching on variables. For comparison, now we discuss some alternative schemes for branching on hyperplanes of the kind

$$\sum_{j \in J^*} x_j \begin{cases} \leq \lfloor \alpha \rfloor \\ \geq \lceil \alpha \rceil \end{cases} \tag{2.1}$$

if $\sum_{j \in J^*} \overline{x}_j = \alpha \notin \mathbb{Z}$ in the node's LP solution.

1. Vanderbeck [Van99] proposes

$$J^* = \{j : \beta(a^j)_\iota = 0 \, \forall \iota \in \Upsilon_0 \quad \bigwedge \quad \beta(a^j)_\iota = 1 \, \forall \iota \in \Upsilon_1\}, \tag{2.2}$$

   where $\beta(a^j)$ is a binary representation of a^j obtained by the binary transformation [MT90] of a knapsack problem and Υ_0, Υ_1 are some index sets. It is possible to choose simple sets, e.g., with cardinalities $|\Upsilon_0| = 0$, $|\Upsilon_1| = 1$, so that the modifications of the pricing problem are acceptable.

   In [Pee02], which is an extension of [DP03], we find that the scheme of Vanderbeck is very powerful and the results are comparable to branching on variables.

2. Valério de Carvalho [dC02] and Alves [AdC03] propose branching rules induced by the arc flow formulation (AFF, Chapter 1). Given a variable $x_{p_1 p_2}$ of AFF, they define the set J^* of patterns having an item with length $p_2 - p_1$ at position p_1. Selection of AFF variables for branching starts with those corresponding to the largest items. Actually, the fractional variable used for branching that corresponds to the largest item is also the one corresponding to the item nearer to the left border of the bin.

   A comparison with this variant is done in the tests.

3. *Branching on slacks* for 2D-2CP (a new scheme). Slack variables e^i $(i = 1, \ldots, m)$ determine how many items of type i should be in a solution. Thus, branching on them is more 'spread' than on single patterns. However, this scheme does not guarantee optimality: even in a fractional LP solution, slacks may be integer. Thus, this scheme should be combined with another one. We left it for future research.

2.3.5.2 Branching Variable Selection

1. *Most infeasibility.* This rule chooses the variable with a fractional part closest to 0.5. The hope is for the greatest impact on the LP relaxation.

2. *Pseudo-costs.* This is a more sophisticated rule in the sense that it keeps the history of the success (impact on the LP bound) of the variables on which one has already branched.

Based on the results in [LS99], we implemented pseudo-costs as follows. Given a variable x_j with a fractional value $\overline{x}_j = \lfloor \overline{x}_j \rfloor + f$, $f > 0$, the *down pseudo-cost* of x_j is

$$\zeta^d = \overline{\Delta}\mathtt{llb}_j^d \cdot f,$$

where $\overline{\Delta}\mathtt{llb}_j^d$ is the average increase of the local LP value (divided by the corresponding infeasibility f) in all left ($\leq$) subproblems produced by branching on x_j. If a subproblem was infeasible, we did not set the increase equal to infinity but to the difference between the current upper bound and the parent's LP value. Similarly,

$$\zeta^u = \overline{\Delta}\mathtt{llb}_j^u \cdot (1 - f)$$

is the *up pseudo-cost*. The *pseudo-cost* of x_j at the current node is

$$\zeta = 2 \min\{\zeta^u, \zeta^d\} + \max\{\zeta^u, \zeta^d\}$$

which produced good results in [LS99].

The calculation is very fast if one has already branched on x_j before. If doing this for the first time, there are no data available. Near the root of the search tree, where branching decisions are important, we have no pseudo-costs yet. For each new variable, to initialize pseudo-costs, we applied *strong branching*: the values of $\overline{\Delta}\mathtt{llb}_j^u$ and $\overline{\Delta}\mathtt{llb}_j^d$ were calculated exactly by temporary branching (also new columns could be generated).

The computational experience [LS99] shows that branching on a most infeasible variable is by far the worst, in terms of CPU time, solution quality, and the number of branch-and-bound nodes. Using pseudo-costs gave much better results and was especially effective when the number of nodes grew large. But we saw no improvement of the lower bound or the number of nodes when using pseudo-costs. This might be explained by column generation: at almost each node many new columns are created which are only slightly different from those already in the LP, which destroys the effect. Thus, after testing of the feature (see results), it was not used.

Pure strong branching is much more expensive. But the number of nodes is smaller (about one half) compared to the pseudo-costs [Mar01]. This decrease does not compensate for higher runtimes in general. Thus, strong branching is not normally used as a default strategy.

2.3.5.3 Node Selection

1. *Best first search (bfs).* A node is chosen with the weakest parent lower bound (promising the *best* solution) or with the smallest estimate of the node's bound based on pseudo-costs. The goal is to improve the global lower bound, which is the minimum of local bounds of all unprocessed leaves. However, if this fails to occur early in the solution process, the search tree grows considerably. Another rule might decide among the nodes with equal bound, e.g., the depth.

2. *Depth first search (dfs).* This rule chooses one of the deepest nodes in the tree. Among the nodes with equal depth, one with the weakest bound is taken. The advantages of dfs are the small size of the search tree and the fast reoptimization of the subproblem. Also, good feasible solutions are found very quickly. The main disadvantage is that the global lower bound stays untouched for a long time resulting in bad solution guarantees if we stop optimization before optimum. Another drawback is that if a wrong branch is chosen at a high level, then much time is spent inside though we could prune the branch if good lower bounds had been known. In [LS99] we read that pure dfs is not good for large problems. For IRUP 1D-CSP instances, however, the correct LP bound is known from the beginning, so pure dfs

was quite efficient; see the tests. In [cpl01]/Constraint Programming it is proposed to investigate several dfs branches in parallel (interleaved).

3. *Estimate-based* strategies use pseudo-costs (see branching variable selection) and estimates of the quality of solutions obtainable at a node [LS99]. We did not test them here.

4. In order to avoid investigating wrong branches, we propose a combination of bfs and dfs which can be called *diving from a bfs node*. Some number of nodes, say, 10, are chosen and processed according to bfs. The last bfs node is taken and its children are followed downwards for, say, m nodes, i.e., its subtree is investigated in a depth-first fashion. The diving can be seen as a kind of rounding procedure. This strategy has been very important for pattern minimization (Chapter 3).

In both bfs and dfs, there remains the question, which of the 'twin' nodes of the same parent is to be preferred, with $\leq$ or $\geq$ constraint. For the Gilmore-Gomory model, many authors [Kup98, DP03] prefer $\geq$ ('right') nodes because $\leq$ implies that the corresponding column should not be generated again which complicates the pricing procedure. However, there is another reason to choose these nodes: by fixing variables at a positive value we can quickly come to feasible solutions. In [DP03, Kup98, AdC03] the enumeration strategy was always depth-first search.

2.3.6 Node Preprocessing

Node preprocessing is a powerful set of LP/IP constraint propagation rules, allowing us to tighten variable bounds, eliminate variables and constraints, or determine infeasibility before solving; see [Mar01] for a survey. In CPLEX such procedures are applied automatically in every LP formulation. The mixed-integer solver of CPLEX considers also implications arising from the integrality of variables. But we employ CPLEX directly only to solve LP's; thus, it is important to recognize such implications ourselves. For logical implications in some other IP problems, see [JT00].

In 2D-2CP, the LP bound can be tightened by finding the *effective knapsack width* $W^{\text{eff}}(\leq W)$, which is the largest integer combination of piece widths not greater than W [TLS87].

Suppose the right-hand side is reduced because some columns are bounded from below at the current node, $b' = b - \sum_{j=1}^{n} a^j \mathcal{L}_j$ where $\mathcal{L}_j$ is the lower bound of column j. In 2D-2CP also $W' = W - \sum_{j=1}^{n} w(a^j)\mathcal{L}_j$ (W' can be further reduced using integer combinations of piece widths). Then for each column we set the upper bound $\mathcal{U}_j = \min_{i:\, a_{ij}>0}\{\lfloor b'_i/a_{ij} \rfloor\}$ which can be already done at the

root node (*integer bounding*). Also, in column generation we should take care not to generate columns in which the number of items of type i is greater than b'_i.

2.4 Cutting Planes

For the Gilmore-Gomory model of the problems under investigation, no problem-specific cutting planes seem to be available as in more structured models. Thus, *general-purpose* cuts have to be used, e.g., GOMORY fractional and mixed-integer cuts. They are known to provide a finite description of the convex hull of feasible solutions. There exist algorithms with finite convergence; however, because of the implicit constraint matrix, they could not be applied. Although measures against cycling were introduced, the finiteness of the pure cutting plane approach has not been proven. Here we repeat some necessary information about these cuts (see, e.g., [NW88]) and also describe how we use them in the Gilmore-Gomory model.

2.4.1 Overview

In Subsection 2.4.2 we describe how to use superadditive cuts which are based on linear combinations of current constraints. An interesting property is that the slack variables of current cuts have coefficients in the derived cuts. Then we propose an approach to use strengthened CHVÁTAL-GOMORY cuts in a numerically stable way. Mixed-integer cuts may have non-integer slacks; this has to be considered in the derived cuts. Finally we mention how to choose the linear combinations of constraints ('basic cuts') so that the last LP solution is cut off by the new cuts.

In Subsection 2.4.3 local cuts are developed, i.e., those which consider branching constraints (variable upper bounds). These cuts are valid in the subproblem where they are created and in its children. Thus, the cut selection procedure described in Subsection 2.4.4 looks for violated cuts at the current node and its parents. Subsection 2.4.5 repeats some basic propositions from [STMB01]. The main difference in the present scheme is that for CHVÁTAL-GOMORY cuts, we no longer require that the cut-generating vectors u be positive; this modification leads to better numerical properties.

2.4.2 GOMORY Fractional and Mixed-Integer Cuts

We consider the discrete optimization problem $\min\{cx : x \in S\}$ (1.26), where $S \subseteq \mathbb{Z}^n_+$, and we formulate it as an integer program by specifying a rational polyhedron $P = \{x \in \mathbb{R}^n_+ : Ax = b\}$, where $A \in \mathbb{Q}^{m\times n}$ and $b \in \mathbb{Q}^m$, such that $S = \mathbb{Z}^n \cap P$. Hence $S = \{x \in \mathbb{Z}^n_+ : Ax = b\}$. Let $P_{IP} = \text{conv}(S)$ be the convex hull of feasible solutions. The problem can be represented by a linear program

with the same optimum solution: $\min\{cx : x \in P_{IP}\}$. We will use integrality and valid inequalities for P to construct suitable valid inequalities $\sum_{j=1}^{n} \pi_j x_j \leq \pi_0$ for the set S. For a column generation formulation, when not all the columns are available explicitly, cuts have to be described functionally because we need to calculate cut coefficients for the new generated columns (*lifting*). Thus, we define a cut as an inequality

$$\sum_{j=1}^{n} F(ua^j)x_j \leq F(ub), \tag{2.3}$$

where u is some vector producing a linear combination of the existing constraints.

Definition 1 ([NW88]) *A function $F : D \subseteq \mathbb{R}^m \to \mathbb{R}^1$ is called* superadditive *over D if $F(d_1) + F(d_2) \leq F(d_1 + d_2)$ for all $d_1,\ d_2,\ d_1 + d_2 \in D$.*

Note that $d_1 = 0$ yields $F(0) + F(d_2) \leq F(d_2)$ or $F(0) \leq 0$. Further, when F is superadditive, we assume $F(0) = 0$ and $0 \in D$.

A function $F : D \subseteq \mathbb{R}^m \to \mathbb{R}^1$ is called *nondecreasing* over D if $d_1, d_2 \in D$ and $d_1 \leq d_2$ implies $F(d_1) \leq F(d_2)$.

Proposition 1 ([NW88]) *If $F : \mathbb{R}^1 \to \mathbb{R}^1$ is superadditive and nondecreasing, then* (2.3) *is a valid inequality for $S = \mathbb{Z}^n \cap \{x \in \mathbb{R}^n_+ : Ax = b\}$ for any (A, b).*

If $\pi x \leq \pi_0$ (for short: (π, π_0)) is a valid inequality for P_{IP}, i.e., $\pi x \leq \pi_0$ for all $x \in P_{IP}$, then

$$F = F(\pi, \pi_0) = \{x \in P_{IP} : \pi x = \pi_0\}$$

defines a face of P_{IP}. If $\dim(F) = \dim(P_{IP}) - 1$ then $\pi x \leq \pi_0$ represents a facet of P_{IP} and (π, π_0) is called a *facet-defining inequality*. The valid inequalities (π, π_0) and (γ, γ_0) of P_{IP} are called *equivalent* when $(\pi, \pi_0) = \lambda(\gamma, \gamma_0) + v(A, b)$ for some $\lambda > 0$ and $v \in \mathbb{R}^m$. If they are not equivalent, then the inequality (π, π_0) *dominates* or *is stronger than* (γ, γ_0) when $\pi \geq \lambda\gamma + v(A, b)$ and $\pi_0 \leq \lambda\gamma_0 + v(A, b)$ for some $\lambda > 0$ and $v \in \mathbb{R}^m$.

If the solution is not optimum, further inequalities are constructed recursively based on linear combinations of original constraints and already added cuts. Suppose there are μ cuts in the LP. The dependence of the coefficients of cut r on the original constraints and previous cuts is shown by the following recursion:

$$\pi^r_j = \pi^r(a^j) = F(\textstyle\sum_{i=1}^{m} u^r_i a_{ij} + \sum_{t=1}^{r-1} u^r_{m+t}\pi^t(a^j)), \quad r = 1, \ldots, \mu, \tag{2.4}$$

where u^r_i, $i = 1, \ldots, m + r - 1$, are the coefficients of the linear combination of all previous constraints. Moreover, the following issue has to be kept in mind: when adding a cut, the constraint matrix is extended by a slack column,

$\sum_{j=1}^{n} F(ua^j)x_j + s = F(ub)$. Further cuts which depend on this cut have their own coefficients in this column. Thus, cut r is written as

$$\sum_{j=1}^{n} \pi_j^r x_j + \sum_{v=1}^{r-1} \pi^r(s_v)s_v + s_r = \pi_0^r. \tag{2.5}$$

To be precise, suppose for simplicity that cut slack variables are integers. Set $\pi^v(s_v) = 1$, $v = 1, \ldots, \mu$. Then coefficient $\pi^r(s_v)$ of cut r for the slack variable of cut v is $\pi^r(s_v) = F(\sum_{t=1}^{r-1} u_{m+t}^r \pi^t(s_v))$, in analogy to (2.4).

In [NW88] it is proposed to eliminate cut coefficients for slacks in the following way: just substitute $s_v = \pi_0^v - \sum_{j=1}^{n} \pi_j^v x_j$ in each cut dependent on cut v. Then in (2.5) we would have zero coefficients for s_v, $v < r$. An advantage would be that we could keep only active cuts in the LP formulation and remove those cuts on whose slacks the active cuts depend. This would also simplify the search for violated cuts, see below. But experience shows that the resulting constraints have worse numerical properties: the more cuts added, the greater the maximum absolute values of the coefficients. We add the cuts directly as in (2.5); the resulting LP has good numerical properties: even with hundreds of cuts added, the absolute values of coefficients are mostly below 1000.

CHVÁTAL-GOMORY valid inequalities for S [NW88] which were applied to 1D-CSP already in [STMB01] are constructed using the superadditive function $F(ua) = \lfloor ua \rfloor$ (rounding down). They do have the convenience of integer slack variables. Combining inequality $\lfloor uA \rfloor x \leq \lfloor ub \rfloor$ with $uAx = ub$ yields a GOMORY fractional cut $\sum_{j=1}^{n} \mathrm{fr}(ua^j)x_j \geq \mathrm{fr}(ub)$, where $\mathrm{fr}(d) = d - \lfloor d \rfloor$. It is very effective to strengthen CHVÁTAL-GOMORY cuts in the following way. If $\mathrm{fr}(ub) = ub - \lfloor ub \rfloor < 1/k$, k integer, then multiplying u by k produces a stronger or equivalent cut. Some other methods are possible, see [LL02]. After multiplication of u by an integer, it is advantageous to prevent numerical difficulties, namely too large numbers. This can be done by setting each component of u to the fractional part of its absolute value with the original sign. An equivalent cut is obtained then, see Proposition 2 below.

Mixed-integer cuts, integrated in this work, are constructed as follows. In [NW88] the following function is described: $F_\alpha(d) = \lfloor d \rfloor + \max\{0, (\mathrm{fr}(d) - \alpha)/(1 - \alpha)\}$ which is a generalization of the rounding down function $\lfloor \rfloor$. Combining inequality $F_\alpha(uA)x \leq F_\alpha(ub)$ with $uAx = ub$ yields the pure integer case $\sum_{j=1}^{n}(ua^j - F_\alpha(ua^j))x_j \geq \alpha$ of the GOMORY mixed-integer cut. The strongest cut is produced with $\alpha = \mathrm{fr}(ub)$. Inequality (2.3) with function F_α and $\alpha \neq 1$ may have a non-integer slack variable. Thus, other inequalities which depend on this cut have another type of coefficients for the slack of this cut: $\pi^r(s_v) = \overline{F}_{\alpha_r}(\sum_{t=1}^{r-1} u_{m+t}^r \pi^t(s_v))$ with $\overline{F}_\alpha(d) = \min\{0, d/(1 - \alpha)\}$ and $\pi^v(s_v) = 1$. CHVÁTAL-GOMORY cuts cannot be constructed after mixed-integer cuts because they cannot handle the non-integer slacks. Note that a mixed-integer

cut of the first rank (i.e., not dependent on any previous cuts) is stronger than the corresponding non-strengthened integer cut.

Now we should clarify where to get the multipliers u. Without column generation we would construct a fractional cut from row i of the simplex tableau with a non-integer basis variable x_i^B. This is modeled by taking row i of the optimum basis inverse for a u-vector if $x_i^B = ub \notin \mathbb{Z}$ ([NW88], basic cuts). Then for the optimum LP solution $\overline{x}$, $\lfloor uA \rfloor \overline{x} > \lfloor ub \rfloor$ holds which means that $\overline{x}$ is cut off by $\lfloor uA \rfloor \overline{x} \leq \lfloor ub \rfloor$.

While it is possible to set $-\mathbf{1} < u < \mathbf{1}$ for CHVÁTAL-GOMORY cuts, there is no such obvious transformation for mixed-integer cuts. However, no numerical difficulties were observed either. We construct all cuts which are possible and store them in a local pool; only a few most violated cuts are added to the LP, see Subsection 2.4.4.

2.4.3 Local Cuts

The term *cut-and-branch* [BP02] denotes the variation of the algorithm when cuts are constructed only at the root node; if no optimum is found and proved after a certain effort, branching is initiated and no new cuts are created in the subproblems. However, experience shows the effectiveness of creating *local cuts*, i.e., cuts valid only in a subproblem (and its descendants) [BP02].

Suppose we set lower and upper bounds on some variables as branching constraints. Then the LP of the current node looks like

$$\min\{cx : \; Ax = b, \; \mathcal{L} \leq x \leq \mathcal{U}\}.$$

Let $J = \{1, \ldots, n\}$. In an LP solution, let B be the index set of basic variables; let $L = \{j \in J : \; x_j = \mathcal{L}_j\}$, $U = \{j \in J : \; x_j = \mathcal{U}_j\}$ be the index sets of non-basic variables at their lower and upper bounds. Let A_B be the basis, $\overline{A} = A_B^{-1} A$, $\overline{b} = A_B^{-1} b$. The last simplex tableau implies

$$x_B + \overline{A}_L (x_L - \mathcal{L}_L) - \overline{A}_U (\mathcal{U}_U - x_U) = \overline{b} - \overline{A}_L \mathcal{L}_L - \overline{A}_U \mathcal{U}_U,$$

where $\mathcal{U}_U - x_U$ is substituted for the non-basic variables x_U which are at their upper bounds. Applying Proposition 1 to each row of this system with the rounding-down function, we obtain CHVÁTAL-GOMORY cuts

$$x_B + \lfloor \overline{A}_L \rfloor (x_L - \mathcal{L}_L) + \lfloor -\overline{A}_U \rfloor (\mathcal{U}_U - x_U) \leq \lfloor \overline{b} - \overline{A}_L \mathcal{L}_L - \overline{A}_U \mathcal{U}_U \rfloor$$

or, equivalently,

$$x_B + \lfloor \overline{A}_L \rfloor x_L - \lfloor -\overline{A}_U \rfloor x_U \leq \lfloor \overline{b} - \overline{A}_L \mathcal{L}_L - \overline{A}_U \mathcal{U}_U \rfloor + \lfloor \overline{A}_L \rfloor \mathcal{L}_L - \lfloor -\overline{A}_U \rfloor \mathcal{U}_U. \quad (2.6)$$

The rounding down function in (2.6) can be replaced by F_α which produces local mixed-integer cuts. Note that some variables already have upper bounds at the root node (see Node Preprocessing), thus some globally valid cuts may consider them.

2.4.4 Selecting Strong Cuts

Column generation (see next section) is expensive when many cuts are added to the LP. Thus, we have to restrict the maximum number of cuts. But to select a subset of strong cuts, we perform several iterations of the cutting plane algorithm without column generation (i.e., on the current LP). Note that when looking in a pool for cuts violated by the local LP solution, we have to compute the cut's slack s_r, see (2.5). For that, we may need to know the values of slacks of the other cuts.

2.4.5 Comparison to the Previous Scheme

In [STMB01, BS02] each cut-generating vector u was modified to satisfy $\mathbf{0} \leq u < \mathbf{1}$, see Proposition 2 from [STMB01] below. This led to large coefficients of the cutting planes: with 150 cuts in the LP, the coefficients exceeded 10^9.

Using the concept of dominance, we discarded superfluous inequalities as follows. Let $\overline{A}$ denote the original matrix A or an already extended (by generated inequalities) coefficient matrix, and let $\overline{m}$ and $\overline{n}$ be the number of rows and columns in $\overline{A}$, respectively, after introducing slack variables in order to get equality constraints again. The columns of $\overline{A}$ are denoted by $\overline{a}^j$, $j \in \overline{J} = \{1, \ldots, \overline{n}\}$. In correspondence to $\overline{A}$ the extended right-hand side $\overline{b}$ can be defined.

Proposition 2 *For any valid inequality $(\lfloor v\overline{A} \rfloor, \lfloor v\overline{b} \rfloor)$ of P_{IP} with $v \in \mathbb{R}^{\overline{m}}_+$ there exists an equivalent inequality $(\lfloor u\overline{A} \rfloor, \lfloor u\overline{b} \rfloor)$ with $0 \leq u < e$, where e has all elements equal to 1.*

Proof Let $\lfloor v\overline{A} \rfloor x \leq \lfloor v\overline{b} \rfloor$ be a valid inequality of P_{IP}. Using $v = w + u$ where $w = \lfloor v \rfloor$, we have $0 \leq u < e$, $\lfloor v\overline{A} \rfloor = \lfloor (u + w)\overline{A} \rfloor = \lfloor u\overline{A} \rfloor + w\overline{A}$ and $\lfloor v\overline{b} \rfloor = \lfloor (u + w)\overline{b} \rfloor = \lfloor u\overline{b} \rfloor + w\overline{b}$. Since $w\overline{A}x = w\overline{b}$ for all $x \in P_{IP}$, the inequalities $\lfloor v\overline{A} \rfloor x \leq \lfloor v\overline{b} \rfloor$ and $\lfloor u\overline{A} \rfloor x \leq \lfloor u\overline{b} \rfloor$ are equivalent. ∎

Proposition 3 *Let $(\pi = \lfloor u^0\overline{A} \rfloor, \pi_0 = \lfloor u^0\overline{b} \rfloor)$ be a dominant valid inequality of P_{IP} for some $u^0 \in \mathbb{R}^{\overline{m}}_+$. Then there exists $u \in \mathbb{R}^{\overline{m}}_+$ with $\pi = \lfloor u\overline{A} \rfloor$ and $\pi_0 = \lfloor u\overline{b} \rfloor$ such that $u\overline{a}^j \in \mathbb{Z}$ for some $j \in \overline{J}$.*

The proof can be done by constructing a contradiction to the dominance of (π, π_0) if such an index j does not exist.

The quality of a valid inequality $(\lfloor u\overline{A}\rfloor, \lfloor u\overline{b}\rfloor)$ can be measured (in some sense) by $\mathrm{fr}(u\overline{b})$. Valid inequalities are not of interest in the case where $\mathrm{fr}(u\overline{b}) = 0$. As will be seen, $\mathrm{fr}(u\overline{b})$ should be large enough to get closer to P_{IP}.

Proposition 4 *Let $u \in \mathbb{R}_+^{\overline{m}}$ and $k \in \mathbb{Z}$ be given with $k > 1$ and $k\,\mathrm{fr}(u\overline{b}) < 1$. Let $v = ku$. Then $(\lfloor v\overline{A}\rfloor, \lfloor v\overline{b}\rfloor)$ dominates the valid inequality $(\lfloor u\overline{A}\rfloor, \lfloor u\overline{b}\rfloor)$, or they are equivalent.*

Proof Because $v\overline{b} = ku\overline{b}$ and $\mathrm{fr}(v\overline{b}) = \mathrm{fr}(ku\overline{b}) = \mathrm{fr}(k\lfloor u\overline{b}\rfloor + k\,\mathrm{fr}(u\overline{b})) = \mathrm{fr}(k\,\mathrm{fr}(u\overline{b})) = k\,\mathrm{fr}(u\overline{b})$, since $k\,\mathrm{fr}(u\overline{b}) < 1$, $\lfloor v\overline{b}\rfloor = \lfloor ku\overline{b}\rfloor = ku\overline{b} - \mathrm{fr}(ku\overline{b}) = k\lfloor u\overline{b}\rfloor$ holds. Hence, $k\lfloor u\overline{A}\rfloor x \le \lfloor ku\overline{A}\rfloor x = \lfloor v\overline{A}\rfloor x \le \lfloor v\overline{b}\rfloor = k\lfloor u\overline{b}\rfloor$. ■

Let R_u be the set of fractional parts of $(u\overline{A})$, i.e., $R_u = \{\mathrm{fr}(u\overline{a}^j) : j \in \overline{J}\}$. If $\gcd(\, u_1, \ldots, u_{\overline{m}}\,) = d$, then $R_u \subseteq \{0, d, \ldots, 1-d\}$.

Proposition 5 *Let $\overline{a}^s$ be a column with $u\overline{a}^s \in \mathbb{Z}_+ \setminus \{0\}$, $s \in \overline{J}$. If $\mathrm{fr}(u\overline{b}) < \max\{r : r \in R_u\}$,[§] then $F(\lfloor u\overline{A}\rfloor, \lfloor u\overline{b}\rfloor)$ is not a facet of P_{IP}.*

From Propositions 4 and 5 it follows that in order to strengthen a cut, its vector u can be multiplied by some $k \in [2, 1/\,\mathrm{fr}(u\overline{b})) \cap \mathbb{Z}$ to increase $\mathrm{fr}(ku\overline{b})$. This was done also in the new implementation for CHVÁTAL-GOMORY cuts. None of these properties holds for mixed-integer cuts.

2.5 Column Generation

In the standard column generation process, the dual multipliers are used to assess the cutting pattern. With cutting planes added, it is necessary to anticipate the values of the coefficients the column will have in the cuts during this process. These coefficients are non-linear and non-separable functions (2.4) of the elements of the cutting pattern vector. Finally, the dual information from the cut rows helps to evaluate the attractiveness of the column. Thus, selecting the most attractive column involves the solution of an integer programming problem with a non-separable objective function, which does not seem to have any usable structure, and has to be solved with branch-and-bound. It is much more difficult than the linear knapsack problem for column generation arising when no cuts are added in the LP.

First, we mention how to avoid generation of columns which are already in the LP – i.e., forbidden columns. This is necessary when some variables have upper bounds and can be done easily in the context of a branch-and-bound method. In order to outline the principles of column generation with general-purpose cutting

[§]One further regularity condition is not relevant in practice, see [STMB01].

planes, we begin with the one-dimensional case. The branching strategy was taken from [STMB01]. Here we repeat it and then introduce a new upper bound and a pricing procedure for 2D-2CP.

2.5.1 Forbidden Columns

During column generation we are not allowed to consider columns bounded from above in the LP formulation. The reason is that in the simplex algorithm with variable bounds, non-basic variables x_j being at their upper bounds $\mathcal{U}_j$ are internally substituted by $\mathcal{U}_j - x_j$ which changes the sign of the reduced cost.

During pricing we search for a best pattern, excluding those which are bounded from above at the current node. That means that, on finding a better solution that is forbidden, we do not save it; enumeration is continued [Kup98, DP03]. Pieces with negative dual multipliers cannot be excluded from consideration because they may be contained in *second-best* solutions.

Experience shows that during the solution process there are only a few cases where the knapsack objective function is improved, i.e., the checking of forbidden columns does not happen often. However, we store the set of LP columns in a dictionary (balanced tree) provided by the C++ Standard Template Library so that a quick search is possible.

2.5.2 Column Generation for 1D-CSP

Let $d_1, \ldots, d_m$ be the simplex multipliers of the demand constraints (1.8). Without cutting planes, the pricing problem is the standard knapsack problem which can be solved, e.g., with branch-and-bound [MT90]. Pieces are sorted according to $d_1/l_1 \geq \ldots \geq d_m/l_m$ and feasible cutting patterns are enumerated in a decreasing lexicographical order.

Suppose μ cuts have been added; their dual multipliers are $d_{m+1}, \ldots, d_{m+\mu}$. The pricing problem is the knapsack problem

$$\max\{\overline{c}(a) : \; a = (a_1, \ldots, a_m) \in \mathbb{Z}_+^m, \; \textstyle\sum_{i=1}^m l_i a_i \leq L, \; a \leq b\} \tag{2.7}$$

with the non-linear objective function

$$\overline{c}(a) = \textstyle\sum_{i=1}^m d_i a_i + \sum_{r=1}^{\mu} d_r \pi^r(a). \tag{2.8}$$

$\overline{c}(a)$ is non-separable in the components of a because of the cut coefficients (2.4). Hence, dynamic programming cannot be applied, at least in its standard concept.

2.5.2.1 Enumeration Strategy

In our branch-and-bound method for the pricing problem, when cuts are added to the master LP, we preserve the same enumeration principle as without cuts, i.e., lexicographical order. However, sorting of pieces has to reflect their approximate impact on the objective function. To estimate this impact, the objective function can be linearly approximated by omitting the non-linear superadditive function in the formula for cut coefficients (2.4). Thus, we define *approximative cut coefficients* recursively as

$$\widetilde{\pi}^r(a^j) = \sum_{i=1}^{m} u_i^r a_{ij} + \sum_{t=1}^{r-1} u_{m+t}^r \widetilde{\pi}^t(a^j), \quad r = 1, \ldots, \mu. \tag{2.9}$$

Now we have linearity: $\widetilde{\pi}^r(a) = \sum_{i=1}^{m} a_i \widetilde{\pi}^r(e^i)$ for all r, where e^i is the i-th unit vector. The approximative objective function is then

$$\widetilde{c}(a) = \sum_{i=1}^{m} d_i a_i + \sum_{r=1}^{\mu} d_{m+r} \widetilde{\pi}^r(a) = \sum_{i=1}^{m} \widetilde{d}_i a_i \tag{2.10}$$

with $\widetilde{d}_i = d_i + \sum_{r=1}^{\mu} d_{m+r} \widetilde{\pi}^r(e^i)$. The sorting of pieces according to $\widetilde{d}_1/l_1 \geq \ldots \geq \widetilde{d}_m/l_m$ proved very effective in finding good solutions [STMB01].

2.5.2.2 A Bound

In order to avoid full enumeration, we need to estimate from above the objective value $\overline{c}(a)$ for a given set of patterns. This estimation should be easy to calculate.

The idea of the bound proposed in [BS02] is to estimate the change of a cut coefficient when adding a piece of type i to the current subpattern: $\lfloor u(a + e_i) \rfloor \leq \lfloor ua \rfloor + \lceil u_i \rceil$. However, usually we want to add several items and this could not be represented satisfactorily in such a way. Here we propose the obvious estimation $ua - 1 \leq \lfloor ua \rfloor \leq ua$ or, generally, $ua - \alpha \leq F_\alpha(ua) \leq ua$.

Proposition 6 *The objective function $\overline{c}(a)$ of* (2.7) *is bounded from above as*

$$\overline{c}(a) \leq \widetilde{c}(a) + \sum_{r=1}^{\mu} \widetilde{\alpha}_r(d_{m+r}) \tag{2.11}$$

for all patterns a, where the approximation error of cut r *is*

$$\widetilde{\alpha}_r(x) = \begin{cases} x\overline{\alpha}_r, & x \geq 0 \\ x\underline{\alpha}_r, & x < 0 \end{cases} \tag{2.12}$$

and the upper/lower approximation errors of cut r *are*

$$\overline{\alpha}_r = \sum_{t<r} \widetilde{\alpha}_t(u_{m+t}^r), \quad \underline{\alpha}_r = -\alpha - \sum_{t<r} \widetilde{\alpha}_t(-u_{m+t}^r).$$

Proof Assume $\widetilde{\pi}_r(a) + \underline{\alpha}_r \le \pi_r(a) \le \widetilde{\pi}_r(a) + \overline{\alpha}_r$ for $r \le k-1$ (for $k = 2$ trivial) which implies the proposition is true for $\mu \le k-1$. Then

$$\pi_k(a) \stackrel{\text{def}}{=} F_{\alpha_k}(\textstyle\sum_{i=1}^m u_i^k a_i + \sum_{t<k} u_{m+t}^k \pi_t(a)) \\ \stackrel{!}{\le} \textstyle\sum_{i=1}^m u_i^k a_i + \sum_{t<k}(u_{m+t}^k \widetilde{\pi}_t(a) + \widetilde{\alpha}(u_{m+t}^k)) \stackrel{\text{def}}{=} \widetilde{\pi}_k(a) + \overline{\alpha}_k$$

(remember that the components of u may be either positive or negative) and

$$\pi_k(a) \stackrel{!}{\ge} -\alpha_k + \textstyle\sum_{i=1}^m u_i^k a_i + \sum_{t<k}(u_{m+t}^k \widetilde{\pi}_t(a) - \widetilde{\alpha}(-u_{m+t}^k)) \stackrel{\text{def}}{=} \widetilde{\pi}_k(a) + \underline{\alpha}_k.$$

Thus, for the non-linear part of $\overline{c}()$, we get
$\sum_{r=1}^k d_{m+r}\pi_r(a) \le \sum_{r=1}^k d_{m+r}\widetilde{\pi}_r(a) + \sum_{r=1}^k \widetilde{\alpha}(d_{m+r})$. ∎

This bound is linear and much easier to calculate than the original objective function. The approximation errors can be calculated *before enumeration* because they are constant.

The enumeration procedure works as follows: having a partial solution $a' = (a'_1, \ldots, a'_p)$, we check the value of the upper bound (2.11) on it. If it is greater than the current lower bound, then we calculate the objective function and check whether it is a better solution; if it is, the lower bound is increased.

Now, there may be some free space in the knapsack: $dL = L - la'$. The question arises whether to fill this space by adding further items or modify a' by deleting some items from it, thus moving to another subset of solutions. In other words, we need to decide whether the subset of patterns with the first p components determined by a' **and some other non-zero components** can contain an improvement.

Proposition 7 *Let a partial solution $a' = (a'_1, \ldots, a'_p)$ be given. Then an upper bound for $\overline{c}()$ on all the patterns built by adding some items of types $i = p+1, \ldots, m$ to a' is given as follows:*

$$\max\{\overline{c}(a) : la \le L,\ a \le b,\ a \in \mathbb{Z}_+^m,\ (a_1, \ldots, a_p) = a',\ \exists i > p :\ a_i > 0\} \\ \le ub(p, dL(a')) + \widetilde{c}(a') + \textstyle\sum_r \widetilde{\alpha}_r(d_{m+r}), \quad (2.13)$$

where

$$ub(p, dL) = \max\{\widetilde{c}(\Delta a) : l\Delta a \le dL,\ \Delta a \le b,\ \Delta a \in \mathbb{Z}_+^m, \\ \Delta a_i = 0\, \forall i \le p,\ \Delta a \ne 0\} \quad (2.14)$$

and $dL(a') = L - la'$.

Proof follows from the linearity of $\widetilde{c}()$ and Proposition (6). ■

Note that $ub(p, dL)$ is not dependent on the current objective value $\overline{c}(a')$. For all possible p and dL, $ub(p, dL)$ can be calculated efficiently using dynamic programming **before** the main enumeration. However, we found the following simplification sufficient in practice:

Proposition 8 *Let* $\bar{i} = \arg\max\{\widetilde{d}_i : i > p\}$. *If* $\widetilde{d}_{\bar{i}} < 0$ *then* $ub(p, dL) \leq \widetilde{d}_{\bar{i}}$. *Otherwise* $ub(p, dL) \leq \max\{\widetilde{d}_i/l_i : i > p\} \times dL$.

Proof for the case $\widetilde{d}_{\bar{i}} \geq 0$ follows from the relaxation of $ub(p, dL)$. Otherwise recall that we must add at least one item in the free space dL. ■

The case $\widetilde{d}_{\bar{i}} < 0$ means that we are going to investigate **solutions with a smaller bound** than the bound of the current partial solution. However, a smaller bound does not mean that they may not have a greater objective value. Moreover, after encountering a forbidden pattern we have to look for alternative solutions, even with a worse value.

2.5.2.3 Practical Issues

The bound (2.13) is sometimes not strong enough when many cuts are added. If a good solution (with a negative reduced cost) is found and enumeration has been performed for several thousand backtracking steps (this limit should be increased with each column generation to reduce the number of columns/tailing off), we terminate, see [BS02]. In the current implementation the following feature proved very practical: if no good solution is found for several million steps, the number of cuts is reduced, both at the current node and at other nodes of the branch-and-price tree.

2.5.3 The Two-Dimensional Case

Let d_0 be the simplex multiplier of the width constraint (1.24), $d = (d_1, \ldots, d_m)$ those of the upper bound constraints (1.23). The pricing problem is

$$z^{2CG} = \min \{\underline{c}(a) = \textstyle\sum_{i=1}^{m}(-p_i - d_i)a_i - d_0 w(a) : \\ a \in \mathbb{Z}_+^m,\ la \leq L,\ a \leq b,\ w(a) \leq W\}. \quad (2.15)$$

The constraint $w(a) \leq W$ may be relevant if W is reduced by branching on the main problem. Assume pieces are sorted so that $w_1 \geq w_2 \geq \cdots \geq w_m$. To reduce the problem to a series of 1D knapsacks, we consider each piece type to initialize

the strip, which also fixes the width. When taking piece i first, we can add only those pieces that are not wider.

$$z^{2CG} = \min\{z'(-p-d, L-l_{i_0}, \widetilde{b}^{(i_0)}) - p_{i_0} - d_{i_0} - d_0 w_{i_0} : \\ i_0 = 1, \ldots, m,\ w_{i_0} \le W,\ b_{i_0} > 0\}, \quad (2.16)$$

where

$$z'(\widetilde{d}, \widetilde{L}, \widetilde{b}) = \min\{\widetilde{d}a :\ a \in \mathbb{Z}_+^m,\ la \le \widetilde{L},\ a \le \widetilde{b}\} \quad (2.17)$$

and

$$\widetilde{b}_1^{(i_0)}, \ldots, \widetilde{b}_{i_0-1}^{(i_0)} = 0, \quad \widetilde{b}_{i_0}^{(i_0)} = b_{i_0} - 1, \quad \widetilde{b}_i^{(i_0)} = b_i,\ \forall i > i_0. \quad (2.18)$$

The initial upper bound for each solution of (2.17) is set using the best known solution value. The proposed scheme allows us to avoid repeated investigation of equal strip patterns. This scheme also applies if cuts are added.

Note that in the assignment model M2 (Chapter 4), boolean variables q_k are distinguished to initialize a strip. This is a similar approach.

2.6 Computational Results

We used the ILOG CPLEX 7.5 Callable Library to solve the LP relaxations and the mixed-integer solver included there to find integer solutions on the current restricted LP. The hardware was AMD K7 Athlon XP 1000 MHz. The program was compiled with GNU C++ 2.96 whose optimized code was about 3 times faster than non-optimized.¶

Notations for results are given in Table 2.1.

2.6.1 The One-Dimensional Cutting Stock Problem

At first we present some benchmark results for (today's) large instances: $m = 200$ and 400. Then we consider a 'hard28' set which consists of what appeared to be the most difficult instances among the huge series computed by Schoenfield [Sch02a]. Here we compare the branching strategies bfs and dfs. On this set and on the so-called triplet problems, our implementation is compared to a branch-and-price implementation based on the arc flow formulation and to implementations of [DP03, Kup98].

¶Some of the test instances for 1D-CSP can be obtained from `www.math.tu-dresden.de/~capad`; test instances for 2D-2CP from `ftp://panoramix.univ-paris1.fr/pub/CERMSEM/hifi/2Dcutting/`

Table 2.1: Notations for test results

v_1, v_2	$l_i \in [v_1 L, v_2 L) \cap \mathbb{Z}$
IP	the best integer solution value
IPmin, IPmax	the minimum / maximum best solution value among the class
inst_name*	*: the 1D-CSP instance is non-IRUP
LP	the strongest LP value
LPval0	the initial LP value (no cuts/branching)
tLP, tIP	time to obtain the best LP bound and IP, seconds
tOpt	optimum solution time
Nopt, $N\overline{\text{opt}}$	number of optimally solved / unsolved instances
N^+	number of instances unsolved at the root node
nodes	number of processed nodes
iter	number of cutting plane iterations in all nodes
LPcol, IPcol	number of columns in the initial LP optimum / all columns
MIP	number of cases in which the best solution was found by the CPLEX MIP solver
nIRUP	number of proven non-IRUP 1D-CSP instances
n	the total number of items in 2D-2CP, i.e., $\sum b_i$

2.6.1.1 Benchmark Results

To benchmark the behavior of the algorithm on 1D-CSP, we could not use any classes of previous authors because they computed up to $m = 100$ with at most 100 instances per class which is at the present time not enough to find difficulties.[‖] We compute $N = 1000$ instances per class for $m = 200$ and 400 and one difficult class with $N = 10000$.

Moreover, a comparison to other algorithms is questionable because we have an excellent rounding procedure, which reduces the number of instances unsolved in the root by a factor of 10 [BS02], and an excellent heuristic, see the values of N^+ in Table 2.2. In [BS02], with $N = 100$ instances per class which were all solved optimally, we observed a comparable behavior of pure cuts and pure branching in the problem with a single stock length. On the benchmark classes presented below, cuts left more instances unsolved. This becomes more clear on difficult instances, see the results for the hard28 set.

Following the tradition established in [WG96], we generate instances with

[‖] Some 1D-BPP instances had larger m, see, e.g., the triplet instances below. Some other 1D-BPP classes, e.g., in the OR Library, have 'formally' up to 1000 piece types, but when $L = 150$ and l_i are integer, the actual number of distinct piece types must obviously be much smaller.

item lengths l_i uniformly distributed in $[v_1 L, v_2 L) \cap \mathbb{Z}$, $v_1 < v_2$, where

$$v_1 \in \{0.001, 0.05, 0.15, 0.25\} \quad \text{and} \quad v_2 \in \{0.2, 0.3, \ldots, 0.8\}, \tag{2.19}$$

and order demands uniformly distributed in $[b_{\min}, b_{\max}) \cap \mathbb{Z}$. L is chosen as $100000 = 10^5$ because for smaller L we usually have too many equal sizes in such a class as $(v_1, v_2) = (0.15, 0.2)$.** We define the *basic class* by

$$(v_1, v_2) = (0.001, 0.7), (b_{\min}, b_{\max}) = (1, 100), m = 200. \tag{2.20}$$

By always varying just one or two of these parameter groups at a time, relative to the basic class, we obtain different classes and avoid an explosion of the number of classes that would arise, e.g., if we were to take all (v_1, v_2) combinations both for $m = 200$ and $m = 400$. A reason to choose $(v_1, v_2) = (0.001, 0.7)$ as basic is that it had the largest number of unsolved instances: 6 out of 1000 (23 at the root node), see Table 2.2. Actually, the instances in the hard28 set (see the next subsection), which were collected from a huge test, also have piece sizes distributed over nearly this same range.

The time limit was 2 minutes per instance. No cuts were used because pure branch-and-price was slightly better than branch-and-cut-and-price, namely the number of instances unsolved in a fixed time limit was 20–50% smaller. The branching strategy was diving bfs with 10 nodes selected according to bfs, then m nodes dfs in the subtree of the last bfs node, and so on.

At first attempt we were unable to solve the classes with $v_2 = 0.2$. The many small items produce a huge amount of combinations, which makes pattern generation by branch-and-bound inefficient. Then we introduced early termination as in the case with cuts: after 10^6 backtracking steps, if a pattern with a negative reduced cost is found, we exit the knapsack solver. This allowed the solution of classes with $v_1 = 0.001$ and $v_1 = 0.05$ ($v_2 = 0.2$). Note that for $v_1 = 0.05$ there are 6 unsolved instances but no single node could be processed in the given time. In the class $(v_1, v_2) = (0.15, 0.2)$, piece sizes and dual multipliers are so close to each other that not far from LP optimum, no patterns with negative reduced cost could be found in an acceptable time. For this class, some combinatorial reductions are very effective [Sch02a]. The branching on hyperplanes induced by the arc flow formulation seems also appropriate because then we can employ dynamic programming for pattern generation even at subnodes (no columns are forbidden).

2.6.1.2 The hard28 Set

We consider the 28 hard 1D-BPP instances from the thorough test of Schoenfield [Sch02a] with $m \in \{140, \ldots, 200\}$. One property of these instances is that the LP

** Equal items are merged, resulting in a slightly smaller average dimension.

Table 2.2: 1D-CSP: benchmark results

		$N\overline{\text{opt}}$	IPmin	IPmax	tLP	tIP	nodes	IPcol	MIP	nIRUP	N^+
$v_1 : 0.001,$	$v_2 : 0.2$	0	811	1216	18.6	18.9	0.00	894	0	0	0
	$v_2 : 0.3$	3	1214	1822	8.3	11.7	0.002	1027	0	1	4
	$v_2 : 0.4$	2	1617	2428	8.0	8.5	0.05	1186	0	0	5
	$v_2 : 0.5$	4	2020	3034	17.2	17.6	0.57	1435	0	0	4
	$v_2 : 0.6$	3	2423	3640	11.7	12.3	0.79	1289	0	0	10
	$v_2 : 0.7$	6	2826	4263	4.1	5.6	4.16	1083	2	0	23
	$v_2 : 0.8$	0	3229	5370	1.6	2.2	0.08	845	2	0	17
$v_1 : 0.05,$	$v_2 : 0.2$	6	1055	1470	8.6	12.4	0.00	788	0	0	6
	$v_2 : 0.3$	0	1459	2075	5.7	5.8	0.00	1063	0	0	0
	$v_2 : 0.4$	0	1863	2679	6.6	6.6	0.01	1212	0	0	1
	$v_2 : 0.5$	1	2266	3284	15.1	15.4	0.16	1402	0	0	3
	$v_2 : 0.6$	2	2670	3890	5.4	6.2	1.35	1160	0	0	13
	$v_2 : 0.7$	1	3075	4661	2.5	3.1	1.28	977	7	0	14
	$v_2 : 0.8$	1	3482	5573	1.1	1.6	2.76	779	9	1	14
$v_1 : 0.15,$	$v_2 : 0.2$	-	-	-	-	-	-	-	-	-	-
	$v_2 : 0.3$	1	1929	2595	8.2	8.4	0.06	1283	0	0	2
	$v_2 : 0.4$	2	2354	3200	12.5	12.8	0.24	1425	0	0	6
	$v_2 : 0.5$	5	2760	3806	5.4	6.5	2.08	1267	0	0	17
	$v_2 : 0.6$	1	3166	4414	2.4	3.1	1.52	1051	8	0	11
	$v_2 : 0.7$	1	3577	5175	1.1	1.4	3.21	850	5	0	6
	$v_2 : 0.8$	0	3989	6209	0.4	0.4	0.02	681	3	0	5
$v_1 : 0.25,$	$v_2 : 0.3$	0	2854	3750	0.8	0.8	0.00	459	0	0	0
	$v_2 : 0.4$	0	2854	3750	2.6	2.6	0.00	887	0	0	0
	$v_2 : 0.5$	0	3306	4523	3.2	3.5	0.15	1135	5	0	4
	$v_2 : 0.6$	0	3666	5094	1.7	1.8	0.03	1395	1	0	2
	$v_2 : 0.7$	0	4082	6073	1.2	1.2	0.00	1328	0	0	0
	$v_2 : 0.8$	0	4663	7232	0.6	0.6	0.00	942	0	0	0
$v_1 : 10^{-4},$	$N : 10^4$	32*	2711	4400	4.2	5.1	2.52	1092	16	1	145
$v_1 : 10^{-4},$	$b_{\max} : 1$	1	63	79	8.1	8.6	0.40	1386	0	1	10
$m : 400$		15	6197	8138	40.7	42.3	0.31	2274	0	0	24

*: for this class from 10000 instances, all others from 1000 instances.

relaxation value is integer or slightly less which makes it difficult to find an optimum for the 23 IRUP instances among them. Schoenfield was able to solve them with a branch-and-bound algorithm that employs various combinatorial bounds. Pure branch-and-price implementations from [Kup98, DP03] solved only about half of the set (unofficial results from the authors). Pure cuts achieved the same only if the CPLEX MIP solver was used to obtain integer solutions of the current LP. Our new implementation and branch-and-price based on the arc flow formulation [AdC03] were successful, see Table 2.3.

Table 2.3 contains some problem data, then algorithmic results in four sections: pure branch-and-price from [AdC03] (**BP-AFF**) and our implementation (**BP**), branch-and-cut-and-price (**BCP**) with at most two cuts in the LP formulation and new cuts generated every eight nodes, and BCP '/200' where cuts are generated only after the initial 200 nodes have been processed (see, e.g., [MPT99]). In this delayed variant, 5 cuts were allowed in the LP and were generated at every node. The CPLEX mixed-integer solver was not used because it takes a very long time on these instances. Note that in IRUP instances, tOpt is equal to the time required to find the best integer solution; in non-IRUP instances, it is the time required to raise the LP bound to the IP value.

We see that without cuts, the time and number of nodes are smaller for practically all instances except for the difficult non-IRUP instances bpp716 and bpp359. Cuts are necessary to solve bpp716 in reasonable time (no branching is needed then; the number of nodes is zero). The variant with delayed cut generation seems to be a wrong compromise: in comparison to BP, the number of nodes (not the time) is reduced for bpp716 and bpp359, but not as much as in BCP; for other instances, no systematic effect is seen.

BP–AFF was executed on a 700 MHz Pentium III with 128 MB and CPLEX 6.5. We see that there are no such extremal solution times as in our case for bpp716. The number of flow variables in the initial relaxation is smaller than the number of patterns generated in the initial Gilmore-Gomory relaxation. But that number is pseudo-polynomial; it grows with L. In these problems we have $L = 1000$. The authors agreed to compute the set for comparison and mentioned that their implementation is 'fresh' and many improvements are possible, in both the acceleration of column generation and (rounding) heuristics.

In BCP we allowed at most two cuts in the LP. In 1D-CSP usually not a single cut is active in IRUP instances (cf. [BS02]) and only one in non-IRUP instances; adding too many cuts complicates column generation. GOMORY mixed-integer cuts were used because they seem more effective for 1D-CSP. A possible explanation is that mixed-integer cuts of low ranks are stronger than fractional cuts; in 2D-2CP, at high ranks, the non-integer slacks of previous cuts make them weaker, see Section 2.4.

The pure bfs (best-first search) node selection strategy is better only for non-

Table 2.3: The hard28 set of the bin-packing problem

					BP – AFF			**BP** – no cuts			**BCP**				**BCP** '/200'	
#	name	IP	LPval0	LPcol	tOpt	nod	LPcol	tOpt	nod	IPcol	tOpt	nod	iter	IPcol	tOpt	nod
1	bpp14*	62	60.998	770	129.34	386	421	5.04	164	958	7.92	102	121	935	5.01	164
2	bpp832	60	59.997	771	234.98	984	423	2.75	30	889	5.75	11	40	820	2.75	30
3	bpp40	59	58.999	877	180.12	412	553	16.3	665	1137	24.7	142	171	1080	34.2	369
4	bpp360	62	62.000	924	64.36	23	417	3.26	12	995	8.26	33	62	1152	3.25	12
5	bpp645	58	57.999	958	193.38	515	471	4.1	18	1102	7.95	44	75	1146	4.13	18
6	bpp742	64	64.000	788	62.88	23	442	0.94	0	788	0.93	0	0	788	0.94	0
7	bpp766	62	61.999	794	47.77	10	448	2.46	19	852	3.61	23	44	860	2.46	19
8	bpp60	63	62.998	730	56.63	54	415	4.74	168	941	43.4	720	756	1089	4.76	168
9	bpp13	67	67.000	1011	80.04	2	605	4.33	13	1044	4.67	4	26	1042	4.34	13
10	bpp195	64	63.996	1028	132	63	584	12.2	227	1637	8.65	20	58	1208	19.6	247
11	bpp709	67	66.999	854	77.67	30	580	6.88	125	1063	6.37	25	50	1020	6.89	125
12	bpp785	68	67.994	982	134.19	110	575	6.52	61	1276	11.1	30	52	1200	6.54	61
13	bpp47	71	71.000	950	71.8	16	425	2.09	0	950	2.07	0	0	950	2.07	0
14	bpp181	72	71.999	882	97.23	140	492	3.86	40	993	5.81	40	68	987	3.84	40
15	bpp359*	76	74.998	741	63.1	62	386	427	24484	1425	26.2	536	570	1029	229	2310
16	bpp485	71	70.997	1098	69.14	29	470	3.57	3	1125	9.64	78	110	1274	3.55	3
17	bpp640	74	74.000	809	64.98	15	428	2.59	12	904	13.4	93	130	1132	2.59	12
18	bpp716*	76	75.000	845	41.07	10	382	7468	270082	2035	2.32	0	13	868	25657	68576
19	bpp119*	77	76.000	1056	69.76	2	633	3.17	2	1061	5.3	2	20	1074	3.15	2
20	bpp144	73	73.000	867	103.06	42	543	8.48	128	1148	232	768	808	1511	8.36	128
21	bpp561	72	71.996	967	161.75	139	553	6.05	28	1201	11.2	30	78	1222	6.01	28
22	bpp781	71	70.999	1157	277.04	352	606	7.88	30	1437	12.7	28	74	1482	7.89	30
23	bpp900	75	74.996	1115	78.65	11	541	7.43	77	1331	31.5	156	197	1362	7.45	77
24	bpp175*	84	83.000	981	104.11	114	506	3.84	8	1063	5.81	24	42	1110	3.84	8
25	bpp178	80	79.995	1054	124.75	115	548	9.33	207	1343	6.77	33	70	1208	65.9	420
26	bpp419	80	79.999	1161	91	6	659	12.4	246	1507	39.3	238	291	1606	20.1	243
27	bpp531	83	83.000	929	60.97	15	391	2.63	2	963	5.67	17	52	1156	2.65	2
28	bpp814	81	81.000	922	78.2	27	398	3.88	19	1090	8.58	28	74	1218	3.88	19
average				929.3	105.4	132.4	496.3	287.2	10602.5	1152.1	19.7	115.2	144.7	1126.0	932.9	2611.6

Note: without cuts (BP) and at the delayed variant ('/200'), pure bfs strategy was applied to bpp716, diving bfs in all other cases

IRUP 1D-CSP instances. Here we practically always have an optimum solution from the beginning; the task is to improve the LP bound. For example, the non-IRUP instance 18 (bpp716) is solved without cutting planes in about six hours with diving bfs, but with pure bfs it is solved in a little over two hours with just over 270000 nodes. With cuts we need just seconds (and no branching) for this instance. There are other known instances of this kind. IRUP problems have a strong LP bound. The goal is usually to find an optimum solution; this is where dfs works best. As the share of non-IRUP instances is small among randomly generated instances, we conclude that dfs is the first-choice strategy for 1D-CSP and 1D-BPP.

2.6.1.3 Triplet Problems

Triplet problems [Fal96] are 1D-BPP or 1D-CSP instances where an optimum solution has exactly three items in each stock bin and no waste. This means that the optimum value is equal to the material bound. In the OR Library of J. E. Beasley there is a collection of four 1D-BPP sets, each with 20 triplet instances. The dimensions of instances (m) are 60, 120, 249, and 501 in the four sets. The LP bound cannot be improved and cutting planes are of no use. Without cuts, the largest set with $m = 501$ was solved in 50 sec. per instance on average, which is almost identical to [Pee02]. With two cuts allowed in the LP, the time exceeded already 300 seconds.

2.6.1.4 Other Algorithm Parameters for 1D-CSP

We found the following settings best. Instead of solving the formulation where overproduction of items is allowed ($\sum_j a_{ij}x_j \geq b_i$, $\forall i$), we solved the equality formulation. The consequences are that non-maximal patterns may appear and that the CPLEX MIP solver has fewer feasible combinations on the restricted formulation. Initializing the root LP by a subdiagonal FFD-like matrix was better than by an SVC solution, or a greedy basis,[††] or an empty basis. The last is a valid starting point because we must have *artificial slacks* e^i $\forall i$ with large objective coefficients (e.g., 10^5): in a left subproblem (produced by $x_j \leq$) the restricted LP of the parent may need further columns to become feasible, in which case such slacks make it possible to obtain simplex multipliers to generate some new columns. Moreover, when applying integer bounding (see Node Preprocessing), an FFD/greedy/SVC starting basis may also be infeasible, in which case the artificial slacks are necessary.

We also tried the so-called *level cut* [AdC03] $\sum_j x_j \geq \lceil z^{LP} \rceil$ but this made other cuts less effective and yielded no overall improvement.

[††] Pattern $a^{i'}$ is equal to $\arg\max\{la : la \leq L,\, a \leq b,\, a \in \mathbb{Z}^m_+,\, a_i = 0\, \forall i < i'\}, i' = 1, \ldots, m.$

2.6.2 Two-Dimensional Two-Stage Constrained Cutting

For 2D-2CP, we show the effects of pseudo-costs and compare the performance to other state-of-the-art approaches. The test set consists of a *medium* class [HR01, LM02] and a *large* class [HM03]. In the medium class, the first 14 instances are *weighted*, i.e., piece prices are not proportional to the areas, and 24 are *unweighted*. In the large class, the first 10 are unweighted and the other 10 are weighted. Problem dimensions m and $n = \sum b_i$ are shown in Table 2.4.

In guillotine cutting, we can distinguish between the first cut made horizontally or vertically, or, in other words, along the first dimension (L) or the second (W). Unlike in 1D-CSP, where we practically always have either the optimum LP bound (for IRUP instances) or the optimum solution obtained by rounding (for non-IRUP instances) from the beginning, in 2D-2CP the objective function and the LP bound can accept various values during optimization. Thus, we report both the times for the best LP bound (not LP value) tLP and for the best integer solution tIP; the maximum of them is the total solution time, at most 10 minutes.

2.6.2.1 Pseudo-Costs

In Table 2.4 we see that pseudo-costs lead to longer overall optimum times on all classes, though on the large class the number of instances unsolved in the time limit of 10 minutes is the same: four. On the large class, however, pseudo-costs lead to finding good solutions much faster. The best LP value is not shown if it equals the optimum. Pseudo-costs were not used in further tests.

2.6.2.2 Comparison with Other Methods

In Tables 2.5 and 2.6 we show, for the medium set, the optimum solution times of three other solution approaches. The combinatorial algorithm from [HR01] was tested on an UltraSparc10 250 MHz. Its runtimes are denoted by HR. M1 and M2 [LM02] are assignment formulations of 2D-2CP with a polynomial number of variables; no column generation is needed. In Chapter 4 we strengthen them by lexicographical constraints and recompute with CPLEX 7.5 on our 1000 MHz CPU. BCP has fewer difficult instances and its average time is smaller on most classes.

Tables 2.7 and 2.8 show the results for the large series. The first two result columns show the objective value and the solution time[‡‡] of the heuristic HESGA [HM03]. Then there follow the data for M1 and M2. M1 provides better solutions than the heuristic, especially on the last 10 weighted instances. The weighted case

[‡‡] Also UltraSparc10 250 MHz.

Table 2.4: 2D-2CP: medium+large instances, effects of pseudo-costs, first cut along the first dimension

			NoPsCosts				PsCosts			
name	m	n	tLPbest	tIPbest	IPbest	LPbest	tLPbest	tIPbest	IPbest	LPbest
HH	5	18	0.02	0	10689		0.02	0		
2	10	23	0.95	0	2535		0.72	0		
3	19	62	0.14	0.05	1720		0.15	0.06		
A1	19	62	0.03	0.06	1820		0.06	0.09		
A2	20	53	0.24	0.02	2315		0.36	0.02		
STS2	30	78	0.29	0.11	4450		0.29	0.11		
STS4	20	50	5.12	1.15	9409		4.8	4.91		
CHL1	30	63	56.4	0.15	8360		119	0.15		
CHL2	10	19	0	0	2235		0.02	0		
CW1	25	67	0.05	0	6402		0.05	0.02		
CW2	35	63	0.05	0.02	5354		0.06	0.03		
CW3	40	96	0.29	0.11	5287		0.35	0.11		
Hchl2	35	75	2.42	1.92	9630		3.88	0.55		
Hchl9	35	76	0.09	0.09	5100		0.11	0.06		
average			4.72	0.26			9.28	0.44		
2s	10	23	0.4	0	2430		0.6	0.02		
3s	19	62	0.06	0.03	2599		0.06	0.03		
A1s	19	62	0.02	0.02	2950		0.03	0.02		
A2s	20	53	0.03	0.03	3423		0.05	0.02		
STS2s	30	78	0.14	0.12	4569		0.15	0.14		
STS4s	20	50	1.2	0.05	9481		1.03	0.05		
OF1	10	23	0	0	2713		0.02	0.02		
OF2	10	24	0.15	0	2515		0.18	0		
W	19	62	0.15	0	2623		0.2	0		
CHL1s	30	63	0.23	0.12	13036		0.21	0.14		
CHL2s	10	19	0.12	0	3162		0.15	0.02		
A3	20	46	0.21	0.03	5380		0.21	0.03		
A4	19	35	0.28	0.11	5885		0.35	0.14		
A5	20	45	0.48	0.24	12553		0.66	0.33		
CHL5	10	18	0.02	0.02	363		0.02	0.02		
CHL6	30	65	3.82	0.37	16572		5.35	0.41		
CHL7	34	75	0.54	0.28	16728		0.48	0.35		
CU1	25	82	0.08	0.06	12312		0.09	0.08		
CU2	34	90	0.03	0.03	26100		0.03	0.03		
Hchl3s	10	51	0.24	0.02	11961		0.21	0.02		
Hchl4s	10	32	1.38	1.38	11408		1.31	0.67		
Hchl6s	22	60	0.8	0.06	60170		0.55	0.06		
Hchl7s	40	90	2.59	0.3	62459		2.9	0.6		
Hchl8s	10	18	0.7	0.08	729		1.11	0.75		
average			0.57	0.14			0.66	0.16		
ATP30	38	192	0.64	0.18	140168		1.95	0.29	140168	
ATP31	51	258	213	147	820260		265	1.86	820260	820261
ATP32	56	249	0.71	0.71	37880		0.69	0.67	37880	
ATP33	44	224	0.05	0.05	235580		0.05	0.03	235580	
ATP34	27	130	589	0.06	356159	356392.6	600	0.05	356159	356325.7
ATP35	29	153	591	0.12	614429	614752.2	569	0.11	614429	
ATP36	28	153	90.1	0.39	129262		5.16	0.47	129262	129262.9
ATP37	43	222	75.5	0.08	384478		600	0.08	384478	384514.8
ATP38	40	202	0.06	0.05	259070		0.08	0.05	259070	259070
ATP39	33	163	181	66.6	266135		2.71	1.54	266135	
average			174.11	21.52	324342.10		204.46	0.52	324342.10	
ATP40	56	290	0.42	0.37	63945		0.47	0.41	63945	
ATP41	36	177	0.05	0.03	202305		0.06	0.03	202305	
ATP42	59	325	543	0.2	32589	32657.65	305	0.2	32589	
ATP43	49	259	124	69.4	208998		190	54.5	208998	
ATP44	39	196	596	338	70901	72972.48	595	0.03	70798	72841.34
ATP45	33	156	0.02	0	74205		0.02	0	74205	
ATP46	42	197	0.08	0.03	146402		0.06	0.03	146402	
ATP47	43	204	20.9	15.8	144317		599	0.41	144317	
ATP48	34	167	0.18	0.09	165428		0.21	0.11	165428	
ATP49	25	119	5.29	3.19	206965		5.42	2.94	206965	
average			128.99	42.71	131605.50		169.52	5.87	131595.20	

Table 2.5: 2D-2CP: medium instances, first cut along the first dimension

			HR	ILP-M1	ILP-M2	BCP, CHVÁTAL-GOMORY cuts						BCP, MI cuts	
	IP	LPval0	tOpt	tOpt	tOpt	tLP	tIP	nodes	iter	LPcol	IPcol	tLP	tIP
HH	10689	10872	0.2	0.05	0.03	0.1	0	0	2	11	12	0	0
2	2535	2658.943	2.9	0.14	0.1	1.6	0	628	669	25	97	1.87	0
3	1720	1893.6	0.2	0.12	0.11	0.17	0.4	88	99	40	53	0.17	0.16
A1	1820	1872.8	0.2	0.24	0.17	0.2	0.1	8	17	40	42	0.1	0.1
A2	2315	2391.552	0.8	0.32	0.28	0.8	0.2	28	36	43	57	0.89	0.2
STS2	4450	4529.035	5.6	6.88	4.81	0.33	0.5	44	57	61	71	0.26	0.5
STS4	9409	9580.129	9.2	4.54	0.86	6.15	0.2	1238	1279	41	162	4.29	0.2
CHL1	8360	8768.395	610	2.79	2.95	1m30	1.19	4332	4377	65	599	4m43	2.5
CHL2	2235	2272.176	0.1	0.03	0.04	0.1	0	0	1	22	22	0.1	0
CW1	6402	6820.133	1	1.02	0.28	0.5	0	10	18	52	58	0.81	0.1
CW2	5354	5462.694	2.1	0.31	0.23	0.4	0.1	6	14	72	74	0.2	0.1
CW3	5287	5427.329	6.4	0.83	0.48	0.16	0.14	18	32	83	94	0.19	0.17
Hchl2	9630	9742.215	N/A	12.86	20.18	5.88	5.41	576	704	75	143	9.24	5.4
Hchl9	5100	5195	858	2.09	1.05	0.2	0.2	0	1	74	75	0.2	0.2
average			115.13	2.30	2.26	7.47	0.495					21.5	0.57
2s	2430	2553.943	4.6	0.15	0.16	0.73	0	288	340	24	63	1.4	0.1
3s	2599	2668.578	0.1	0.08	0.08	0.5	0	28	41	39	42	0.5	0.1
A1s	2950	2970.37	0.1	0.11	0.16	0.1	0	0	1	39	39	0.1	0.1
A2s	3423	3474.3	0.2	0.54	0.58	0.2	0	6	14	42	44	0.7	0.1
STS2s	4569	4614.475	1.1	4.29	7.17	0.5	0.1	8	17	61	61	0.19	0.1
STS4s	9481	9643.37	8.9	3.12	7.41	1.41	0.2	328	366	41	99	2.41	0.2
OF1	2713	2713	0.1	0.02	0.05	0	0	0	0	22	22	0	0
OF2	2515	2716.333	0.1	0.09	0.07	0.7	0	40	55	22	34	0.2	0
W	2623	2785	0.2	0.23	0.13	0.28	0.1	100	108	39	84	0.16	0.1
CHL1s	13036	13193.85	6.1	1.96	2.36	0.8	0.1	12	20	61	66	44.14	0.1
CHL2s	3162	3306.882	0.1	0.07	0.07	0.1	0.1	54	66	22	33	0.17	0
A3	5380	5572.824	0.3	0.66	0.87	0.8	0.1	22	33	41	56	0.34	0.1
A4	5885	6100.765	1.6	0.77	0.59	0.25	0.2	60	76	39	68	0.38	0.2
A5	12553	13182.84	2.4	1.51	0.85	2.66	0.1	546	565	42	144	2	0.1
CHL5	363	379	0.1	0.02	0.03	0	0	0	1	22	22	0	0
CHL6	16572	16866	38.2	6.8	17.48	12.65	0.2	1532	1611	62	218	18.39	0.2
CHL7	16728	16813.35	44.6	9.28	65.08	0.38	0.2	54	65	71	80	0.3	0.2
CU1	12312	12500	1	9.01	2.08	0.7	0.1	14	22	51	54	0.5	0.1
CU2	26100	26250	2.7	2.01	0.7	0.7	0.2	6	15	69	70	0.1	0.2
Hchl3s	11961	12093.09	15.3	450.7	4.45	0.79	0	288	311	22	160	0.69	0
Hchl4s	11408	11753.54	507	192.4	2.12	1.5	0.98	264	321	23	114	2.14	0.71
Hchl6s	60170	60948.64	14.8	4.02	15.7	0.61	0.2	130	156	46	69	1.73	0.2
Hchl7s	62459	63154.5	96.2	56.9	280.32	3.47	0.64	284	309	82	112	11.61	1.42
Hchl8s	729	766.8	26.4	0.16	0.17	0.78	0.68	176	223	26	153	1.13	0.96
average			32.18	31.04	17.03	1.09	0.105					3.63	0.14

Table 2.6: 2D-2CP: medium instances, first cut along the second dimension

			HR	ILP-M1	ILP-M2	BCP, CHVÁTAL-GOMORY cuts						BCP, MI cuts	
	IP	LPval0	tOpt	tOpt	tOpt	tLP	tIP	nodes	iter	LPcol	IPcol	tLP	tIP
HH	9246	9813	0.3	0.08	0.06	0.14	0.1	92	113	13	45	0.14	0.1
2	2444	2530.621	3.8	0.26	0.15	0.47	0	108	132	24	94	0.47	0
3	1740	1893.976	0.2	0.61	0.22	0.14	0.2	52	63	39	65	0.34	0.2
A1	1820	1991.594	0.3	0.53	0.19	1.25	0.17	422	443	40	172	0.51	0.33
A2	2310	2441.667	0.7	0.29	0.22	0.55	0.4	160	181	44	76	0.96	0.4
STS2	4620	4620	0.5	0.39	0.69	0	0	0	0	61	61	0	0
STS4	9468	9689.5	1.9	1.12	3.38	0.26	0.2	90	98	41	68	0.11	0.2
CHL1	8208	8264.386	1496	1.98	2.29	0.2	0.11	16	28	65	70	0.42	0.34
CHL2	2086	2119.632	0.1	0.09	0.05	0.1	0	0	3	21	22	0.1	0
CW1	6402	6557.76	1	0.24	0.18	0.1	0	0	1	52	52	0.4	0.1
CW2	5159	5629.514	2.2	0.53	0.37	14.83	0.5	2338	2353	73	569	17.42	0.5
CW3	5689	5708.568	6.3	2.47	0.43	0.2	0.1	0	1	82	82	0.2	0.1
Hchl2	9528	9604.816	N/A	72.04	52.15	0.82	0.65	72	105	73	97	1.16	0.78
Hchl9	5060	5076.154	1017	2.69	2.07	0.5	0	0	5	73	75	0.4	0
average			195	5.95	4.46	1.40	0.17					1.62	0.22
2s	2450	2545.253	4.8	0.24	0.14	0.69	0.1	170	199	24	102	0.53	0.1
3s	2623	2785	0.1	0.34	0.2	0.28	0	100	108	39	84	0.14	0.1
A1s	2910	2937.565	0.2	0.26	0.24	0.1	0.1	0	1	39	39	0.1	0
A2s	3451	3531.478	0.2	1.18	0.91	0.1	0.1	0	1	42	42	0.5	0.1
STS2s	4625	4638	0.8	2.51	2.7	0.1	0.1	0	1	61	61	0.1	0
STS4s	9481	9541.182	1.9	1.71	5.64	0.4	0.2	2	12	41	43	0.7	0.1
OF1	2660	2660.621	0.1	0.01	0.05	0	0	0	0	21	21	0	0
OF2	2522	2696	0.1	0.1	0.08	0.6	0.1	340	351	21	136	0.11	0.1
W	2599	2668.578	0.1	0.09	0.09	0.7	0.1	28	41	39	42	0.4	0.1
CHL1s	12602	12672.84	21.5	9.31	227.05	0.5	0.2	6	15	61	63	0.72	0.2
CHL2s	3198	3198	0.1	0.09	0.1	0.1	0	0	0	22	22	0.1	0
A3	5403	5453.032	0.4	0.8	1.03	0.4	0.1	12	20	42	42	0.2	0.1
A4	5905	5961.957	2.1	0.53	0.38	0.8	0.1	28	39	40	44	0.5	0.1
A5	12449	12630	2.5	2.21	6.29	0.44	0.1	138	157	42	68	10.74	0.1
CHL5	344	381.5	0.1	0.04	0.03	0.5	0.5	22	33	22	31	0.8	0.2
CHL6	16281	16470	33.4	10.39	474.65	0.95	0.5	200	218	62	103	1.16	0.5
CHL7	16602	16682.86	50.6	17.46	344.08	0.76	0.2	110	131	71	96	10.58	0.2
CU1	12200	12342	1	235.33	0.98	0.45	0.1	220	235	51	94	0.25	0.1
CU2	25260	26192	2.8	5.3	2.32	40.35	0.54	3604	3624	69	946	9.6	7.79
Hchl3s	11829	11997.83	20.4	321.52	1.11	0.58	0.1	206	231	23	50	0.57	0.2
Hchl4s	11258	11418.88	268.4	0.62	0.22	0.17	0.8	70	86	21	54	0.36	0.8
Hchl6s	59853	60499.31	7.2	9.19	102.02	3.47	0.4	676	729	48	155	19.12	0.4
Hchl7s	62845	63052.51	39.1	62.87	350.94	0.13	0.8	14	27	84	89	0.11	0.11
Hchl8s	791	820	28.7	0.11	0.09	0.17	0.2	43	88	24	50	0.23	0.25
average			20.28	28.43	63.39	2.06	0.05					2.25	0.35

Table 2.7: 2D-2CP: large instances, first cut along the first dimension

	HESGA		M1		M2		BCP		
name	obj	time	obj	time	obj	time	obj	tAll	tIP
ATP30	140168	7.3	138765	608.13	137813	601.98	140168	0.65	0.06
ATP31	818512	21.4	818364	610.74	813748	612.92	820260	601	11.5
ATP32	37880	6.2	37503	606.24	36940	609.01	37880	0.11	0.06
ATP33	234565	19.4	235580	608.25	233016	613.11	235580	0.09	0.09
ATP34	356159	12.7	356159	604.86	354962	600.42	356159	1.25	0.06
ATP35	613784	14.8	610554	601.92	611109	600.97	614429	230	8.32
ATP36	129262	6.4	129262	604.53	129262	361.2	129262	3.3	2.86
ATP37	382910	26.4	381224	608.7	380592	609.88	384478	4.85	0.06
ATP38	258221	14.2	259070	604.9	257540	610.21	259070	0.05	0.03
ATP39	265621	11.9	266135	603.05	264470	604.57	266135	46.4	1.59
average	323708.20	14.07	323261.60	606.13	321945.20	582.43	324342.10	88.77	2.46
unsolved				10		9		1	
ATP40	63945	12.1	63945	608.53	63622	610.07	63945	1.15	0.8
ATP41	202305	10.5	202305	119.9	202305	5.32	202305	0.03	0.03
ATP42	32589	21.8	32589	614.45	32589	253.23	32589	7.8	0.15
ATP43	208571	12.4	208998	414.94	208998	50.96	208998	158	97.1
ATP44	70678	12.5	70940	604.72	70940	42.55	70916	601	1.37
ATP45	74205	11.2	74205	31.75	74205	1.21	74205	0.02	0.02
ATP46	146402	14.1	146402	42.05	146402	10.68	146402	0.06	0.05
ATP47	143458	13.7	144317	105.93	144317	10.48	144168	601	0.05
ATP48	162032	12.5	165428	34.99	165428	7.77	165428	0.27	0.17
ATP49	204574	9.2	206965	602.58	206965	43.42	206965	601	0.08
average	130875.90	13.00	131609.40	317.98	131577.10	103.57	131592.10	197.03	9.98
unsolved				4		1		3	

Table 2.8: 2D-2CP: large instances, first cut along the second dimension

	HESGA		M1		M2		BCP		
name	obj	time	obj	time	obj	time	obj	tAll	tIP
ATP30	140007	14.5	139622	612.08	138904	616.03	140067	601	31.7
ATP31	818296	19.3	814827	609.04	811507	611.59	821073	1.38	0.94
ATP32	37744	8.2	37973	607.97	37478	612.67	37973	2.75	0.08
ATP33	234538	12.3	233743	611.11	233703	612.41	234670	0.5	0.46
ATP34	353590	15.8	357741	601.47	357741	200.41	357741	0.18	0.18
ATP35	614132	17.1	614336	605.01	614336	241.56	614132	601	0.06
ATP36	128814	12.1	128306	604.6	126238	618.6	128814	2.38	1.23
ATP37	385811	15.1	385811	606.71	379956	618.29	385811	0.21	0.21
ATP38	258040	14.4	257979	606.35	256629	617.93	258812	601	208
ATP39	265330	12.2	266378	471.11	261942	621.83	266378	601	0.41
average	323630.20	14.10	323671.60	593.55	321843.40	537.13	324547.10	241.14	24.33
unsolved				9		8		4	
ATP40	65044	16.2	65584	185.41	65584	78.79	65584	601	1.18
ATP41	195453	15.4	196559	187.25	196559	23.45	196559	49	26.2
ATP42	32937	25.1	33012	610.8	33012	52.88	33012	0.03	0.03
ATP43	212062	15.8	212062	79.84	212062	11.48	212062	0.05	0.03
ATP44	69732	9.3	69784	606.77	69784	45.49	69784	2.01	1.22
ATP45	69857	6.4	69929	603.99	69988	21.8	69988	602	356
ATP46	147021	12.2	147021	62.73	147021	5.36	147021	0.03	0.03
ATP47	142935	10.2	142935	117.12	142935	12.7	142935	601	0.24
ATP48	160318	19.5	162458	131.66	162458	11.83	162458	8.98	0.03
ATP49	210169	17.4	211784	210.99	211784	21.44	211784	601	0.37
average	130552.80	14.75	131112.80	279.66	131118.70	28.52	131118.70	246.51	38.53
unsolved				3		0		4	

is much easier for M1 and M2 than the unweighted; M1 seems to provide better solutions but worse bounds than M2 (fewer optimally solved instances).

Finally there follow the results for BCP: the best objective value, the overall time and the time to find the best solution. The average objective value is mostly better than in the other approaches; the time to obtain the best solution is mostly negligible. No special difficulties are seen on the unweighted part, in contrast to M1 and M2.

2.6.2.3 Algorithm Parameters for 2D-2CP

Table 2.9 contains, for each performance indicator, 3 lines which show results for pure branching, pure cutting plane algorithm (at most 40 cuts in the LP), and pure optimization (no rounding heuristics, feasible solutions could be obtained only as integer LP solutions). We see that pure cuts are slightly weaker than pure branching but the combined approach is the best, even without rounding. This shows the importance of local cuts at the nodes. Though the average best solution value without rounding is smaller than for pure branching on the weighted large set, the number of optimally solved instances is the same as with rounding. We also see that increasing the rounding frequency from every 100 nodes (rnd100, as in pure branching/cuts) to 10 nodes (rnd10) leads to better solutions on the unweighted large set. We can see that finding an optimum integer solution is much easier for BCP than the optimality proof.

From Tables 2.5 and 2.6 it follows that strengthened CHVÁTAL-GOMORY cuts are more effective for 2D-2CP than GOMORY mixed-integer cuts. At most 5 cuts were allowed in the LP and this number is necessary, because in 2D-2CP many cuts are usually active in an LP optimum, while in 1D-CSP only 1, at most 2 cuts are active. For the instance sets being discussed, 5 cuts is the best limit. Only the best-first search strategy produces sensible results for 2D-2CP. FFD starting basis was the best in all aspects, as in 1D-CSP.

2.7 Implementation

There are 3 software libraries providing the basic functionality of branch-and-cut-and-price: ABACUS, `bc-opt` and SYMPHONY. In our specialization of BCP, very much implementation effort was devoted to cut management and to column generation with cuts, while the branching scheme demanded only about one-third of the work. As the cutting plane algorithm was already available, the branching scheme was implemented from scratch.

As usual, all values, obtained from the LP solver, should be handled with some numerical tolerance, in particular the cut-generating vectors u, e.g., when

Table 2.9: 2D-2CP: medium+large instances, pure approaches: pure branching, pure cuts, effects of rounding

First cut along L				First cut along W			
wei	unwei	largeU	largeW	wei	unwei	largeU	largeW
Nopt							
(14/14)	(22/24)	(3/10)	(1/10)	(14/14)	(23/24)	(4/10)	(2/10)
(9/14)	(10/24)	(3/10)	(5/10)	(10/14)	(14/24)	(3/10)	(4/10)
(14/14)	(24/24)	(9/10)	(7/10)	(14/14)	(24/24)	(6/10)	(6/10)
LPval0							
5534.71	12375.9	325630	133803	5424.37	12240.7	326081	133172
LP							
5379.12	12184.3	324872	132419	5270	12089.3	324947	131655
5418.32	12222.6	324979	132546	5291.68	12140.7	325151	131902
5379.12	12171.7	324442	131812	5270	12089.2	324934	131265
IP							
5379	12168.1	324213	131570	5270	12089.2	324209	131061
5372.36	12133.9	323988	131425	5267.14	12074	323903	130514
5379	12171.7	324300	131244	5270	12089.2	324335	131051
+rnd100		324303	131592			324541	**131118.7**
+rnd10		**324342**	131592			**324547**	**131118.7**
HESGA		323708.2	130875.9			323630.2	130552.8
M1		323261.6	**131609.4**			323671.6	131112.8
M2		321945.2	131577.1			321843.4	**131118.7**
tIP							
6.69	2.17	6.66	59.98	0.11	0.48	9.84	92.67
9.57	1.05	42.63	0.10	16.77	0.34	15.70	0.07
1.38	0.41	6.71	11.41	0.41	1.11	66.76	41.48
nodes							
2434.43	4590.83	13962.80	15483.40	1101.14	3072.58	13394.20	18691.80
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
473.71	234.50	2112.30	2862.60	151.14	261.67	3826.80	5565.40
IPcol							
496.57	1060.62	4193.90	5859.40	392.64	581.46	4005.50	4117.50
55.29	47.96	84.20	90.80	56.21	47.29	86.00	90.60
131.14	86.25	702.10	961.90	88.86	117.38	1680.40	1079.20
MIP							
3	8	4	3	4	9	1	2
4	10	4	3	5	10	1	3
0	0	0	0	0	0	0	0

comparing them. These vectors are taken as the lines of the simplex basis inverse A_B^{-1}. As CPLEX does not provide that information, we retrieved line i of A_B^{-1} as the vector of simplex multipliers for the linear problem $\max\{e^i x_B : A_B x_B = b - A_N x_N,\ x_B \in \mathbb{R}_+^m\}$, where e^i is the i-th unit vector and the non-basic variables x_N are fixed.

To embed CPLEX in an application, there are two possibilities: a high-level C++ class library Concert Technology, where inequalities are stated in a readable form using overloaded operators, and a C-based Callable Library. In Concert we discovered a bug which did not allow the adding of columns after deleting a cut. ILOG reported that they had removed the bug in CPLEX 8.1. In ABACUS there is a large set of its own *container* classes, implementing lists, arrays, balanced trees, etc. ABACUS has been (re)designed since the end of the 80's. Now most programming languages have standard container libraries, e.g., the Standard Template Library in C++.

Each column, generated at some node, is valid for the whole problem. Thus, we kept a common LP formulation for all nodes. In the early versions this was helpful to find the following error: sometimes, a child node produced a weaker LP bound than its parent because of columns generated in another branch. Experiments with 'local columns', where each node knows only the optimum LP basis of its parent, led to a larger total number of columns. In contrast, local cuts were stored at a node locally. To simplify the search for violated cuts, each cut stored coefficients for all columns in the LP. To avoid creation of similar cuts or columns, they were stored in a balanced tree allowing quick search (for cuts, a tree of pointers to all cuts valid at a node has to be filled). For many instances, the program has different performance depending on the algorithm settings. When producing a 'black box' for external use, one should consider parameter variation and parallelization (algorithm portfolios).

2.8 Conclusions

A branch-and-price scheme for the one-dimensional cutting stock problem was enhanced by general-purpose cutting planes and also applied to the two-dimensional two-stage constrained cutting problem. Features and strategies of branch-and-cut(-and-price) were discussed. For 1D-CSP, the clear winner is depth-first search because non-IRUP instances are rare; best-first search is the winner for 2D-2CP. It is difficult to apply pseudo-costs in variable selection with column generation because many new variables arise. Comparisons with quite different algorithms for 2D-2CP show significant advantages on large-scale problem classes, especially on the unweighted instances which represent pure trim loss minimization. In 2D-2CP, very good or optimum solutions are found very easily;

the optimality proof is the bottleneck.

The optimality proof for 2D-2CP, similar to that for 1D-MCSP, is very difficult because possible objective values belong to a set of integer combinations of piece prices. This makes useful every means to strengthen the bound; thus, a combination of cuts and branching is more effective than each approach alone. Local cuts are especially important. In 2D-2CP, as in 1D-MCSP [BS02], cuts of high rank arise very often.

In 1D-CSP, most instances are IRUP, i.e., the root LP relaxation already gives the correct lower bound and pure branching is faster in finding the optimum integer solution. Only in some non-IRUP instances, cuts are necessary to prove optimality in an acceptable time. But in most instances no branching is required then if cuts are applied intensively at the root node.

Surprisingly, the pure BP approach is able to prove optimality for 4 of the 5 non-IRUP instances in the hard28 set, and it requires a comparable number of nodes to close the gap as the BCP algorithm. Moreover, the version where cuts were added only after the initial 200 nodes had been processed led to a systematic decrease of the total number of nodes only in non-IRUP instances; in IRUP instances, the effect was not clear. One would at least expect and demand that a BCP algorithm requires less nodes, since this is the principal aim of adding cutting planes. Let us remark that an optimum in non-IRUP instances is found in all known cases already at the root node, but the pure branch-and-price approach cannot close the gap and prove optimality, which is obviously a weaker deficiency than the other way around.

Another striking conclusion is that even an unbalanced branching scheme is often able to close the gap in the case of non-IRUP instances. A preliminary branch-and-price implementation with more balanced branching rules based on the arc flow formulation of 1D-CSP [AdC03] shows only a slightly worse performance. It seems to have no difficulties on non-IRUP instances. For 2D-2CP, a more balanced scheme would be branching on slacks.

The general-purpose cuts considered in this work make the pricing problem extremely complicated and much implementation effort is required. The question arises, whether any other classes of cuts, which do not complicate the pricing problem, e.g., cover inequalities in 2D-2CP, can provide a comparable contribution.

Chapter 3

Number of Setups and Open Stacks in One-Dimensional Cutting

In this chapter we are concerned with auxiliary costs and constraints arising in industrial applications. Most of the results can be straightforwardly applied in two-dimensional two-stage strip packing, see Chapter 1.

In Section 3.1 we compare two models of Vanderbeck for setup minimization and investigate a new simple model. In Section 3.2 we improve a sequential heuristic and modify it to restrict the number of open stacks. In Section 3.3 a combined approach to reduce both setups and open stacks is presented. In Section 3.4 we investigate problem splitting to further reduce the number of open stacks in the combined approach. In Section 3.5 we propose some IP models for open stacks minimization and show how additional variables can strengthen their relaxations.

3.1 Minimizing the Number of Different Patterns

The primary objective in cutting and packing problems is trim loss or material input minimization (in stock cutting) or value maximization (when packing into a knapsack). However, in real-life production we usually have many other objectives (costs) and constraints. Probably the most complex auxiliary criteria of a solution are the number of different cutting patterns (setups) and the maximum number of open stacks during the cutting process. For each new pattern we need time to set up cutting equipment and to perform test runs. We propose a new simple model for setup minimization (in fact, an extension of the Gilmore-Gomory model for trim loss minimization) and test it on problems with industrially relevant sizes of up to 150 product types. The behavior is investigated on a broad range of problem classes and

significant differences between instances of a class are found. Allowing even 0.2% more material input than the minimum significantly improves the results, this tradeoff has not been investigated in the earlier literature. Comparison to a state-of-the art heuristic KOMBI shows mostly better results; to a previous exact approach of Vanderbeck, slightly worse solutions and much worse LP bound, which is a consequence of the simplicity of the model.

3.1.1 Introduction

Most of the research on cutting and packing deals with material input minimization. In industrial cutting operations, material input is a very important criterion. Furthermore, the number of cutting patterns contained in a solution may be crucial for the cutting equipment, since switching between different patterns often necessitates time-consuming and sometimes error-prone setups, especially if this is done manually.

After giving a short overview of some known approaches, we review two models proposed by Vanderbeck [Van00a]. The first model captures pattern multiplicities in the constraint columns, i.e., for each feasible pattern there are separate variables for each multiplicity. On its basis he developed an exact algorithm by branching on hyperplanes (i.e., constraints involving groups of variables). Our model can be seen as a simplification of his second model becoming suitable for practical computation after eliminating the huge number of constraints. Although the model becomes non-linear, it can be linearly relaxed. The number of variables is smaller than in the first model of Vanderbeck because each pattern is represented by a single variable like in the Gilmore-Gomory model for material input minimization, which can be stated as

$$z^{\text{1D-CSP}} = \min\{\textstyle\sum_{j=1}^{n} x_j : \; Ax \geq b,\; x \in \mathbb{Z}_+^n\} \tag{3.1}$$

(Chapter 1), where A is the matrix of patterns. We propose to apply branching on single variables and show that this strengthens the relaxation. Then we develop an exact approach.

3.1.2 Some Known Approaches

- **Sequential Heuristic Procedure** of Haessler [Hae75]. A cutting plan is constructed sequentially choosing such patterns that can be applied with a high frequency. Increase of material input is possible.
- **KOMBI (post-optimization)** of Foerster and Waescher [FW00] generalizes some previous combination methods. It combines 2 to 1, 3 to $\{2,1\}$, 4

to $\{3,2,1\}$ patterns. It keeps the material input constant; nevertheless, better results than those of Haessler are obtained on most classes.

- **Exact approach branch&cut&price** of Vanderbeck [Van00a]. Capturing of pattern multiplicities in the columns. Simple cutting planes. Branching on hyperplanes.

- **Reduction of product amount deviation under a fixed number of different patterns** by Umetani, Yagiura, Ibaraki [UYI03]. The problem specifics comes from chemical industry. The main scheme is a local search metaheuristic: a neighboring solution is obtained by exchanging a pattern. Then pattern frequencies are computed by the non-linear Gauss-Seidel method so that to minimize the sum of squares of produced amount deviations. Trim loss cannot be controlled directly but the effort is made to choose 'good' patterns.

3.1.3 Column Generation Models of Vanderbeck

Vanderbeck [Van00a] investigates the following model. Let a^j $(j = 1, \ldots, n)$ be the set of feasible cutting patterns. Let

$$u(a) = \min_{i:\, a_i > 0} \{\lfloor b_i / a_i \rfloor\} \tag{3.2}$$

denote the implicit upper bound on the multiplicity x of pattern a in an integer solution. Let λ_{jk} $(k = 1, \ldots, u(a^j))$ be boolean variables meaning that pattern j is applied k times in the solution. The model is as follows:

$$\begin{array}{lrcll} & Z_{\mathrm{M}} & = & \min \sum_j \sum_{k=1}^{u(a^j)} \lambda_{jk} & \\ \text{[M]} \quad \text{s.t.} & & & & \\ & \sum_j \sum_{k=1}^{u(a^j)} k a_{ij} \lambda_{jk} & = & b_i, & i = 1, \ldots, m \\ & \sum_j \sum_{k=1}^{u(a^j)} k \lambda_{jk} & \leq & z^{\text{1D-CSP}} & \\ & \lambda_{jk} & \in & \{0,1\}, \qquad \forall j,\ k = 1, \ldots, u(a^j). & \end{array}$$

Additionally, the author describes the following model: let x_j $(j = 1, \ldots, n)$ be pattern frequencies like in 1D-CSP (3.1) and $\lambda_j = 1$ if pattern j is used in the

solution, otherwise $\lambda_j = 0$.

$$
\begin{array}{lrcll}
 & Z_{\text{M'}} & = & \min \sum_j \lambda_j & \\
\text{[M']} \quad \text{s.t.} & & & & \\
 & \sum_j a_{ij} x_j & = & b_i, \qquad & i = 1, \ldots, m \\
 & \sum_j x_j & \leq & z^{\text{1D-CSP}} & \\
 & x_j & \leq & z^{\text{1D-CSP}} \lambda_j, \qquad & \forall j \qquad (3.3) \\
 & x_j & \in & \mathbb{Z}_+, \qquad & \forall j \\
 & \lambda_j & \in & \{0,1\}, \qquad & \forall j.
\end{array}
$$

First of all, this model is not suitable for practical computation, it has too many constraints. Second, the author argues that the relaxation of this model is very weak: its optimum value is 1. However, this leak can be easily repaired by the following tightening:

$$x_j \leq u(a^j)\lambda_j, \qquad \forall j. \qquad (3.3')$$

Model [M'] strengthened by (3.3') will be denoted by [M"].

To investigate the strength of continuous relaxations of [M] and [M"], let us denote the relaxations by $\overline{M}$ and $\overline{M}''$, respectively.

Lemma 1 *If λ is a basic solution of $\overline{M}$ then for each $j = 1, \ldots, n$ it holds: $|\{k : \lambda_{jk} > 0\}| \leq 1$, i.e., for each group of variables representing the same pattern, only one is basic.*

Proof Columns of the constraint matrix corresponding to a group differ by a factor, i.e., they are linearly dependent. ■

What is called 'LP cheating' in [Van00a], i.e., taking of columns with high pattern multiplicities, can be stated as

Lemma 2 *Let λ be an optimum of $\overline{M}$. Consider some $j \in \{j : \exists k : \lambda_{jk} > 0\}$. Then $\lambda_{j,u(a^j)} > 0$ and $\lambda_{jk} = 0$ for $k < u(a^j)$.*

Proof Suppose $\lambda_{jk} > 0$ for some $k < u(a^j)$. Then by setting $\lambda_{j,u(a^j)} = k\lambda_{jk}/u(a^j)$ and $\lambda_{jk} = 0$ we obtain a feasible solution (Lemma 1) with a smaller objective value. ■

The following Lemma deals with $\overline{M}''$.

Lemma 3 *If (λ'', x'') is an optimum of $\overline{M}''$ then $x''_j = u(a^j)\lambda''_j \; \forall j$.*

Proof According to (3.3'), $x''_j \leq u(a^j)\lambda''_j \; \forall j$. Suppose $x''_j < u(a^j)\lambda''_j$ for some j. The setting of $\lambda''_j = x''_j/u(a^j)$ leaves the solution feasible and decreases the objective. ■

Proposition 9 *Continuous relaxations $\overline{M}$ and $\overline{M}''$ have equivalent sets of optimum solutions.*

Proof Let us define mappings of optima of $\overline{M}$ and $\overline{M}''$ onto the feasible solution set of the corresponding counterpart.
Conversion 1 Given an optimum of $\overline{M}$ λ, it satisfies $(\lambda_{jk} > 0 \Rightarrow k = u(a^j))\ \forall j$ (Lemma 2). Thus, solution (λ'', x'') of $\overline{M}''$ with $\lambda''_j = \lambda_{j,u(a^j)}$ and $x''_j = u(a^j)\lambda''_j$ $\forall j$ represents the same set of cutting operations.
Conversion 2 Given an optimum of $\overline{M}''$ (λ'', x''), it satisfies $x''_j = u(a^j)\lambda''_j\ \forall j$ (Lemma 3). Thus, solution λ of $\overline{M}$ with $\lambda_{j,u(a^j)} = \lambda''_j$ and $\lambda_{jk} = 0$ $(k < u(a^j))\ \forall j$ represents the same set of cutting operations.

Let us show that these conversions produce optima in the corresponding target model. Suppose, (λ'', x'') obtained by Conversion 1 is not an optimum, then there exist $(\lambda''^{(1)}, x''^{(1)})$ with $\mathbf{1}\lambda''^{(1)} < \mathbf{1}\lambda''$. Applying Conversion 2 to $(\lambda''^{(1)}, x''^{(1)})$, we get $\lambda^{(1)}$ with $\mathbf{1}\lambda^{(1)} < \mathbf{1}\lambda$ which is a contradiction to the optimality of λ. The opposite direction is similar. ■

Vanderbeck uses model [M]. He branches on hyperplanes involving many λ-variables.

3.1.4 Modeling

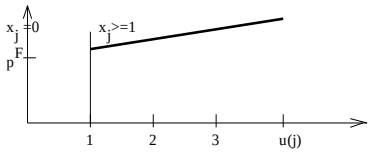
Our model can be seen as a simplification of [M"] becoming suitable for practical computation after eliminating the huge number of constraints (3.3') and variables λ. Although the model becomes non-linear, it can be linearly relaxed. The number of variables is smaller than in [M] because each pattern is represented by a single variable like in the Gilmore-Gomory model.

When minimizing the number of setups, we allow a maximum

$$K = z^{\text{1D-CSP}} + \Delta K \tag{3.4}$$

total number of stock sheets where ΔK is some tolerance. In the model of Vanderbeck ΔK was not considered. Usually [Sch02b] the customers prefer not to increase material input because its price is high. However, the final decision concerning this tradeoff must be left to the user. Let a^j $(j = 1, \ldots, n)$ be the set of proper cutting patterns for 1D-CSP and x_j their frequencies. If the user allows $\Delta K > 0$ then we may consider not only setup costs (pure setup minimization), but also combine them with material costs. Let us consider material costs p^V per stock sheet and pattern setup costs p^F. Then the costs $\delta(x)$ of using the j-th pattern x times in the solution equal

$$\delta(x) = \begin{cases} 0, & x < 1 \\ p^F + p^V x, & x \geq 1. \end{cases} \quad (3.5)$$



The **combined trim loss/pattern minimization problem** can be stated as follows:

$$\text{[PMP]} \qquad \begin{aligned} & \min \textstyle\sum_{j=1}^{n} \delta(x_j) && (3.6) \\ \text{s.t.} \quad & \textstyle\sum_{j=1}^{n} a_{ij} x_j = b_i, \quad \forall i && (3.7) \\ & \textstyle\sum_{j=1}^{n} x_j \leq K && (3.8) \\ & x_j \in \mathbb{Z}_+, \quad \forall j. && (3.9) \end{aligned}$$

In contrast to trim loss minimization (3.1), we choose constraints $Ax = b$ (3.7). Allowing overproduction, for example like this:

$$b_i \leq \textstyle\sum_{j=1}^{n} a_{ij} x_j \leq b_i + \Delta b_i, \quad \forall i$$

is certainly very favorable for the problem because this flexibility often reduces the minimum number of patterns. Actually, this is very often done in practice [Sch02b] and shows large advantages, e.g., in paper industry [ZB03] and chemical industry [UYI03]. However, it is difficult to systematically estimate costs of overproduction or to obtain real-life data and thus we postpone this extension to future research.

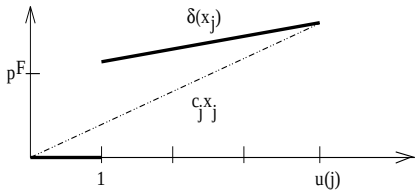
The optimality of a feasible solution is proved when it can be shown that no better objective value is possible, e.g., using some lower bound. Objective values of (3.6) belong to the set of linear integer combinations of p^F and p^V. To avoid the construction of the whole set, we work implicitly with its extension representing all multiples of the greatest common divisor $\gcd(p^F, p^V)$. The smaller its value, the more difficult the optimality test.

3.1.5 Lower Bound

We can linearize the contribution (3.5) of variables x_j to the objective (3.6) as follows. Let us define

$$c_j = p^V + \tfrac{p^F}{u(a^j)} \quad (3.10)$$

so that $c_j x_j$ is a linear approximation of $\delta(x_j)$ on $[0, u(a^j)]$ (see picture).



The bounds $x_j \leq u(a^j)$, $\forall j$ can be explicitly added to tighten the formulation of

the following linear continuous relaxation of [PMP], called the *master problem*:

$$\underline{z}^{\text{PMP}} = \min\{\textstyle\sum_{j=1}^{n} c_j x_j : Ax = b,\ \mathbf{1}x \le K,\ x_j \in [0, u(a^j)]\ \forall j\} \tag{3.11}$$

with the column generation (*slave*) subproblem

$$z^{\text{CG}} = \min\{p^V + p^F/u(a) - \textstyle\sum_{i=1}^{m} d_i a_i - d_0 : \\ la \le L,\ a \le b,\ a \in \mathbb{Z}_+^m\}, \tag{3.12}$$

where $d_1, \ldots, d_m$ are the dual multipliers of (3.7) and d_0 that of (3.8).

Proposition 10 *In the special case when $\Delta K = 0$, $p^F = 1$, and $p^V = 0$, LP* (3.11) *is equivalent to the continuous relaxations of the models [M] and [M"].*

Proof A consequence of Lemma 3 is that in the relaxation $\overline{M}''$, the variables λ can be eliminated provided that $x_j \le u(a^j)$ for all j. Which yields (3.11). ■

Thus, what is called 'LP cheating' in [Van00a], see Lemma 2, is the linear approximation of the objective in our model. This approximation is very weak. We cannot expect it to produce a strong bound or to guide our search toward good solutions. Some means to strengthen it are necessary.

Cutting planes can not significantly strengthen the relaxation because we relax the objective. In contrast, cuts can be effective in model [M].

3.1.6 Column Generation

A consequence of the equivalence of the LP relaxations of [M] and [PMP] is the identity of the column generation (pricing) procedures. Pricing is the optimality test of the LP. If $z^{\text{CG}} = 0$ after solving (3.12), then the current subset of columns (restricted formulation) contains an LP optimum, otherwise the corresponding column is added.

An upper bound on the maximum multiplicity of any cutting pattern is given by

$$q_0^{\max} = \min\{K - \underline{D} + 1, \max\{b_i\}\},$$

where $\underline{D}$ is a lower bound on the number of different patterns, e.g., the optimum of the bin-packing problem (1D-BPP) produced by setting $b_i = 1\ \forall i$. A brute force approach for (3.12) would be to enumerate on all $q_0 = 1, \ldots, q_0^{\max}$ and, at each iteration, to solve the bounded integer knapsack problem

$$\max\{\textstyle\sum_{i=1}^{m} d_i a_i : \sum_{i=1}^{m} l_i a_i \le L,\ a_i \le u_i(q_0)\,\forall i,\ a \in \mathbb{Z}_+^m\}, \tag{3.13}$$

where the piece upper bounds are

$$u_i(q_0) = \lfloor b_i/q_0 \rfloor. \tag{3.14}$$

Algorithm 3.1.1 Pricing for setup minimization

Set $q_0 = 1$.
Set $c = 0$ as the initial upper bound on (3.12).
While ($q_0 \leq q_0^{\max}$)
  Let $\underline{v} = -c - d_0 + p^F/q_0^{\max} + p^V$ be a lower bound on (3.13).
  Solve the bounded knapsack (3.13) with the initial incumbent $\underline{v}$.
  If no better solution exists, goto Exit.
  Let v^* and a^* be the optimum value and solution of (3.13).
  Set $q_0 = u(a^*)$.
  If $p^V + p^F/q_0 - v^* - d_0 < c$ then
    Set $c = p^V + p^F/q_0 - v^* - d_0$;
    Record the better column a^*.
  Set $q_0 = q_0 + 1$.
End While
Exit: **END**

But in fact only a subset of multiplicity values q_0 needs to be considered. A solution $a^* \in \mathbb{Z}_+^m$ of (3.13) for a given q_0 remains optimum for all q_0 values up to the multiplicity $u(a^*)$. Therefore, the next q_0 that must be considered is $q_0 = u(a^*)+1$. The procedure to solve the pricing problem is given in Algorithm 3.1.1.

For the LP (3.11), adding cutting planes can affect only the integrality of solutions but the approximation of the objective remains weak.* In [Van00a] simple cuts are used which are based on only a single previous constraint and they have proved very effective: at the root node, LP bound could be improved by 20%.

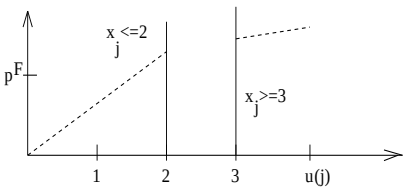
3.1.7 Branching

Relaxation (3.11) is very weak because its objective function is an approximation. To tighten the approximation, *branching on variables* can be employed. It means that we pick up a variable x_j and separate the problem in two subproblems: the one with $x_j \leq \mathcal{U}_j$ (*left son*) and the other with $x_j \geq \mathcal{L}_j$ (*right son*), where $\mathcal{U}_j + 1 = \mathcal{L}_j$. For an x_j to be branched on, it need not have a fractional value in the LP solution.

*However, even cuts which involve several original constraints for construction, such as CHVÁTAL-GOMORY cuts ([NW88] and Chapter 2), would still allow the skipping of some q_0 values in the pricing procedure, similar to the above. In model [M] the situation is the opposite: the capturing of pattern multiplicities in the columns extends the effectiveness of cuts but cuts involving several original constraints for construction would necessitate the enumeration of all q_0 values because the cut coefficients of a column may be different for each q_0. Moreover, the knapsack problem (3.13) would become non-linear in both models.

The reason is that an integral LP solution is not automatically an optimum like it is for the relaxation of 1D-CSP (3.1). Given an integral solution $\overline{x}$, it is optimum if its objective value $\sum_j \delta(\overline{x}_j)$ is equal to some lower bound. This means that we can also branch on non-integer-valued variables to improve their cost approximation. Consider an example of branching.

In the left son we can improve the approximation of the objective function by setting $c_j = p^V + p^F/\mathcal{U}_j$. In the right son we can already apply the original costs $\delta(x_j)$: we increase $\underline{z}^{\text{PMP}}$ by p^F and set $c_j = p^V$.



Note that the parent LP solution is not such a good starting point for the son's LP as in usual MIP problems because of the change of the objective function. Moreover, right branches tighten the objective coefficients of other variables because of the reduction of the right-hand side: $b' = b - A\mathcal{L}$ which may reduce $u(a^j)$ for some j and change coefficients (3.10). This tightening effect is absent when branching on hyperplanes.

The question is how to choose the branching variable x_j and the branching value, i.e., $\mathcal{U}_j$ with $\mathcal{L}_j$. A further question is, which subproblem is selected to be processed next among the list of open subproblems (for a general scheme of a branch-and-price algorithm, see, e.g., [JT00, LS99, Van00a]). Here we can approach two main strategies: either we try to improve the global lower bound which is the minimum of the local LP bounds of all open subproblems; or we try to improve the upper bound represented by the best feasible solution found so far. In the first variant we always select the subproblem promising the best solution, i.e., whose local LP bound is the worst (weakest); hence the name *best-first search* or *bfs*. In the second variant we always select the deepest subproblem in the tree; hence the name *depth-first search* or *dfs*. Sometimes it is proposed to investigate several different branches in parallel during dfs, see, e.g., [cpl01]/ Constraint Programming, which helps to avoid spending too much time investigating a branch chosen wrongly at a high level. In [LS99] it is proposed to combine bfs and dfs to get the best of both strategies and to avoid exploring wrong branches.

We found the following combination of bfs and dfs best. The first 10 nodes are chosen according to bfs; after that, m nodes are chosen in the dfs fashion but 'diving' from the last bfs node. Only the subtree of that node is investigated. Right branches are preferred ($x_j \geq \ldots$); branching variables are chosen according to the largest value. Vanderbeck does actually the same but in a rounding procedure; thus, the subproblems created during rounding can be created again. He tries to begin diving each time from a new variable.

Also, the situation is possible when all basic variables have lower bounds greater than 0 (or those with 0 lower bounds have values at their upper bounds). In

this case we proceed as follows. If the values are integer, then the LP solution is an optimum for the subproblem because in the LP we have the real objective value. Otherwise we branch on the most fractional variable as in standard IP methods. But as long as some variables with zero lower bounds are present in the basis, we choose only among them for branching, in order to improve the approximation of the objective. Thus, we have a two-level branching scheme.

There exist further variable selection strategies, e.g., pseudo-costs which consider the experience of how branching on a given variable affects the local LP bound of the sons. However, in branch-and-price new variables are generated dynamically, which destroys the comparing effect of pseudo-costs (Chapter 2).

3.1.8 Feasible Solutions

We rely on the LP and its strengthening by branching on variables to guide the search process toward good solutions. To explore the feasible solutions in the neighborhood of an LP solution, we try several integer vectors $[x]$ derived from the LP solution $\overline{x}$ that differ by some components rounded up or down, see Algorithm 1.4.1. Then $b' = b - A[x]$ forms a *residual problem* which is solved by the heuristic presented in the next section. This is done at every 10th node of the branch-and-price tree and after adding 30 new columns to the formulation.

3.1.9 Implementation

In [Van00a] an artificial column equal to the right-hand side for '=' constraints, with a large objective coefficient, was used to initialize the LP. Setting a previous 1D-CSP solution as a starting basis produced worse results. We observed the opposite on average. Similarly, in our LP model 'infeasible' slacks e^i $(i = 1, \ldots, m)$ with large objective coefficients ($\approx 10^5$) are always present because both left and right branches can have infeasible restricted LP formulations[†] and further columns may be needed to make them feasible. If we finally find a feasible solution for the LP (which is not guaranteed), it does not mean that the node is feasible: there may exist no integer solutions. If no improving column can be found and an artificial column is still in the basis with a value above 10^{-6}, the node's LP is considered infeasible. All columns of the LP are stored in a common pool. Allowing each node's LP to start from a local pool containing only the basis of the parent's LP led to a higher number of different columns.

[†] left branches can have an infeasible restricted LP already in 1D-CSP (Chapter 2); in PMP, also right branches, because of the trim loss constraint.

3.1.10 Computational Results

In the experiment we benchmarked the behavior of the method on a spectrum of problems, investigated the tradeoff with material input, compared the approach against the heuristic KOMBI and the approach of Vanderbeck. Tests were run on a 1000 MHz Athlon XP with 512 MB memory (though the maximum requirement was about 10 MB) under Linux. The algorithm was implemented in GNU C++ 2.96. The LPs were solved by the ILOG CPLEX 7.5 Callable Library [cpl01].

We considered classes of problems (m, L, l, b) with parameters $L = 10000$; $m \in \{20, 50, 150\}$; l_i uniformly distributed in $[v_1 L, v_2 L) \cap \mathbb{Z}$ for all i, where $v = (v_1, v_2) \in \{(0.01, 0.2), (0.01, 0.4), (0.01, 0.7), (0.2, 0.4), (0.2, 0.7)\}$; b_i uniformly distributed in S_j for all i, where $S_1 = [1, 10) \cap \mathbb{Z}$, $S_2 = [1, 100) \cap \mathbb{Z}$, $S_3 = [50, 100) \cap \mathbb{Z}$. The reasons behind our choice are that $m = 150$ is the maximum number of product types in nearly all applications [Sch02b]; $v_1 = 0.01$ is nearly always the minimum product size ratio. The basic class was ($m = 50$, $v = (0.01, 0.07) = v^3$, $b_i \in S_2$).[‡] By varying the parameters of the basic class, we obtain the 9 test classes ($m = 20$,,), (,v^1,), ..., (,v^5,), (,,S_1), (,,S_3), ($m = 150$,,) shown in Table 3.1. The reason to consider this new set is that other researchers took problem sizes $m \leq 40$ [FW00]. Moreover, by always varying only one of the parameters we avoid an explosion of the number of test classes. In comparisons with other authors we set $\Delta K = 0$ (3.4), thus the problem is pure setup minimization and p^V with p^F (3.5) can be in any relation (if $p^F > 0$). However, $p^V = 0$ and $p^F = 1$ is numerically the best choice.

3.1.10.1 Benchmark Results

The behavior of the method on the chosen spectrum of problem classes is shown in Table 3.1. The time limit was $t = m$ seconds per instance (increasing this limit did not lead to notable improvements on average). 20 instances were calculated for each class. As each product length was generated uniformly in $[v_1 L, v_2 L) \cap \mathbb{Z}$, some lengths were equal and these product types were merged. Thus, the average number of product types is smaller, see m^{ave}. The following results are given as average values: the best LP lower bound LB; the best solution UB^{ave}; the times to obtain them t LB and t UB; the number of patterns in the 1D-CSP solution obtained by branch-and-price (Chapter 2) UB^0; the initial LP bound before branching LB^0; the 1D-CSP optimum $z^{\text{1D-CSP}}$; the number of branch-and-price nodes visited during the time limit 'nod'; the number of columns in the root LP 'col^0' and in all nodes 'col'. Results given as absolute values: min and max best solution $\text{UB}^{\min}$ and $\text{UB}^{\max}$. It can be seen that larger right-hand side b and larger item sizes lead to more different patterns.

[‡] See also 1D-CSP benchmark results in Chapter 2.

Table 3.1: Pure setup minimization: benchmark results

Class	1	2	3	4	5	6	7	8	9
m	20	50	50	50	50	50	50	50	150
v	v^3	v^1	v^2	v^3	v^4	v^5	v^3	v^3	v^3
S	S_2	S_2	S_2	S_2	S_2	S_2	S_1	S_3	S_2
m^{ave}	19.95	49.85	49.85	49.85	49.85	49.85	49.85	49.85	148.3
$z^{\text{1D-CSP}}$	349	258	502	878	743	1146	88	1321	2657
LB^0	9.5	6.88	12.2	23.4	19.7	26.9	20	19.9	67.6
UB^0	21	55	51.5	51	51.7	51.25	41.9	52	150.5
LB	10.9	6.92	12.4	24.7	21.2	27.7	20.9	21.2	68.5
UB^{ave}	15.75	20.9	29.5	41.05	38.6	42.6	24.9	43	123.2
$\text{UB}^{\min}$	12	17	25	36	35	40	21	37	106
$\text{UB}^{\max}$	18	25	34	46	47	46	30	49	147
t LB	4.1	1.77	2.33	15.1	29.9	11	9.4	17	23
t UB	3.23	16	36.8	17.1	20.9	24	14.5	26	52
nod	2869	72	208	1980	2591	6547	2515	2850	1521
col^0	70	469	281	171	129	112	159	113	530
col	913	4298	4380	2252	1958	1400	2810	2427	3351

Table 3.2: Setup minimization: tradeoff with material input
Entries: $\text{LB:UB}^{\text{ave}}\text{:UB}^{\max}$

Class	1	4	9
$\Delta K = 0$	10.90:15.90:18	24.64:41.1:48	68.50:123.2:147
$\Delta K = \lceil 0.001 z^{\text{1D-CSP}} \rceil$	10.30:15.40:17	22.46:38.30:42	58.84:106.4:118
$\Delta K = \lceil 0.002 z^{\text{1D-CSP}} \rceil$	10.30:15.40:17	21.77:36.75:39	57.54:103.85:111
$\Delta K = \lceil 0.01 z^{\text{1D-CSP}} \rceil$	9.82:15.15:18	20.61:35.30:38	56.41:97.50:107
$\Delta K = \lceil 0.05 z^{\text{1D-CSP}} \rceil$	9.28:14.5:16	20.21:34.15:37	56.37:95.30:104

In 5 of the 20 instances with $m = 150$ almost no improvement was achieved: for them, the best solution was not smaller than 140, see $\text{UB}^{\max}$. Experimentation with algorithm parameters did not give better results. One attribute of these instances is that in the time of 150 seconds, the solution process investigates 4000 nodes and more, while in better instances it is 10 times less, 200–300 nodes. Another attribute is that far fewer columns are generated: in the root LP only 400 (in much shorter time) while 600 in other instances, and overall 1300 while otherwise 3500–5000. The maximum node depth in the solution tree is much smaller.[§] The explanation may be that these instances have only a few solutions with the minimum material input, the search has no freedom, that is why so few patterns are generated. In practice we may cope with difficult instances by allowing larger material input or flexible order demands.

3.1.10.2 Tradeoff with Material Input

Table 3.2 contains, for each class, LB:UB^{ave}:$\text{UB}^{\max}$ for different ΔK. Allowing even $\Delta K = \lceil 0.2\%(z^{\text{1D-CSP}}) \rceil$ more material input in solutions is very favorable for our approach making the task easier. The reasons are that the minimum number of different patterns decreases and that the freedom to obtain 'loose' solutions helps to find good solutions quickly.

3.1.10.3 Comparison with KOMBI234

Our approach of branching on variables (BrOnVar) is compared to the state-of-the-art heuristic KOMBI234 of Foerster/Waescher [FW00]. The test set is organized as follows. Material size is $L = 1000$; integer product sizes are chosen as $v_1 L \leq l_i \leq v_2 L \ \forall i$ with $(v_1, v_2) \in \{(0.01, 0.2), (0.01, 0.8), (0.2, 0.8)\}$; the number of product types $m \in \{10, 40\}$; the average demand $\overline{b} \in \{10, 100\}$, which yields 12 combinations. Each combination represents a problem class and is denoted by a 3-digit code, e.g., 111 means $(v_1, v_2) = (0.01, 0.2)$, $m = 10$, and $\overline{b} = 10$. 100 instances were generated for each class by the generator CUTGEN [FW00] with the seed 1994 to provide the identity of test data.

Solution parameters: $\Delta K = 0$ (as in [FW00]), $p^V = 0$, $p^F = 1$, the maximum time 40 seconds per instance. Search strategy: 10 nodes bfs, m nodes diving dfs from the last bfs node. The average number of patterns obtained is shown in columns N of Table 3.3. The results are better for 11 classes out of 12. The advantage on class 111 for KOMBI can be explained by the low number of patterns: KOMBI is strong at combining 2, 3, and 4 patterns into a smaller number. For KOMBI, t gives the average total running time. For BrOnVar, t_{best} gives the

[§]When searching for reasons, we found that at depths of 40–60, nodes become infeasible: some artificial slacks have values of the order 10^{-4} and no new columns can repair the situation.

Table 3.3: Average number of patterns in comparison to KOMBI

	KOMBI234		BrOnVar	
CPU	66 MHz		1000 MHz	
Class	t	N	t_{best}	N
111	0.35	3.40	1.94	3.43 -
112	1.26	7.81	5.01	6.08
121	40.03	10.75	9.15	10.47
122	383.30	25.44	14.3	18.71
211	0.11	7.90	0.24	$\underline{7.57}^{\alpha}$
212	0.24	9.96	1.47	8.98
221	36.98	28.74	10.8	25.18
222	77.41	37.31	9.92	33.75
311	0.13	8.97	0.01	$\underline{8.79}^{\alpha}$
312	0.18	10.32	0.22	9.97
321	51.31	31.46	7.47	29.15
322	71.31	38.28	6.93	35.99

$^{\alpha}$ classes 211 and 311 were solved to optimality completely.

average time to find the best solution. Taking CPU frequency into attention, out approach needs more time.

3.1.10.4 Comparison with the Exact Approach of Vanderbeck

Vanderbeck uses bfs but his rounding heuristic mimics diving into the search tree, i.e., dfs. He employed an HP9000/712/80 MHz with 64MB memory allowing a time limit of 2 hours per instance. In Table 3.4 we show our results for the instances from [Van00a]. The time limit was 30 minutes per instance. The columns of the table are:
instance number,
name instance name,
m number of product types,
$\underline{\text{b}}$ the lower bound on the number of patterns, given by the LP bound of the corresponding 1D-BPP,
LB^0 the LP value at the root node,
LB the strongest global LP value/bound,
UB the best feasible solution,
K the 1D-CSP optimum,
nod the number of nodes,
col^0 and **col** the number of columns in the LP at the root node and total,

Table 3.4: Comparison to the exact method of Vanderbeck

					BrOnVar								Vanderbeck				
#	name	m	b	LB^0	LB	UB	K	nod	col^0	col	tLB	tUB	LB	UB	nod	tLB	tUB
1	kT03	7	4	4.77	6 =	6	66	47	25	47	0.03	0.03	6 =	6	91	4	28.1
2	kT05	10	5	5.65	9 =	9	47	1250	27	66	0.69	0.02	9 =	9	37	10.9	3.3
3	kT01	5	1	2.00	2.1 =	3	14	3	20	40	0	0.02	3 =	3	1	1	2.8
4	kT02	24	13	15.9	18 =	18	66	346	66	118	0.3	0.24	18 =	18	1	1.7	3.1
5	kT04	16	6	6.71	9 =	9	38	65661	50	419	400	0.94	9 =	9	57	35.8	3.6
6	d16p6	16	6	6.71	9 =	9	38	68532	50	417	426	0.61	9 =	9	39	29	5.1
7	7p18	7	2	3.73	6 =	6	91	269	27	134	0.31	0.11	6 <	7	1351	2105	11.5
8	d33p20	23	5	6.04	6.70 <	8	29	7085	134	6021	0.05	7.8	8 =	8	655	2051	9.1
9	12p19	12	2	2.88	3.49 <	5	23	7549	72	6199	0.06	73.8	5 =	5	47	141	10.5
10	d43p21	32	7	7.86	8.51 <	10	40	4073	190	7166	0.23	148	10 =	10	33	230	143
11	kT06	9	1	1.65	2.24 <	4	53	6836	53	6341	19.6	895	4 =	4	51	1796	66.2
12	kT07	11	2	2.86	3.27 <	5	68	4733	83	7656	0.6	1.13	5 =	5	163	6818	209
13	14p12	14	2	3.71	5 =	5	56	981	96	3172	97.8	86.2	5 =	5	1	29.8	68.3
14	kT09	14	2	3.54	4.01 <	6	115	3636	84	7879	1411	946	5 <	6	140	21.2	66.1
15	11p4	11	1	2.46	2.70 <	6	101	3225	90	8911	0.08	2.75	4 <	5	19	76.2	2801
16	30p0	26	4	5.50	5.81 <	11	90	1384	186	13522	0.14	107	7 <	8	13	37.5	3563

tLB the time of the best global LP bound given by LB,
tUB the time of the best solution UB.

While the solution values are practically the same,[¶] our lower bound is much weaker. This is the cost of the simplicity of the model.

3.1.11 Summary and Conclusions

We have tested an exact approach to pattern reduction. It is based on a model with a smaller number of variables than in the previous exact approach of Vanderbeck. We employed *branching on variables* to tighten the relaxation.

- LP bound cannot be improved significantly already for medium instances because of the high combinatorial complexity (a large number of variables). This is the price for the simplicity of the model whose relaxation cannot be tightened by cuts. The approach of Vanderbeck produced much better bound but still could not solve all the instances optimally.

- Considering the best found solutions, for some instances there is no improvement, e.g., 147 setups for a problem with $m = 150$ where it is easy to find a starting solution of, say, 155. The behavior of the search process lets us think that the solution space of such instances is small.

  To increase the solution space we propose to allow, say, 0.2% more material input than the minimum, which is probably acceptable in most industries. This significantly improves the results. Another alternative would be to allow under- and overproduction.

- The proposed scheme of branching on variables with some appropriate branching strategies mostly leads to better solutions than the heuristic KOMBI and only to slightly worse solutions than the previous exact approach of Vanderbeck.

3.2 Restricting the Number of Open Stacks

During cutting, for each started and not yet finished product type we need a stack (pallet) standing around and occupying place. There are applications where the number of stacks is restricted to two. But existing methods usually produce much worse results in terms of open stacks or the increase of material input is hardly acceptable. We design a sequential heuristic to minimize material input while restricting

[¶] For instance 13, the number of diving steps including backtracking was $1.5m$, for others $2m$

the number of open stacks to any given limit. Earlier versions of this heuristic which was used for pure material minimization based on the idea to penalize products which constitute bad patterns and to accumulate such information from several solutions. We propose a further effective means to eliminate bad item combinations, namely *overproportional item weights*. The approach is shown to be very powerful in minimizing material input, even for problems with a large right-hand side. Further we add the option to control the number of open stacks which increases material input only negligibly, enabling the industrial application of the method. Different solutions are evaluated in relation to the multiple objectives using the PARETO criterion.

3.2.1 Introduction

Usually the set of product types is separated into *lots*, i.e., groups [JRZ99]. Each lot is shipped to a separate customer in one or several trucks /pallets /stacks. During cutting of the sequence of patterns specified in a solution, each open stack corresponds to a lot whose products have been started and not finished. Moreover, if several stacks are needed for the whole lot, we may consider an option to interrupt the lot when the current stack for it is closed. This option will not be considered in the current work since we have no real-life data. However, in the Outlook we present ideas how this option could be incorporated in our approach and when this is useful. Separation of the product set into lots is a simplification of the problem and can be modeled easily. Moreover, in practice some lots have a higher priority [JRZ99], which further helps to determine which products should be in the beginning of the cutting sequence. We do not consider lots either because of the absence of real-life data. Thus, we assume that each product type needs a separate stack.

Each stack, waiting to be completed and shipped, occupies some place near the cutting machine (temporary removal of an uncompleted stack is certainly not acceptable). In some applications we are restricted to 15–20 stacks. In other applications the maximum equals 2 [Sch02b]. Concerning this criterion, most of the research has been done on sequencing patterns of a given solution [Yue91b, YBS99]. In [YBS99], for problems with $m = 150$ product types, lower bounds on the maximum number of open stacks are as high as 100 and solutions of about 130 are found. Thus, it is necessary to construct new patterns that lead to a better sequence. The only published approach of this kind is, to our knowledge, [JRZ99]. There the authors report on huge savings in the industry, resulting from their methods. They also illustrate that reducing the number of open stacks can increase the number of different patterns, which is undesirable. The average number of open stacks may also play a role [Yue91b]. To strictly control the num-

ber of open stacks, we may resort to *sequential* solution approaches (see below); however, they are reported to produce bad material utilization ratios [Yue91b].

Another criterion connected to the open stacks is *spread*. It is the physical duration of a stack's being open. We have no real-life data about this criterion. It would be realistic to suppose that a restriction on spread can be connected with the product size and demand. In our observations we saw a tight correlation between the spread and the number of open stacks.

To summarize, we give an overview of the problem structure, see also [DF92].

Costs in 1D cutting:

- Material costs, i.e., the total number of applications of patterns
- Setup costs, i.e., the number of different cutting patterns
- Average number of open stacks during the cutting sequence

Constraints in 1D cutting:

- Order demands of products
- Number of open stacks at any moment of time

Additional specific constraints, such as:

- The maximum number of knives, i.e., how many items can be in a pattern
- The maximum number of knives with the distance between them smaller or greater than a given size
- The minimum length of a pattern (corresponds to the maximum allowed waste)
- Layout constraints, e.g., that definite product types should be located in definite parts of a pattern
- Product combination constraints, e.g., which product types can be combined in a pattern

The specific constraints can be more or less straightforwardly incorporated into pattern generation and reduce the solution space. But the numbers of patterns and open stacks are more sophisticated parameters. Moreover, we are not aware of research aimed at combining pattern reduction with open stacks reduction. Note that, for 1D-BPP, the latter criteria are not relevant, so we consciously consider test problems with a larger right-hand side.

3.2.2 Some Known Approaches

- **Sequencing of patterns of a given solution**, [YBS99, Yue91b] and others. Given a solution, a permutation of its patterns is searched for, that minimizes the number of open stacks needed. In [YBS99], for problems with $m = 150$ product types, lower bounds on the maximum number of open stacks are as high as 100 and solutions of about 130 are found.

- **Trying to generate a solution** satisfying a given maximum of open stacks [JRZ99]. By performing pivot steps in the simplex method, many different solutions are generated. Each solution is sequenced by a heuristic. Finding a solution satisfying a certain maximum of open stacks is not guaranteed. There are no test results for large problems in the paper, thus we do not make any comparison. In the next section we generalize this approach by incorporating a sequencing heuristic into a pattern minimization approach.

- **Sequential approaches** are mentioned by Yuen [Yue91b], Haessler [Hae75], Mukhacheva [MZ93], Kupke [Kup98], Vahrenkamp [Vah96], and others, for different tasks. There a solution is constructed sequentially, pattern after pattern. In this way, the number of open stacks can be controlled and limited. However, Yuen argues that such approaches, though used in industry, produce solutions with a bad ratio of material utilization. Other authors, to our knowledge, have not investigated their approaches on large problems. There are no theoretical results either. Below we describe a heuristic of this kind and investigate its effectiveness on various problem classes. Actually, we used it in Chapter 2 to solve residual problems.

3.2.3 A Sequential Heuristic Approach

In [MZ93] a *sequential value correction* (**SVC**) heuristic is proposed for 1D-CSP. Like any sequential heuristic, it constructs a solution pattern after pattern and assigns a frequency to each one. However, SVC constructs several solutions iteratively and each pattern is generated using some information from previous solutions. This information, *pseudo-prices* of products, reflects the size of waste in the patterns containing a given product type. Each pattern is generated so as to obtain a large enough total price of items inside. To avoid accumulation of good patterns (with small items) in the beginning which leaves no possibilities for good combinations in the last patterns, the total price of a pattern should not be maximum. In [MZ93] it is proposed to set some minimum requirement ('barrier' or aspiration level) on the total price. Another heuristic of this kind, *Max-Len-Mini-Items* [Kup98], tries to minimize the waste of each next pattern but the number of items inside is kept small so that smaller items are not overused. We propose

Algorithm 3.2.1 Sequential value correction

Input: An instance (m, L, l, b') of 1D-CSP;
The simplex multipliers $d_1, \ldots, d_m$;
Output: A feasible solution vector $(\widetilde{x}_{a^j})_{j\in\{1,\ldots,n\}}$;

Initialize: $y_i = \max\{1, Ld_i\}$ for all $i = 1, \ldots, m$;
Iteration number $k = -1$;
Repeat *(iterate)*
$b'' = b'$; $\widetilde{x} = \mathbf{0}$; $k = k + 1$; 'start new solution
Repeat
$a = \arg\max\{ya : la \le L,\ a \le b'',\ a \in \mathbb{Z}^m_+\}$;
$\chi = \min_{i:\, a_i>0}\{\lfloor b''_i/a_i \rfloor\}$; 'choose pattern frequency
$\widetilde{x}_a = \widetilde{x}_a + \chi$; $b'' = b'' - \chi a$; 'reduce the right-hand side
Update Weights y_i, $1 \le i \le m$; ''value correction'
Until ($b''_i = 0$ for all $1 \le i \le m$); 'until the solution is complete
If $\widetilde{x}$ is an improvement, save it;
Until (Optimality is Proved) or (Iterations Limit is Exceeded);

---

another scheme to reduce bad combinations on the basis of SVC: the total price of each next pattern is maximized and product prices are *overproportional* to product lengths (a separate study could be useful to compare the three approaches). The rest of the scheme is identical to the previous version of SVC. The pseudo-prices $y \in \mathbb{R}^m$ are maintained as follows. They are initialized with scaled simplex multipliers. After generating pattern a maximizing ya, they are 'corrected': let $w = L - \sum_{i=1}^m l_i a_i$ be the pattern waste. The value $[l_i]^p L/(L - w)$ for $p > 1$ is called the *overproportional material consumption norm* of piece i in the generated pattern. The new weight of piece i is the following weighted average:

$$y_i \leftarrow g_1 y_i + g_2 \frac{L}{L - w}[l_i]^p, \quad \forall i :\ a_i > 0,$$

where g_1, g_2 are update weights with

$$g_1/g_2 = \Omega(b'_i + b''_i)/a_i, \tag{3.15}$$

$g_1 + g_2 = 1$; b'_i is the total order amount of product i; b''_i is the unused order quantity of product i; the value of the randomization factor Ω is uniformly chosen before each pattern from $[1/\overline{\Omega}, \overline{\Omega}]$; $\overline{\Omega}$ is uniformly chosen from $[1, \overline{\overline{\Omega}}]$ for each new solution. $\overline{\overline{\Omega}} = 1.5$ was the default.

An intuitive explanation of the principle: the worse the patterns which contain the piece, the less promising that piece type, i.e., it does not combine well with

other types. It should be 'packed' with a high priority, so it gets a higher weighting. In the correction scheme (3.15), the overproportional material consumption norm of type i is weighted in correlation with the number of items of type i in the last generated pattern, while the old value is weighted in correlation with the total demand of the type. The scheme of the method is given in Algorithm 3.2.1. To be exact, we should mention that the pseudo-values are not corrected after the last pattern of the solution if it was generated from all remaining items, i.e., if $lb'' \leq L$ made the knapsack problem dispensable.

3.2.4 Computational Results Concerning Material Input

Similar to Section 3.1, we considered classes of problems (m, L, l, b) with parameters $L = 10000$; $m \in \{20, 50, 150\}$; $l_i \in [v_1 L, v_2 L) \cap \mathbb{Z}\ \forall i$ with $v = (v_1, v_2) \in \{(0.01, 0.2), (0.01, 0.4), (0.01, 0.7), (0.2, 0.4), (0.2, 0.7)\}$; $b_i \in S_j\ \forall i$ with $S_1 = [1, 10) \cap \mathbb{Z}$, $S_2 = [1, 100) \cap \mathbb{Z}$, $S_3 = [50, 100) \cap \mathbb{Z}$. The basic class was ($m = 50$, $v = (0.01, 0.07) = v^3$, $b_i \in S_2$). By varying the parameters of the basic class, we obtain the 9 test classes ($m = 20$,,), (,v^1,), ..., (,v^5,), (,,S_1), (,,S_3), ($m = 150$,,). Moreover, by always varying only one of the parameters we avoid an explosion of the number of test classes.

3.2.4.1 Benchmarking the New Heuristic

In Table 3.5 we compare the effectiveness of the heuristic for the main benchmark classes 1, 4, and 9, which are obtained by variation of m, 20 instances per class. SVC parameters: at most 200 iterations were allowed. $z^{\text{1D-CSP}}$ shows the average 1D-CSP optimum for each class; the entries in the main part contain the average best result of SVC. Each line was calculated with a different value of the overproportionality parameter p. The best values for each class are shown in bold.

To refine the search, we calculated all 9 classes, 100 instances per class (Table 3.6). Because the test set size changed, the average 1D-CSP optimum also changed. The entries which seem to represent local minima are bold. Class 2 was solved to optimality completely. Classes 2 and 3 with small product sizes are best solved with $p \approx 1$ because this parameter was introduced to prohibit larger items remaining 'in the tail' of the solution. Classes 6 and 7, as class 1, seem to have their best p-values above 1.04.

The line 'Gap, %' shows the average best optimality gap (distance to the optimum) in per cent. As it can be seen, the heuristic is very powerful: the optimality gap is measured in tenth per cent units. The average number of iterations to find the best solution was about 60 with 200 iterations total; i. e., the search process does not stall. The two lines below the gap expose the results when p is varied

Table 3.5: Average best solution values of SVC for different p

Class	1	4	9
$z^{\text{1D-CSP}}$	348.95	877.4	2656.85
$p = 1$	349.75	880.75	2660.75
$p = 1.02$	349.35	878.55	**2658.15**
$p = 1.05$	349.3	**878.25**	2659.8
$p = 1.1$	**349.2**	879.25	2662.5
$p = 1.15$	349.35	880.3	2664
$p = 1.2$	349.8	880.95	2665.5

Table 3.6: Average best solution values of SVC for different p: a refined search

Class	1	2	3	4	5	6	7	8	9
m^{ave}	19.95	49.85	49.85	49.85	49.85	49.85	49.85	49.85	148.3
$z^{\text{1D-CSP}}$	373.02	261.56	510.21	896.9	749.44	1173.18	90.1	1341.48	2669.56
$p = 1.005$	374.18	**261.56**	**510.24**	898.92	749.87	1179.29	90.29	1343.85	2672.53
$p = 1.01$	374.13	**261.56**	510.3	898.45	749.89	1179.08	90.25	1343.17	2671.54
$p = 1.02$	373.84	261.57	510.32	898.1	749.9	1178.03	90.2	1342.75	**2670.94**
$p = 1.03$	373.76	261.57	510.43	**897.74**	**749.85**	1177.57	90.19	**1342.6**	2671.26
$p = 1.04$	373.66	261.58	510.51	897.81	749.87	1176.79	90.18	1342.62	2671.82
Gap, %	.1716	.0	.0058	.0936	.0547	.3077	.0888	.0835	.0517
$p \in [1.01, 1.03)$ chosen every 100 iterations, 2000 iterations all									
	373.48	261.68	510.91	899.24	749.87	**1173.56**	90.27	1345.01	2676.71
$p \in [1.01, 1.03)$ chosen every 5 iterations, 200 iterations all									
	373.52	261.69	510.91	899.31	750.09	1173.65	90.29	1345.05	2676.77
t/100 iter	.06	1.12	.46	.33	.5	.33	.25	.34	3.8
t_{BP}^{opt}	.01	0.15	.18	.09	.15	.05	.01	.09	2.8

Table 3.7: Effects of randomization, different weighting schemes, and the number of iterations in SVC: average material gap, % of the minimum

Class	1	2	3	4	5	6	7	8	9
default	.1029	0	.0219	.1230	.0593	.3681	.1046	.0910	.0487
$\overline{\overline{\Omega}} = 1$	.0899	0	.0217	.1006	.0583	.3712	.0472	.1157	.0505
WS1	.9463	0	.0219	1.096	.2089	1.083	.8660	.8340	.2961
WS2	.1472	0	.0219	.1222	.0590	.4232	.1046	.1018	.0411
WS3	.1472	0	.0219	.1236	.0527	.3307	.1518	.1390	.0484
it^{1000}	.8999	0	.0219	.0751	.0379	.3057	.1046	.0731	.0393

dynamically. It is not advantageous except for classes 1 and 6 which seem to have their best p-values above 1.04.

In the line 't/100 iter', the time spent for 100 iterations is shown. For comparison, the line t_{BP}^{opt} gives the average optimum solution time when using branch-and-price (Chapter 2). It can be seen that the heuristic cannot compare with the exact method, both in terms of solution quality and time, though it is used inside the latter to solve the smaller residual problems.[‖] The effectiveness of the heuristic even on large-scale problems encourages us to apply it to tasks with several objectives where exact approaches are too complex.

3.2.4.2 Further Parameters of SVC

In Table 3.7, the average optimality gap $(N^z - z^{\text{1D-CSP}})/z^{\text{1D-CSP}}$ is compared for different algorithm parameter settings (20 instances per class). In the upper line there are the default results with $p = 1.02$, $\overline{\overline{\Omega}} = 1.5$, 200 iterations, and weighting scheme (3.15). In weighting scheme 1, $g_1/g_2 = \Omega b_i''/a_i$; in scheme 2, $g_1/g_2 = \Omega b_i'/a_i$; in scheme 3, $g_1/g_2 = \Omega(b_i'' - a_i)/a_i$. The default scheme seems to be the best on average. The effects of randomization are very different. In the last line we see the results with 1000 iterations per instance, which makes some improvement.

3.2.4.3 Problems with Small Order Demands

In 1D-BPP there is no question about the number of setups or open stacks.[**] Thus, we investigate the effectiveness of SVC on such problems separately. The Falkenauer's triplet instances [Fal96] are instances where each bin in the optimum solution contains 3 items without waste. They have been easy for the group genetic

[‖] With at most 20 iterations.

[**] Though solutions with a small number of products per pattern may be desired.

algorithm [Fal96] and exact approaches ([Pee02] and Chapter 2). But SVC finds only solutions with 1 superfluous material length. However, on randomly generated 1D-BPP instances such as class 4 but with $b_i = 1\ \forall i$, all 100 instances were solved optimally with default settings.

We conclude that SVC is very effective on broad classes of 1D-BPP and 1D-CSP excluding some special cases.

3.2.5 PARETO Criterion for the Multiple Objectives

As already mentioned, the number of open stacks is a secondary criterion to evaluate a solution while material input is the main criterion. We say that a solution (N^z, N^o) with N^z material lengths used and N^o maximum open stacks is PARETO-better than solution ($N^{z\prime}$, $N^{o\prime}$) if both criteria are not worse *and* at least one is better: $N^z \leq N^{z\prime}$, $N^o \leq N^{o\prime}$ *and* $(N^z - N^{z\prime}) + (N^o - N^{o\prime}) < 0$. The criterion can be straightforwardly extended to the case where each solution is additionally evaluated by N^d, the number of different patterns.

3.2.6 Updating Prices Considering Open Stacks

During the sequential construction of a solution in SVC, we can observe the current number of open stacks and try not to allow it to increase. For all products which are started but not yet finished after a pattern, in the next pattern generation we multiply their prices by $\pi = \pi_0 \times \rho^k$ where k is the SVC iteration number, $\pi_0 > 1$, and $\rho > 1$. A disadvantage is that we have no control over the first pattern: it is unclear which products should be contained there. In practice this situation can be simplified by priorities assigned to different lots.

We illustrate the solution process by an example. A test instance from class 9 ($m = 150$) is solved. The minimum material input is 2578 stock lengths. When in some iteration a solution is found which is PARETO-better than all previously known according to the 3 criteria (N^o, N^d, N^z), it is added to the set of solutions and printed as (N^o, $N^d[N^{d\prime}]$, N^z), where $N^{d\prime}$ is the number of patterns in the sequence that could be greater than N^d if we had applied some patterns not exhaustively (at the maximum frequency). Settings: $\pi_0 = 1.0002$, $\rho = 1.0001$, $p = 1.02$.

```
"m150l100l7000L10000b1b100f0r0-5" #1 U2578 L2578

Iter 0. Sol. found: (34 148[148] 2579)
Iter 3. Sol. found: (33 157[157] 2580)
Iter 4. Sol. found: (33 154[154] 2580)
Iter 7. Sol. found: (33 151[151] 2579)
Iter 9. Sol. found: (31 156[156] 2580)
```

⋮

```
Iter 120. Sol. found: (12 152[152] 2585)
Iter 137. Sol. found: (10 155[155] 2595)
Iter 139. Sol. found: (11 158[158] 2583)
Iter 162. Sol. found: (12 156[156] 2584)
Iter 175. Sol. found: (9 159[159] 2611)
Iter 183. Sol. found: (10 158[158] 2593)
Iter 185. Sol. found: (10 165[165] 2587)
```

We see that the number of different patterns (setups) N^d is somewhat larger than m. The same can be said about rounded LP solutions for this class. In solutions with more open stacks we have less setups.

3.2.7 Restricting the Number of Open Stacks

In a sequential approach it is possible to control the number of open stacks. Suppose that a maximum of $N^o_{\max}$ open stacks is allowed. When constructing each following pattern in the sequence, the current number of open stacks N^o_{cur} is known. In the new pattern we allow at most $N^o_{\max} - N^o_{\text{cur}}$ new product types, which can be easily controlled in a branch-and-bound pattern generation procedure. Note that several items of a type can be included. Usually each pattern a is applied exhaustively, i.e., at the highest multiplicity $\min_{i:a_i>0}\{\lfloor b''_i/a_i \rfloor\}$, however if $b''_{i_{\min}}$ mod $a_{i_{\min}} > 0$ then product $i_{\min}$ is still open.

An interesting issue is the last pattern of a solution. If we face the situation where all unpacked items fit in one material length ($\sum_i b''_i l_i \leq L$) and the number of remaining types exceeds $N^o_{\max}$ then formally the constraint is violated, but for this single pattern we can ship all types sequentially; thus, the solution remains feasible. In the solution examples this is a thinkable situation: for example, in the instances of class 9, some small products with low demands occur. A related question is whether the presence of a product in only a single stock length needs a separate stack (we assumed yes if not in the last pattern).

In the above example, restricting the number of open stacks to 10 produces the following output:

```
Iter 0. Sol. found: (10 154[154] 2580)
Iter 5. Sol. found: (10 153[153] 2589)
Iter 11. Sol. found: (10 147[147] 2580)
Iter 180. Sol. found: (9 159[159] 2629)
Iter 188. Sol. found: (9 159[159] 2626)
```

Thus, the explicit restriction helps to obtain solutions with better material utilization and fewer different patterns. The last two solutions have only 9 open stacks

(but much more material input) because π_0 and ρ were kept as above. Setting them both equal to 1 makes no significant changes in this example. To get good solutions with 9 open stacks, it is effective to set this limit explicitly. In the Appendix we show a solution for a problem with $m = 50$ using 4 open stacks.

In Table 3.8, 20 instances were solved per class, $p = 1.02$. For each $N^o_{\max}$, the 3 lines show the average and maximum material surplus $100 \times (N^z - z^{\text{1D-CSP}})/z^{\text{1D-CSP}}$ and the number of different patterns. With only 4 open stacks, the maximum material surplus is below 2% while the average is below 0.54%, which is probably acceptable in most industries.

For class 7 with small order demands, the restriction of open stacks is most sensitive in terms of material input since it becomes difficult to generate good patterns with a small number of different product types in a pattern. Class 6 obviously needs another value of p, as it follows from the above experiments. Numbers of patterns are a bit out of line in classes 2 and 3, when compared to the rounded LP solutions. Generally, the fewer open stacks allowed, the greater the material input and the number of setups. For comparison, the last 4 lines show the results plus the average obtained number of open stacks for the unrestricted calculation.

3.2.8 Summary

An approach was tried where open products get a higher weight in pattern generation to prohibit opening of further products. However, the explicit restriction of the number of open stacks produces better material utilization and fewer setups.

3.3 Combined Minimization of Setups and Open Stacks

The heuristic presented in the previous section is simplified and integrated into a pattern minimization approach in order to combine setup and open stacks minimization. The heuristic is simplified to a sequencing procedure to sequence patterns of a given solution, which is done at some nodes of the branch-and-price tree. Thus, we have no explicit control over the number of open stacks, we can only look for good solutions. To further reduce the number of open stacks, we propose to split up the problem, which leads to an increased material input. Again, the PARETO criterion evaluates different solutions in relation to multiple objectives. We are not aware of any previous research aimed at combining setup and open stacks minimization.

Table 3.8: Average/maximum material surplus (%) and the number of setups for SVC with a limited number of open stacks

Class	1	2	3	4	5	6	7	8	9
m^{ave}	19.95	49.85	49.85	49.85	49.85	49.85	49.85	49.85	148.3
$z^{\text{1D-CSP}}$	348.95	257.65	502.3	877.4	743.15	1146.25	88	1321	2656.85
$N^o_{\max} = 2$	2.051	1.729	1.434	2.346	2.734	0.962	4.087	2.094	1.685
	4.273	2.262	1.957	6.155	4.263	2.720	8.490	5.358	2.876
	23.65	70.05	70.1	61.25	65.95	54.05	45.6	63.55	184.25
$N^o_{\max} = 3$	.5250	.1996	.2996	.8692	.6202	.3222	.7208	.6660	.5349
	1.173	.4525	.4640	2.671	.9472	1.188	1.887	2.158	1.022
	21.7	67.35	61.35	56.9	59.75	53	41.4	57.2	170.9
$N^o_{\max} = 4$	.3089	.0392	.1093	.5383	.3504	.3457	.4933	.4236	.3169
	1.928	.4	.2320	1.380	.5755	1.566	1.887	.8766	.6810
	20.75	64.2	59.9	54.25	56.6	52.4	39.55	55.65	163.7
$N^o_{\max} = 6$	.1892	.0392	.0797	.3438	.1741	.3316	.2624	.2469	.1891
	1.830	.4	.2320	1.563	.2813	1.304	1.887	.6263	.3723
	20.25	64.85	57.85	51.45	54.8	51.75	38.35	53.15	155.5
$N^o_{\max} = 10$	.1029	.02	.0521	.1289	.0993	.3655	.1046	.1616	.1197
	.6865	.4	.2320	.4646	.1531	2.219	1.149	.6743	.2462
	20.8	65.45	57.1	50.85	51.9	51.05	37.65	50.4	152.05
$N^o_{\max} = \infty$	.1029	0	.0219	.1230	.0593	.3681	.1046	.0910	.0487
	.6865	0	.2320	.8280	.1531	2.377	1.149	.4720	.1055
	20.45	67.45	60.65	51.1	53.5	51.85	39.7	52.9	150.95
N^o	6.95	17.55	14.7	14.35	13.85	11.15	11.3	16.1	34.85

The branch-and-price scheme for setup minimization (Section 3.1) is used as a framework to provide different solutions which are passed on to a sequencing procedure trying to find a sequence of patterns with a small number of open stacks. There are many sequencing procedures in the literature, e.g., [Yue91b, YBS99]. We modified SVC in the following way: the starting pattern of a new sequence is chosen at random and the following patterns are chosen from the given solution according to the maximum total pseudo-value. 200 sequences are tried. The value correction scheme is $p = 1.02$, $\pi_0 = 1.02$, $\rho = 1.001$.

During branch-and-price, the rounding procedure is applied at each 10th node to investigate the neighborhood of the LP solution by constructing several residual problems (Algorithm 1.4.1). Each residual problem is solved by pure SVC (no open stacks restriction). The solution of the residual is combined with the rounded part giving a solution for the whole problem. Also, a feasible solution may be obtained as an integer solution of the node's LP. If it is better in terms of the number of setups, it is sequenced. If not better, sequencing is carried out with a probability of 0.1.

In Table 3.9, the time limit was $2m$ seconds per instance, other settings were as in Section 3.1. 20 instances were solved in each class. The tests were done for different levels of material surplus. ΔK is the number of stock lengths which can be additionally used in the solution while $z^{\text{1D-CSP}}$ is the minimum. Values of ΔK were integers rounded up from 0, 0.1%, 0.2%, 1%, and 5% of $z^{\text{1D-CSP}}$.

For each ΔK, each couple of lines shows the average number of open stacks and the average number of setups for each class. For $\Delta K = 0$ and $\Delta K = \lceil 0.002 z^{\text{1D-CSP}} \rceil$ we show the average solutions with the best number of setups ('$\min N^d$') and the best number of open stacks ('$\min N^o$'). For all ΔK we show the so-called average *neutral* solutions. For each instance we obtain a set of PARETO-best solutions regarding N^d and N^o. A neutral solution ($\overline{N}^d$, $\overline{N}^o$) minimizes the value of $\overline{N}^d \min N^o + \overline{N}^o \min N^d$, i.e., we are indifferent about the increase either in the one or the other criterion by 1% of its minimum value (if we imagine that the line representing PARETO-best solutions is differentiable). Graphically, the neutral solution lies on the lowest line parallel to that connecting $\min N^o$ and $\min N^d$ on the axes. A practitioner could choose another relation, for example, he could have a (not necessarily linear) cost function of N^o and N^d but our task is to make a compromise between both criteria and not always choose an extreme.

By allowing more material input, we have significantly reduced the number of setups and notably reduced the number of open stacks (20 instead of 30 for $m = 150$). An interesting issue is the relation of '$\min N^d$' and '$\min N^o$'-solutions. Only in classes 2 and 3 (smaller items) we see much difference. For the other classes we may say that setup and open stacks minimization are not contradicting

Table 3.9: Combining setups and open stack minimization

Class	1	2	3	4	5	6	7	8	9
m	20	50	50	50	50	50	50	50	150
$z^{\text{1D-CSP}}$	349	258	502	878	743	1146	88	1321	2657
$\Delta K = 0$									
$\min N^d$	6.35	23.15	18.3	12.5	13.15	6.65	5.85	14	33.7
	15.75	20.7	28.5	40.8	38.4	42.65	24.95	43.15	122.9
$\min N^o$	5.45	17.15	15.35	11.6	11.75	5.4	5.2	12.95	32.35
	17.35	42.45	43.6	42.2	40.3	44.65	25.7	46	125.1
neutral	5.65	22.35	17.25	11.7	11.85	5.4	5.2	13.2	32.5
	16.65	21.15	29.1	41.75	39.6	44.65	25.7	44.4	124.35
$\Delta K = \lceil 0.001 z^{\text{1D-CSP}} \rceil$									
neutral	5.75	21.25	14.8	11	11.3	5.4	4.55	11.45	25.15
	16.55	21.45	28.5	38.8	37.05	43.6	23.15	40.3	106.25
$\Delta K = \lceil 0.002 z^{\text{1D-CSP}} \rceil$									
$\min N^d$	6.65	22.4	15.7	11.65	11.85	6.35	5	11.95	25.7
	15.3	20.65	27.95	36.55	34.7	41.7	22.6	38.05	102.95
$\min N^o$	5.6	16.05	12.2	10.7	10.4	5.3	4.55	10.55	22.25
	17.3	42	48.8	37.4	40.8	43.2	23.15	42.15	115.65
neutral	5.75	21.25	14.65	10.7	10.85	5.3	4.55	10.85	23.25
	16.55	21.45	29	37.4	35.8	43.2	23.15	39.4	105.4
$\Delta K = \lceil 0.01 z^{\text{1D-CSP}} \rceil$									
neutral	5.65	20.6	13.9	9.85	9.2	5.2	4.55	10.05	18.55
	16.2	21.45	28.6	36.6	36.05	43.15	23.15	36.8	98.9
$\Delta K = \lceil 0.05 z^{\text{1D-CSP}} \rceil$									
neutral	5.95	21.1	13.6	10	8.45	5.1	4.3	9.05	18.5
	15.7	21.85	27.6	35.55	35.3	42.3	22.1	35.55	100.45

goals.

3.4 Problem Splitting

> For the chosen class with $m = 150$ product types, in the previous section we achieved, on average, solutions with, e.g., 1% increase of material input, 99 different patterns and 19 open stacks. The results are certainly strongly dependent on problem parameters, see the classes with $m = 50$. To get a smaller number of open stacks, we may split up the problem into several parts of smaller dimension.

Thus, the index set $I = \{1, \ldots, m\}$ of product types is partitioned into subsets $I_l \subset I$, $l = 1, \ldots, n_s$. Then each subproblem $(m_l = |I_l|, L, l_{i \in I_l}, b_{i \in I_l})$ is optimized for setups and open stacks. For material input in a subproblem we set an upper bound $N^z \leq \lceil 1.002 \overline{z}^{\text{1D-CSP}} \rceil$, where $\overline{z}^{\text{1D-CSP}}$ is the value of the best solution found (not all subproblems could be solved optimally). For each subproblem there is produced a set of PARETO-best solutions regarding N^d and N^o. From these partial solutions we form PARETO-best solutions for the whole problem, regarding all 3 criteria N^d, N^o, and N^z. Note that for each splitting variant we get another N^z in a composed solution.

To choose a neutral solution, we have to define a relation of the costs for setups, open stacks, and material. As above, we consider a change of $\varepsilon \min N^d$ in N^d equivalent to $\varepsilon \min N^o$ in N^o. However, material is more costly: we value its change of $(1/20)\varepsilon \min N^z$ equal to 1ε of the minima of both other criteria. Thus, we define a neutral solution $(\overline{N}^d, \overline{N}^o, \overline{N}^z)$ to minimize the expression

$$\overline{N}^d \min N^o \min N^z + \overline{N}^o \min N^d \min N^z + 20\overline{N}^z \min N^d \min N^o. \qquad (3.16)$$

Again, this is done only in order to find a 'compromise' solution; a practitioner could choose another cost function.

3.4.1 Splitting Methods

We tried only random partitioning of the product set. Two partitioning strategies were applied. In the first method, each product type was independently assigned to a subproblem $l \in \{1, \ldots, n_s\}$. For $m = 150$ and $n_s = 3$, the resulting values of m_l were observed between 30 and 70. In the second method, the subproblem sizes were kept nearly equal, i.e., $m_l \leq \lceil m/n_s \rceil$.

Some splitting variants may be very disadvantageous. In an instance with $m = 150$, the first random splitting into 3 parts with $m_l = 50$ produced a total

LP bound that was by 10% greater than that in the whole problem. The second splitting variant was only 2% greater. For each instance we tried 100 variants and chose a few of them with the smallest total material input (for $100n_s$ subproblems, $n_s = 3$, this was less than a minute of CPU time on average).

Another approach to split up a problem would be to take a solution with a small number of open stacks produced by SVC and to select subsets of items contained in subsequences of cutting patterns. This way could lead to better material utilization in each subproblem. However, each product type is usually contained in several patterns, so even here difficulties are possible. Also, no systematic method to improve a given splitting has been found. Note that in the industry the task would be easier: we usually have lots (groups of products) and lot priorities.

3.4.2 Computational Results

In Table 3.10 we show some average results for the 20 instances of class 9 and two other classes with $m = 150$. The first 3 data columns give average absolute values (N^o, N^d, N^z) for the neutral composed solutions. $\underline{N}^z$ is the minimum material input obtained for the whole (non-split) problem. $\underline{N}^d$ and $\underline{N}^o$ are the minimum N^d and N^o obtained when solving the whole problem with allowed increase of material input equal to that in the best composed solution. $t^{\text{1D-CSP}}$ is the average time needed to optimize the $100n_s$ subproblems for material input with a time limit of 300 seconds per subproblem. $N^{\overline{\text{Opt}}}_{\text{1D-CSP}}$ is the number of subproblems for which the 1D-CSP could not be solved optimally (counting for all 20 whole instances). $\underline{m_l}$, $\overline{m_l}$ are the minimum and maximum m_l. The time limit was $\overline{m_l}$ seconds for a subproblem and 300 seconds for a whole problem.

For comparison to the case without splitting, line 1 repeats data from Table 3.9 with $\Delta K = \lceil 1\%(z^{\text{1D-CSP}}) \rceil$ for class 9. For the line '10/100', 10 splittings with the best material input were chosen from 100 obtained by the first method ($n_s = 3$). For the line '3/100', only 3 best were chosen. This leads to a better N^z but worse N^o and N^d. For the line '$m_l \leq 50$' (the second splitting method), 3 variants out of 100 were tried. We see that subproblems with nearly equal size, i.e., obtained by the second splitting method, give worse results. For the line '10/10', all 10 variants obtained by the second method were optimized. As no selection regarding material input was made, both other criteria have good levels. For $n_s = 5$ and the first splitting method we obtain 8.3 stacks and a material input increase of 1.6%. Classes 3' and 8' resemble those from Table 3.9 but with $m = 150$, i.e., class 3' had $v_2 = 0.4$ and class 8' had $b_i \in S_3$.

Table 3.10: Setups and open stacks minimization combined: further reduction of open stacks by problem splitting for $m = 150$

Class 9	N^o	N^d	N^z	$N^o/\underline{N}^o$	$N^d/\underline{N}^d$	$N^z/\underline{N}^z$	$t^{\text{1D-CSP}}$	$N^{\overline{\text{Opt}}}_{\text{1D-CSP}}$	$\underline{m}_l$	$\overline{m}_l$
NoSplit	18.55	98.9				1.01				
10/100	10.7	112.95	2673.1	0.56	1.13	1.00612	28.4	0	30	69
3/100	10.95	114.1	2672.15	0.57	1.14	1.00576	28.3	0	30	69
$m_l \leq 50$	11.2	113.35	2673.1	0.58	1.14	1.00612	41.9	1	48	50
10/10	10.7	113.7	2680.3	0.56	1.16	1.00881				
$n_s = 5$	8.3	117.05	2699.2	0.47	1.20	1.01594	29.5	1	15	48
Class 3’	14.95	84.95	1537.35	0.82	1.21	1.00307	57.0	0	30	70
Class 8’	11.35	118.15	3997.9	0.65	1.21	1.00536	29.8	0	30	69

3.5 IP Models for Open Stacks Minimization

We extend some assignment models of 1D-CSP, including the model of Kantorovich, by boolean variables that account the states of stacks in the cutting sequence. A similar extension of the Gilmore-Gomory model and the subpattern model is possible. The continuous relaxation needs some further variables in order to take the stack states into account. The models are probably too large for to-day optimizers.

Let $\overline{K}$ be an upper bound on the number of different patterns in an optimum solution. Let $x_{ij} \in \{0, \ldots, b_i\}$ be the number of items of type i in the j-th pattern ($j = 1, \ldots, \overline{K}$) and z_j be the intensity of pattern j. Let

$$\overleftarrow{s}_{ij} = \begin{cases} 0, & \text{if } \sum_{k=1}^{j} x_{ik} z_k = 0 \quad \text{Product } i \text{ not started up} \\ & \qquad\qquad\quad \text{to pattern } j \\ 1, & \text{if } \sum_{k=1}^{j} x_{ik} z_k > 0 \quad \ldots \text{already started} \end{cases} \tag{3.17}$$

and

$$\underleftarrow{s}_{ij} = \begin{cases} 0, & \text{if } \sum_{k=1}^{j-1} x_{ik} z_k < b_i \quad \text{Product } i \text{ not finished} \\ & \qquad\qquad\quad \text{before pattern } j \\ 1, & \text{if } \sum_{k=1}^{j-1} x_{ik} z_k \geq b_i \quad \ldots \text{already finished} \end{cases} \tag{3.18}$$

for $i = 1, \ldots, m$, $j = 1, \ldots, \overline{K}$.
Note that $\overleftarrow{s}_{ij} = 1$ for $j = \overline{K}$ and $\underleftarrow{s}_{ij} = 0$ for $j = 1, \forall i$.

Alternatively, we may define $\underleftarrow{s}_{ij} = 1 - \overrightarrow{s}_{ij}$ with

$$\overrightarrow{s}_{ij} = \begin{cases} 1, & \text{if } \sum_{k=j}^{\overline{K}} x_{ik} z_k > 0 \quad \text{Product } i \text{ not finished} \\ & \qquad\qquad\quad \text{before pattern } j \\ 0, & \text{if } \sum_{k=j}^{\overline{K}} x_{ik} z_k = 0 \quad \ldots \text{already finished.} \end{cases} \tag{3.19}$$

Now we can define a linear function

$$\text{open}(i, j) = \overleftarrow{s}_{ij} - \underleftarrow{s}_{ij} \in \{0, 1\}. \tag{3.20}$$

Let $maxOS$ be the maximum number of open stacks:

$$maxOS \geq \sum_{i=1}^{m} \text{open}(i, j), \qquad \forall j \tag{3.21}$$

and $maxSpr$ be the maximum spread:

$$maxSpr \geq \frac{1}{l_i b_i} \sum_{j=1}^{\overline{K}} \text{open}(i, j) \times z_j, \qquad \forall i. \tag{3.22}$$

The model has one of the following objectives:

$$\min\ maxOS \qquad \text{or} \qquad \min\ maxSpr \tag{3.23}$$

subject to

$$\sum_{k=1}^{j} x_{ik} z_k \le b_i \overleftarrow{s}_{ij}, \qquad \forall i, j \tag{3.24}$$

$$\sum_{k=j}^{\overline{K}} x_{ik} z_k \le b_i \overrightarrow{s}_{ij}, \qquad \forall i, j \tag{3.25}$$

$$\sum_{i=1}^{m} l_i x_{ij} \le L, \qquad \forall j \tag{3.26}$$

$$\sum_{j=1}^{\overline{K}} z_j \le z^{1D-CSP} \tag{3.27}$$

$$\sum_{j=1}^{\overline{K}} x_{ij} z_j \ge b_i, \qquad \forall i \tag{3.28}$$

$$x_{ij} \in \{0, \ldots, b_i\}, \qquad \forall i, j \tag{3.29}$$

$$z_j \in \{0, \ldots, z^{1D-CSP}\}, \qquad \forall j \tag{3.30}$$

$$\overleftarrow{s}_{ij}, \overrightarrow{s}_{ij} \in \{0, 1\}, \qquad \forall i, j. \tag{3.31}$$

This model is non-linear. To linearize it, let us introduce separate variables for each stock length's pattern like in the Kantorovich model: $y_j = 1$ if stock length j is used in the solution, otherwise $y_j = 0$ ($j = 1, \ldots, z^{1D-CSP}$). The constraints are then

$$\sum_{k=1}^{j} x_{ik} \le b_i \overleftarrow{s}_{ij}, \qquad \forall i, j = 1, \ldots, z^{1D-CSP} \tag{3.32}$$

$$\sum_{k=j}^{z^{1D-CSP}} x_{ik} \le b_i \overrightarrow{s}_{ij}, \qquad \forall i, j \tag{3.33}$$

$$\sum_{i=1}^{m} l_i x_{ij} \le L y_j, \qquad \forall j \tag{3.34}$$

$$\sum_{j=1}^{z^{1D-CSP}} x_{ij} \ge b_i, \qquad \forall i \tag{3.35}$$

$$x_{ij} \in \{0, \ldots, b_i\}, \qquad \forall i, j \tag{3.36}$$

$$y_j \in \{0, 1\}, \qquad \forall j \tag{3.37}$$

$$\overleftarrow{s}_{ij}, \overrightarrow{s}_{ij} \in \{0, 1\}, \qquad \forall i, j \tag{3.38}$$

and the definition of $maxSpr$ has to be correspondingly changed:

$$maxSpr \ge \frac{1}{l_i b_i} \sum_{j=1}^{z^{1D-CSP}} \mathrm{open}(i, j), \qquad \forall i. \tag{3.39}$$

Now, theoretically we may have a different pattern for each physical stock length. To maintain information about open stacks, there are $2m$ boolean variables for each physical stock length. If the material input in the solution is high then the model grows very large.

We can decompose the model to a form with column generation as in the Gilmore-Gomory model. Alternatively, extending the subpattern model (Chapter 1) to handle open stacks in a similar way seems to be an interesting topic.

In the continuous relaxation, $\mathrm{open}(i,j)$ will be 0 for the patterns not containing item i. Thus, we will be only in some sense minimizing the number of items in a pattern. When using this relaxation in an enumerative approach, we cannot expect it to lead the search process toward good solutions.

To enable the relaxation to take some approximation of the 'open' stack state into account even for patterns not containing an item, let us introduce help boolean variables

$$\overrightarrow{ds}_{ij} = \max\{\overrightarrow{ds}_{i,j+1}, \overleftarrow{s}_{ij} - \underleftarrow{s}_{ij}\} \tag{3.40}$$

and

$$\overleftarrow{ds}_{ij} = \max\{\overleftarrow{ds}_{i,j-1}, \overleftarrow{s}_{ij} - \underleftarrow{s}_{ij}\}, \tag{3.41}$$

and redefine

$$\mathrm{open}(i,j) = \overrightarrow{ds}_{ij} + \overleftarrow{ds}_{ij} - 1 \tag{3.42}$$

or

$$\mathrm{open}(i,j) = \overrightarrow{ds}_{ij} + \overleftarrow{ds}_{ij} - mds_{ij} \tag{3.43}$$

with

$$mds_{ij} = \max\{\overrightarrow{ds}_{ij}, \overleftarrow{ds}_{ij}\} \tag{3.44}$$

which may better reflect the fractional values in the relaxation.

When branching on variables ds, setting, e.g., $\overrightarrow{ds}_{ij} = 0$ implies $\overrightarrow{s}_{ij} = 0$ and $\sum_{k=j}^{z^{1D-CSP}} x_{ij} = 0$; setting $\overrightarrow{ds}_{ij} = 1$ implies $\overrightarrow{s}_{ij} = 1$ and $\sum_{k=j}^{z^{1D-CSP}} x_{ij} \geq 1$. It is an interesting question whether CPLEX node preprocessing routines can find these implications. In the root node relaxation, variables s are redundant with ds, but we may branch on them, as an alternative to branching on ds.

3.6 Summary and Outlook

The investigation of the proposed methods on a broad range of test problem classes has shown the following.

Largely, the effects of problem parameters are: smaller products allow less patterns but necessitate more stacks. Smaller order demands improve both criteria. About 25% of the instances of the chosen class with $m = 150$ seem to have only a few material-minimum solutions so that setup minimization is not very successful. A small increase of material input reduces the difficulty.

The proposed modification of the sequential heuristic SVC shows an average material surplus under 1% even for 3 open stacks allowed in problems with up to 150 product types. Previous published approaches, based on sequencing of a

single solution, gave as many as 130 open stacks for 150 product types; some other more flexible approaches were not documented on representative test sets.

A combined setup/open stacks minimization approach gives as many as 25 open stacks for the standard class with 150 product types when allowing material surplus of 0.2%. This can be reduced down to 11 open stacks when splitting up the problem into 3 parts; material surplus is about 0.6% then.

The options of lots and of several trucks for a lot (see Introduction to Section 3.2) can be incorporated in SVC as follows. If after applying a pattern at a (not necessarily maximum) frequency, some trucks/stacks are finished, then we are free whether to continue the open lot with the next truck or to switch to another lot. Some further technological criteria may apply here.

Chapter 4

Models Without Column Generation for 2D-2CP

We consider some assignment formulations of 2D-2CP. Some new models with *variable strip widths* are developed. Symmetries in the search space are eliminated by *lexicographic constraints* which are already known from the literature. However, previously known models with fixed strip widths are more effective (but on large instances both are inferior to the Gilmore-Gomory model used in Chapter 2). The models are solved with the branch-and-cut algorithm of CPLEX.

4.1 Models with Fixed Strip Widths from the Literature

Consider the two ILP models for 2D-2CP from [LM02]. They distinguish between items *initializing* a strip and *additional* items in the strip. The initializer width determines the width of the strip.

4.1.1 Model 1

Each item is considered distinct, i.e., for each piece type i ($i = 1, \ldots, m$) we define b_i identical items j with $\overline{l}_j = l_i$, $\overline{w}_j = w_i$, $\overline{p}_j = p_i$. Let $n = \sum_{i=1}^{m} b_i$ be the total number of items. Let the items be ordered so that $\overline{w}_1 \geq \ldots \geq \overline{w}_n$. The model assumes that n potential strips may be initialized: strip k, if used, must be initialized by item k ($k = 1, \ldots, n$). Then, the possible cutting of the n items

from the potential strips is described by the following binary variables:

$$x_{jk} = \begin{cases} 1, & \text{if item } j \text{ is cut from shelf } k, \\ 0, & \text{otherwise.} \end{cases} \quad (k = 1, \ldots, n; j = k, \ldots, n) \tag{4.1}$$

Model M1 is then as follows:

$$\max \textstyle\sum_{j=1}^{n} \overline{p}_j \sum_{k=1}^{j} x_{jk} \tag{4.2}$$

$$\text{s.t.} \quad \textstyle\sum_{k=1}^{j} x_{jk} \leq 1 \quad (j = 1, \ldots, n) \tag{4.3}$$

$$\textstyle\sum_{j=k+1}^{n} \overline{l}_j x_{jk} \leq (L - \overline{l}_k) x_{kk} \quad (k = 1, \ldots, n-1) \tag{4.4}$$

$$\textstyle\sum_{k=1}^{n} \overline{w}_k x_{kk} \leq W \tag{4.5}$$

$$x_{jk} \in \{0, 1\}. \quad (k = 1, \ldots, n; j = k, \ldots, n) \tag{4.6}$$

The objective function (4.2) maximizes the sum of the profits of the cut items. Inequalities (4.3) guarantee that each item is cut at most once, and only from strips whose width is at least equal to the width of the item. Inequalities (4.4) assure that the length constraint for each strip is satisfied, and that either item k is on strip k or strip k is empty, whereas inequality (4.5) imposes the width constraint. Note that the meaning of each variable x_{kk} $(k = 1, \ldots, n)$ is twofold: $x_{kk} = 1$ implies that item k is cut from strip k, i.e., strip k is used and initialized by its corresponding item. The model has $O(n^2)$ variables and $O(n)$ constraints.

4.1.2 Model 2

The next model from [LM02] has an integer variable for items of a certain type in a strip. Let $\alpha_0 \equiv 0$, $\alpha_i = \alpha_{i-1} + b_i$, $n = \alpha_m$. Let $\beta_k = \min\{i : \alpha_i \geq k\}$ be the initial piece type in strip k, $k = 1, \ldots, n$. Let q_k denote initialization of strips: $q_k = 1$, if strip k is used **and** at least one piece of type β_k initializes it. Then $x_{\beta_k k}$ is the number of **additional** items of type β_k. Let $w_1 \geq w_2 \geq \ldots \geq w_m$ and x_{ik} be the number of items of type i to be cut in strip k, $i > \beta_k$. Model M2 is as follows:

$$\max \textstyle\sum_{i=1}^{m} p_i \sum_{k=1}^{\alpha_i} x_{ik} + \sum_{k=1}^{n} p_{\beta_k} q_k \tag{4.7}$$

$$\text{s.t.} \quad \textstyle\sum_{k=1}^{\alpha_i} x_{ik} + \sum_{k=\alpha_{i-1}+1}^{\alpha_i} q_k \leq b_i \quad (i = 1, \ldots, m) \tag{4.8}$$

$$\textstyle\sum_{i=\beta_k}^{m} l_i x_{ik} \leq (L - l_{\beta_k}) q_k \quad (k = 1, \ldots, n) \tag{4.9}$$

$$\textstyle\sum_{k=1}^{n} w_{\beta_k} q_k \leq W \tag{4.10}$$

$$x_{ik} \in \mathbb{Z}_+ \quad (\forall i,\ k = 1, \ldots, \alpha_i) \tag{4.11}$$

$$q_k \in \mathbb{B}. \quad (k = 1, \ldots, n) \tag{4.12}$$

The model has $O(mn)$ variables and $O(n)$ constraints. The Gilmore-Gomory model can be obtained from it by DANTZIG-WOLFE decomposition [NW88].

4.1.3 Anti-Symmetry Constraints

The authors prove that the following constraints do not exclude all optimum solutions of M1 and M2.

- *Ordering inequalities* (OI) for the strip initializers:

$$x_{tt} \geq x_{t+1,t+1} \qquad (t \in [\alpha_{i-1}+1, \alpha_i - 1]\,\forall i) \tag{4.13}$$

  for M1 and

$$q_t \geq q_{t+1} \qquad (t \in [\alpha_{i-1}+1, \alpha_i - 1]\,\forall i) \tag{4.14}$$

  for M2.

  Note that these imply $x_{\alpha_{i-1}+1+t,\alpha_{i-1}+1+t} = q_{\alpha_{i-1}+1+t} = 0\,\forall t > \lfloor W/w_i \rfloor, \forall i$. Such implications can be possibly found by CPLEX preprocessing routines.

- *Extended ordering inequalities* (OI) for the additional items of the initializer type:

$$\sum_{s=t+1}^{\alpha_i} x_{st} \geq \sum_{s=t+2}^{\alpha_i} x_{s,t+1} \qquad (t \in [\alpha_{i-1}+1, \alpha_i - 2]\,\forall i) \tag{4.15}$$

  for M1 and

$$x_{it} \geq x_{i,t+1} \qquad (t \in [\alpha_{i-1}+1, \alpha_i - 1]\,\forall i) \tag{4.16}$$

  for M2.

OI and EOI can be seen as basic lexicographic ordering (see below).

4.1.4 Tightening the LP Relaxation of M2

The authors [LM02] show that when inequalities

$$\sum_{s=k}^{\alpha_i} x_{is} \leq b_i - (k - \alpha_{i-1}) \qquad (k \in [\alpha_{i-1}+1, \alpha_i],\ \forall i) \tag{4.17}$$

are added to M2, then M1 and M2 have equivalent LP relaxations. However, for the non-relaxed model (4.7)–(4.12) these inequalities follow from the anti-symmetry constraints EOI:

Proposition 11 *In model M2, EOI* (4.16) *imply* (4.17).

Proof For $k = \alpha_{i-1} + 1$, (4.17) follows from (4.8) and (4.9). Suppose (4.17) is violated for some $k > \alpha_{i-1} + 1$. Then $\sum_{s=k}^{\alpha_i} x_{is} \geq 1$. EOI (4.16) mean $x_{it} \geq 1$, $\forall t \in [\alpha_{i-1}+1, k-1]$; summing these up gives $\sum_{s=\alpha_{i-1}+1}^{k-1} x_{is} \geq k - \alpha_{i-1} - 1$; together with the violation of (4.17), $\sum_{s=\alpha_{i-1}+1}^{\alpha_i} x_{is} > b_i - 1$. This together with (4.8) gives $\sum_{s=\alpha_{i-1}+1}^{\alpha_i} q_s < 1$ which is a contradiction to (4.9). ∎

4.2 Variable Width Models

Notice that the maximum number of strips in a solution is not greater than $N = \lfloor W/w_{min} \rfloor$ which is usually smaller than n. This bound can be reduced by considering the bounds b_i for each piece type. A further reduction is can be achieved by the following

Proposition 12 *Let* $\overline{w}_1 \leq \overline{w}_2 \leq \ldots \leq \overline{w}_n$ *and let* $b_j = 1\ \forall j = 1, \ldots, n$. *Let* $\gamma_0 = 0$ *and* $\gamma_k = \max\{\gamma : \overline{l}_{\gamma_{k-1}+1} + \ldots + \overline{l}_\gamma \leq L\}$, $k = 1, 2, \ldots$. *Then* $N = \max\{k : \sum_{p=1}^{k} \overline{w}_{\gamma_p} \leq W\}$ *is a valid upper bound on the number of active strips.*

Proof A better solution (with a better objective value) than that implied by the construction of $\{\gamma_k\}_{k=1,2,\ldots}$ can be obtained by: 1. more dense packing of current items; 2. introducing further items into the solution. Both ways can not create any narrower strips. ■

Let us make the width of each strip variable: $\widetilde{w}_k$, $k = 1, \ldots, N$. To restrict the width of each strip by the widest item in it, let us consider each item to be distinct as in M1 and represent the items by binary variables x_{jk} meaning that item j is present in strip k. This gives model M3:

$$\max \sum_{j,k} \overline{p}_j x_{jk} \tag{4.18}$$

$$\text{s.t.} \quad \widetilde{w}_k \geq \overline{w}_j x_{jk}, \quad \forall j, k \tag{4.19}$$

$$\sum_k \widetilde{w}_k \leq W \tag{4.20}$$

$$\sum_j \overline{l}_j x_{jk} \leq L, \quad \forall k \tag{4.21}$$

$$\sum_k x_{jk} \leq 1, \quad \forall j \tag{4.22}$$

$$x_{jk} \in \mathbb{B},\ \forall j, k, \qquad \widetilde{w}_k \in \mathbb{Z}_+,\ \forall k \tag{4.23}$$

which has $O(nN)$ constraints and variables. But if the original problem has large upper bounds, the transformation to the equivalent problem with distinct items increases the dimension significantly. Thus, it could be sensible to have one binary variable q_{ik} for each piece type, which also restricts the strip width, and an integer variable x_{ik} giving the number of **additional** items of type i in the strip. However, we must not allow any additional pieces without restricting the width, thus $x_{ik} \leq$

$(b_i - 1)q_{ik}$. Model M4 is as follows:

$$\max \sum_{i,k} p_i(q_{ik} + x_{ik}) \tag{4.24}$$
$$\text{s.t.} \quad \widetilde{w}_k \geq w_i q_{ik}, \quad \forall i,k \tag{4.25}$$
$$\sum_k \widetilde{w}_k \leq W \tag{4.26}$$
$$x_{ik} \leq (b_i - 1)q_{ik}, \quad \forall i,k \tag{4.27}$$
$$\sum_i l_i q_{ik} + \sum_i l_i x_{ik} \leq L, \quad \forall k \tag{4.28}$$
$$\sum_k q_{ik} + \sum_k x_{ik} \leq b_i, \quad \forall i \tag{4.29}$$
$$x_{ik} \in \mathbb{Z}_+,\ q_{ik} \in \mathbb{B}\ \forall i,k, \qquad \widetilde{w}_k \in \mathbb{Z}_+,\ \forall k. \tag{4.30}$$

This model is equivalent to M3, even in terms of LP relaxation, if $b_i = 1\ \forall i$ already holds. To get rid of the mN constraints (4.25), let us change the sense of q_{ik} to correspond to **the widest** item in the strip. Assume $w_i \geq w_{i+1}$ for $i = 1, \ldots, m-1$. Model M5 is as follows:

$$\max \sum_{i,k} p_i x_{ik} \tag{4.31}$$
$$\text{s.t.} \quad \widetilde{w}_k = \sum_i w_i q_{ik}, \quad \forall k \tag{4.32}$$
$$1 \geq \sum_i q_{ik}, \quad \forall k \tag{4.33}$$
$$\sum_k \widetilde{w}_k \leq W \tag{4.34}$$
$$x_{ik} \leq b_i \sum_{l=1}^{i} q_{lk}, \quad \forall i,k \tag{4.35}$$
$$\sum_i l_i x_{ik} \leq L, \quad \forall k \tag{4.36}$$
$$\sum_k x_{ik} \leq b_i, \quad \forall i \tag{4.37}$$
$$x_{ik} \in \mathbb{Z}_+,\ q_{ik} \in \mathbb{B}\ \forall i,k, \qquad \widetilde{w}_k \in \mathbb{Z}_+,\ \forall k \tag{4.38}$$

which has about 2 times fewer constraints. But when $b_i = 1$, $\forall i$, it has twice as many active variables.

For the LP relaxation of M5, it is reasonable to tighten (4.36) by $\sum_i l_i x_{ik} \leq L \sum_i q_{ik}$. Also, in (4.35) and (4.27) we can replace b_i by $\min\{b_i, \lfloor L/l_i \rfloor\}$. For all models M3, M4, M5 it is reasonable to tighten the search space of a branch-and-bound by

$$\widetilde{w}_k \geq \widetilde{w}_{k+1}, \quad k = 1, \ldots, N-1 \tag{4.39}$$

and for models M4, M5 by

$$q_{1k} \geq q_{1,k+1}, \quad k = 1, \ldots, N-1 \tag{4.40}$$
$$x_{1k} \geq x_{1,k+1}, \quad k = 1, \ldots, N-1 \tag{4.41}$$

(for M3 that would be (4.41) only) which are the analogs of OI and EOI for the fixed-width models.

4.3 Lexicographical Branching

In all models M1–M5 there are many equivalent groups of variables. For example, in the models with variable width, all groups of variables representing strips are interchangeable, so that equivalent (*symmetric*) solutions are produced by exchanging them. Thus, an enumerative approach could investigate many equivalent subproblems. To introduce some basic anti-symmetry protection, we can add (4.39)–(4.41) in the models with variable width and OI, EOI in the fixed width models. However, if the initial variables are equal in some two strips, then the rest is interchangeable. We can define some kind of dynamic lexicographical order of the subproblems of the current branch&bound tree with respect to equivalent solutions they may contain. It is especially important to prune equivalent subproblems at the initial levels of the branch&bound tree to reduce the amount of nodes investigated.

Consider a general model

$$\max\{cx : \; Ax = b, \; \mathcal{L} \leq x \leq \mathcal{U}, \; x \in \mathbb{Z}^n\},$$

where some groups of variables are equivalent.

Lexicographic pruning rule *Let* $(x_{i_1}, \ldots, x_{i_1+\gamma})$ *and* $(x_{i_2}, \ldots, x_{i_2+\gamma})$ *be equivalent variable groups and* $i_1 < i_2$*. Suppose that at the current node the restrictions on both groups are equal except for two corresponding variables:* $(\mathcal{L}_{i_1+\beta}, \mathcal{U}_{i_1+\beta}) = (\mathcal{L}_{i_2+\beta}, \mathcal{U}_{i_2+\beta})$ *for all* $\beta \neq \beta_0$ *and* $\mathcal{U}_{i_1+\beta_0} < \mathcal{L}_{i_2+\beta_0}$*. Then prune the current node.*

Proposition 13 *When applied alone (without further ordering of equivalent groups), the above rule does not cause any loss of solutions, i.e., at least one of a set of equivalent solutions will not be pruned.*

Proof Suppose the constraint $x_{i_1+\beta_0} \leq \mathcal{U}_{i_1+\beta_0}$ has been added at level d_1 of the solution tree and the constraint $x_{i_2+\beta_0} \geq \mathcal{L}_{i_2+\beta_0}$ at level d_2. Let $d_1 < d_2$. Then the opposite subproblem at level d_1 is defined by $x_{i_1+\beta_0} \geq \mathcal{U}_{i_1+\beta_0} + 1$ and thus this subproblem is equivalent or even larger than the one being pruned because of the interchangeability of the variable groups. It will not be pruned by the rule. Similar in the case $d_2 < d_1$. ■

We can suppose (and be confirmed by tests) that there are very few cases when the rule is applicable. The assumptions are very strong.

4.4 Lexicographically Ordered Solutions

Instead of ordering the subproblems dynamically during branching, we may statically order the solutions themselves. For each couple of equivalent variable groups $(x_{i_1}, \ldots, x_{i_1+\gamma})$ and $(x_{i_2}, \ldots, x_{i_2+\gamma})$ for $i_1 < i_2$, we may demand that the first vector should be lexicographically not smaller than the second one. This is done in [cpl01, ilo02] in an example of application of Constraint Programming to the Kantorovich model of 1D-CSP. Suppose that the variables have bounds

$$0 \le x_{i+\kappa} \le \mathcal{U}_{i+\kappa} \qquad (\kappa \in [0, \gamma],\ i \in \{i_1, i_2\}) \tag{4.42}$$

(in our case such upper bounds, e.g., for the number of items of type i, would be $\min\{b_i, \lfloor L/l_i \rfloor\}\ \forall i$). Then the function

$$\prod_{\kappa=1}^{\gamma}(\mathcal{U}_{i+\kappa}+1)x_i + \prod_{\kappa=2}^{\gamma}(\mathcal{U}_{i+\kappa}+1)x_{i+1} + \cdots + x_{i+\gamma} \qquad (i \in \{i_1, i_2\}) \tag{4.43}$$

assigns a unique 'lexicographic value' to the vectors. Its coefficients can grow very large, and with numbers like 10^{15} CPLEX had numerical problems on some instances while on the others the results were significantly better because CPLEX scales the data during preprocessing. Thus, we tried all instances allowing maximum coefficients of 10^8, reducing the length of the compared vectors if necessary.

The following vectors are equivalent in the corresponding models:

- In model M1, for each $i = 1, \ldots, m$, the strips initialized by piece i are equivalent:

$$\left(\sum_{j=k}^{\alpha_i} x_{jk},\ \sum_{j=\alpha_i+1}^{\alpha_{i+1}} x_{jk},\ \ldots,\ \sum_{j=\alpha_{m-1}+1}^{n} x_{jk} \right), \qquad k \in [\alpha_{i-1}+1, \alpha_i]. \tag{4.44}$$

- Similar for M2:

$$(q_k + x_{ik},\ x_{i+1,k},\ \ldots,\ x_{mk}), \qquad k \in [\alpha_{i-1}+1, \alpha_i]. \tag{4.45}$$

- For the variable width models, all strips are equivalent. M3:

$$\left(\sum_{j=1}^{\alpha_1} x_{jk},\ \sum_{j=\alpha_1+1}^{\alpha_2} x_{jk},\ \ldots,\ \sum_{j=\alpha_{m-1}+1}^{n} x_{jk} \right), \qquad k = 1, \ldots, N. \tag{4.46}$$

- For M4:

$$(q_{1k} + x_{1k},\ \ldots,\ q_{mk} + x_{mk}), \qquad k = 1, \ldots, N. \tag{4.47}$$

- For M5:

$$(x_{1k},\ \ldots,\ x_{mk}), \qquad k = 1, \ldots, N \tag{4.48}$$

and

$$(q_{1k},\ \ldots,\ q_{mk}), \qquad k = 1, \ldots, N. \tag{4.49}$$

4.5 Computational Results

The lexicographic pruning rule (dynamic lexicography) was not efficient: in many instances no pruning occurred at all. Among the variable width models, we implemented at first M4; however, its results were dominated by M1 and M2, thus, M3 and M5 were not tested. The test set was the same as in Chapter 2 [LM02, HM03]: 14 medium weighted, 24 medium unweighted, 10 large unweighted, 10 large weighted instances. The models were implemented in ILOG Concert Technology 1.2 over CPLEX 7.5 and tested on an AMD K7 Athlon XP 1000 MHz. In all models, we assigned higher branching priorities to strip initializers [LM02]; in M4, the strip width variables w_k obtained the lowest priority and the variables q_{ik} the highest.

In Tables 4.1 and 4.2, t is the time without lexicographical constraints, t^{lex} when also the lexicographical constraints are added with maximum coefficients of 10^8. t^{old} is the time from [LM02] computed with CPLEX 6.5.3 on a Digital Alpha 533 MHz. The maximum time was 10 minutes, thus all greater times mean suboptimal solution. Results for M4 on the large set are not satisfactory and thus not shown: only 2 instances are solved optimally and the bound gap is not acceptable (a few per cent).

The last 10 instances of the large class are solved better than the first 10 instances. A reason is that the last 10 instances are weighted and the weights are rather disproportional to piece areas. This peculiarity does not occur in the Gilmore-Gomory model, Chapter 2.

Lexicographical ordering is on average better with M2 and M4 on the medium set, with M1 on the medium unweighted set. To find out, why lexicography gave better results for M1 on instances CU1 and Hchl3s, we looked at the problem data: while in most instances the item sizes are more than 1/5 of the corresponding dimension, in those instances it is down to 1/10. But we could not construct a class with systematically better results obtained when using lexicography.

Table 4.1: 2D-2CP: medium+large classes, models M1, M2, and M4, first cut along the first dimension

				M1			M2			M4	
name	m	n	obj	t^{old}	t	t^{lex}	t^{old}	t	t^{lex}	t	t^{lex}
HH	5	18	10689	0.28	0.05	0.07	0.05	0.03	0.03	0.08	0.1
2	10	23	2535	0.35	0.14	0.14	0.32	0.1	0.1	6.4	3.97
3	19	62	1720	0.35	0.12	0.18	0.23	0.14	0.11	0.7	0.73
A1	19	62	1820	0.88	0.24	0.37	0.40	0.15	0.17	0.73	0.41
A2	20	53	2315	1.33	0.32	0.78	0.58	0.36	0.28	0.56	0.48
STS2	30	78	4450	16.82	6.88	8.19	15.27	5.08	4.81	68.29	34.8
STS4	20	50	9409	11.42	4.54	5.18	9.98	1.19	0.86	9.54	10.64
CHL1	30	63	8360	8.30	2.79	3.67	4.00	2.62	2.95	11.97	11.39
CHL2	10	19	2235	0.12	0.03	0.04	0.13	0.03	0.04	0.3	0.32
CW1	25	67	6402	2.32	1.02	3.76	0.82	0.26	0.28	0.36	0.48
CW2	35	63	5354	0.87	0.31	0.48	0.78	0.32	0.23	0.37	0.63
CW3	40	96	5287	2.55	0.83	0.93	1.72	0.32	0.48	0.83	0.6
Hchl2	35	75	9630	61.77	12.86	17.23	300.02	30.87	20.18	601.81	511.68
Hchl9	35	76	5100	3.62	2.09	1.53	1.90	0.72	1.05	348.33	132.77
average				7.93	2.30	3.04	24.01	3.01	2.26	75.02	50.64
2s	10	23	2430	0.48	0.15	0.17	0.43	0.13	0.16	8.73	8.4
3s	19	62	2599	0.33	0.08	0.19	0.25	0.06	0.08	0.63	0.63
A1s	19	62	2950	0.27	0.11	0.14	0.47	0.14	0.16	0.56	0.73
A2s	20	53	3423	2.57	0.54	1.39	0.77	0.52	0.58	0.93	0.62
STS2s	30	78	4569	10.12	4.29	5.79	11.85	6.22	7.17	149.61	303.24
STS4s	20	50	9481	13.10	3.12	5.52	15.25	4.21	7.41	7.28	22.74
OF1	10	23	2713	0.07	0.02	0.04	0.05	0.02	0.05	0.25	0.17
OF2	10	24	2515	0.28	0.09	0.15	0.22	0.09	0.07	0.19	0.37
W	19	62	2623	0.75	0.23	0.34	0.52	0.17	0.13	0.14	0.2
CHL1s	30	63	13036	4.30	1.96	2.56	5.15	1.56	2.36	20.38	11.51
CHL2s	10	19	3162	0.18	0.07	0.05	0.17	0.09	0.07	0.53	0.47
A3	20	46	5380	1.78	0.66	0.71	1.87	0.67	0.87	3.94	3.12
A4	19	35	5885	1.58	0.77	0.9	1.85	0.56	0.59	1.53	0.88
A5	20	45	12553	3.97	1.51	1.16	1.53	0.66	0.85	7.37	7.79
CHL5	10	18	363	0.03	0.02	0.03	0.03	0.02	0.03	0.15	0.17
CHL6	30	65	16572	21.50	6.8	8.92	38.52	29.64	17.48	524.16	601.79
CHL7	34	75	16728	54.23	9.28	15.54	181.73	66.16	65.08	601.82	602.35
CU1	25	82	12312	11.78	9.01	4.26	1.70	1.03	2.08	0.48	0.55
CU2	34	90	26100	3.67	2.01	1.99	1.80	0.43	0.7	2.73	3.52
Hchl3s	10	51	11961	312.93	450.7	138.55	13.97	4.01	4.45	35.79	21.89
Hchl4s	10	32	11408	402.13	192.4	372.14	5.62	2.63	2.12	40.9	13.59
Hchl6s	22	60	60170	19.60	4.02	7.04	45.25	15.56	15.7	602.62	94.52
Hchl7s	40	90	62459	168.20	56.9	68.87	751.40	371.25	280.32	601.87	601.65
Hchl8s	10	18	729	0.72	0.16	0.14	0.42	0.16	0.17	0.43	0.44
average				43.11	31.04	26.52	45.03	21.08	17.03	108.88	95.89
ATP30	38	192	137813		608.13	605.84		607.15	608.98		
ATP31	51	258	813748		610.74	611.52		613.78	610.38		
ATP32	56	249	36940		606.24	607.71		606.17	605.16		
ATP33	44	224	233016		608.25	611.61		618.09	612.26		
ATP34	27	130	354962		604.86	603.81		608.24	608.63		
ATP35	29	153	611109		601.92	604.02		611.69	612.52		
ATP36	28	153	129262		604.53	605.19		615.86	617.6		
ATP37	43	222	380592		608.7	611.07		613.76	610.31		
ATP38	40	202	257540		604.9	605.63		608	615.13		
ATP39	33	163	264470		603.05	606.02		616.5	610.83		
ATP40	56	290	63622		608.53	613.21		488.23	616.45		
ATP41	36	177	202305		119.9	414.03		4.76	7.92		
ATP42	59	325	32589		614.45	619.55		106.6	175.83		
ATP43	49	259	208998		414.94	612.32		67.01	73.64		
ATP44	39	196	70940		604.72	606.94		40.36	35.41		
ATP45	33	156	74205		31.75	25.85		1.66	1.44		
ATP46	42	197	146402		42.05	65.08		7.14	13.74		
ATP47	43	204	144317		105.93	442.52		15.72	7.2		
ATP48	34	167	165428		34.99	31.3		5.37	7.38		
ATP49	25	119	206965		602.58	602.57		62.37	51.13		
average					462.06	505.29		345.92	355.10		

Table 4.2: 2D-2CP: medium classes, models M1, M2, and M4, first cut along the second dimension

				M1			M2			M4	
name	m	n	obj	t^{old}	t	t^{lex}	t^{old}	t	t^{lex}	t	t^{lex}
HH	5	18	9246	0.25	0.08	0.19	0.13	0.04	0.06	0.12	0.13
2	10	23	2444	1.33	0.26	0.29	0.42	0.13	0.15	0.39	0.3
3	19	62	1740	1.35	0.61	0.63	0.72	0.24	0.22	0.11	0.14
A1	19	62	1820	1.30	0.53	0.89	0.77	0.23	0.19	0.13	0.19
A2	20	53	2310	1.37	0.29	0.42	0.75	0.22	0.22	0.47	0.61
STS2	30	78	4620	1.25	0.39	0.48	0.65	0.18	0.69	7.04	2.56
STS4	20	50	9468	4.88	1.12	1.46	6.00	2.59	3.38	38.1	20.69
CHL1	30	63	8208	6.03	1.98	3.13	9.80	2.11	2.29	601.86	601.58
CHL2	10	19	2086	0.22	0.09	0.06	0.25	0.06	0.05	0.25	0.14
CW1	25	67	6402	1.00	0.24	0.84	0.57	0.15	0.18	0.36	0.27
CW2	35	63	5159	1.82	0.53	0.72	1.42	0.29	0.37	0.47	0.46
CW3	40	96	5689	7.48	2.47	2.83	1.37	0.33	0.43	0.91	1.52
Hchl2	35	75	9528	93.80	72.04	84.9	1674.82	117.73	52.15	602.32	601.81
Hchl9	35	76	5060	6.13	2.69	2.82	5.12	3.02	2.07	45.09	31.04
average				9.16	5.95	7.12	121.63	9.09	4.46	92.69	90.10
2s	10	23	2450	0.65	0.24	0.3	0.35	0.14	0.14	0.27	0.22
3s	19	62	2623	0.75	0.34	0.28	0.45	0.14	0.2	0.18	0.18
A1s	19	62	2910	1.08	0.26	0.32	0.48	0.24	0.24	0.2	0.25
A2s	20	53	3451	3.08	1.18	1.36	0.73	0.25	0.91	0.51	0.67
STS2s	30	78	4625	4.42	2.51	1.61	1.90	1.36	2.7	5.01	7.13
STS4s	20	50	9481	5.42	1.71	2.18	13.83	4.64	5.64	34.51	17.49
OF1	10	23	2660	0.07	0.01	0.03	0.07	0.04	0.05	0.54	0.47
OF2	10	24	2522	0.20	0.1	0.1	0.15	0.07	0.08	0.26	0.29
W	19	62	2599	0.35	0.09	0.19	0.18	0.06	0.09	0.63	0.62
CHL1s	30	63	12602	11.42	9.31	11.52	80.22	161.66	227.05	601.94	601.72
CHL2s	10	19	3198	0.18	0.09	0.09	0.13	0.09	0.1	0.16	0.16
A3	20	46	5403	1.68	0.8	1.58	3.25	1.22	1.03	1.95	3.44
A4	19	35	5905	1.87	0.53	0.95	2.12	0.33	0.38	13.18	15.95
A5	20	45	12449	8.20	2.21	2.17	3.18	3.69	6.29	601.57	70.21
CHL5	10	18	344	0.05	0.04	0.04	0.05	0.03	0.03	0.15	0.22
CHL6	30	65	16281	21.43	10.39	20.8	720.00	601.88	474.65	602.27	601.6
CHL7	34	75	16602	43.35	17.46	26.07	475.50	282.42	344.08	602.65	601.76
CU1	25	82	12200	448.50	235.33	14.1	1.93	0.62	0.98	2.06	1.97
CU2	34	90	25260	11.02	5.3	6.38	4.40	1.61	2.32	2.99	1.44
Hchl3s	10	51	11829	119.41	321.52	181.8	2.22	0.65	1.11	41.73	4.67
Hchl4s	10	32	11258	3.20	0.62	1.03	1.10	0.31	0.22	25	4.69
Hchl6s	22	60	59853	37.13	9.19	12.29	139.38	52.8	102.02	601.67	132.12
Hchl7s	40	90	62845	58.25	62.87	56.55	907.13	434.39	350.94	602.4	601.73
Hchl8s	10	18	791	0.42	0.11	0.11	0.28	0.06	0.09	1.34	2.4
average				32.59	28.43	14.24	98.29	64.53	63.39	155.97	111.31

Summary

This work is devoted to LP-based branching schemes and heuristics for one-dimensional stock cutting and two-dimensional two-stage constrained cutting. Some industrial technological criteria were considered.

In Chapter 2 we investigated a branch-and-cut-and-price algorithm with branching on variables. For 1D-CSP, benchmarking in new test classes of large problems (with $m \in \{200, 400\}$) with various problem parameters, shows only a few unsolved instances in all classes but one: a class with almost equal product sizes (which is hopefully not practical) could not be solved because of an enumerative procedure for pattern generation. Branching on hyperplanes, e.g., as proposed by F. Vanderbeck or based on the arc flow formulation, may be a solution for this class. A comparison with a preliminary version of an arc flow scheme by C. Alves and J.M. Valério de Carvalho in a selected set of instances shows comparable results. This approach seems to have no difficulties in non-IRUP instances; however, comparison in a broad set of instances is needed because this scheme may have other difficult classes.

For the scheme under investigation, difficult instances have usually an LP value integer or slightly less. The combination of branching on variables and GOMORY mixed-integer cuts helped to reduce the number of nodes and time in some instances. But on average, pure branch-and-price (no cuts) was somewhat faster. There are some very rare instances with a large optimality gap (non-IRUP) which could not be solved in an acceptable time without cuts. But when a sufficient number (say, 40) of cuts were allowed at the root node, they were quickly solved without branching. Depth-first search was, on average, the best enumeration strategy because of a strong LP bound (the MIRUP conjecture).

For 2D-2CP, LP-based column generation with branching was not carried out before. The combination of branching and GOMORY integer cuts was much more effective than either standalone approach because of the complicated behavior of the objective function. A similar situation occurs in 1D-CSP with multiple stock lengths and in 2D two-stage strip packing, which makes possible the advantage of the combined approach as well. Local cuts (valid at a subnode) were particularly essential. Best-first search was the best enumeration strategy. Good solutions

were found very quickly. Comparisons to other algorithms show mostly better solutions and higher optimality rates in large instances. The more balanced scheme of branching on slacks was postponed for future research.

Considering the implementation effort needed and the results in 1D-CSP for the chosen class of cutting planes, our recommendation is to investigate other cuts, formulations, and branching rules, e.g., arising from other formulations.

In Chapter 3 we proposed a new simple model for setup minimization which allows effective integer solution. Comparison to the heuristic KOMBI shows better performance, although with a slightly longer running time; comparison to the previous exact approach of F. Vanderbeck shows only slightly worse solutions but a significantly worse LP bound, because his model is more 'decomposed', it enables the effective application of cutting planes. Diving best-first search was essential. A new sequential heuristic produced high-quality cutting plans under a restricted amount of open stacks. An approach to reduce both setups and open stacks was proposed and refined by problem splitting. Both objectives seemed not to contradict for most problem classes, except for those with all small product sizes (below 40% of the stock length). All these methods were investigated in broad ranges of test classes with industrially relevant problem sizes of up to 150 product types. Then some IP models were proposed for open stacks minimization. They are probably too difficult for modern optimizers.

Chapter 4 investigated some known and new assignment formulations of 2D-2CP. The ILP models of A. Lodi and M. Monaci provided satisfactory solutions and were easy to implement. However, for the (multiobjective) optimization of 1D-CSP, pattern generation seemed to be by far the best approach. To enable its industrial application, which implies easy adaptation to further technological constraints, a standard software system is needed. Currently, the developers of ILOG CPLEX are discussing the option of adding new columns at the nodes of a branch-and-cut tree. This would also necessitate the means to support cut lifting for the new columns. Alternatively, some branch-and-cut-and-price systems exist but they are not so intensively supported and standardized.

Bibliography

[AdC03] C. Alves and J. M. Valério de Carvalho, *A branch-and-price algorithm for integer variable sized bin-packing problems*, Tech. report, Universidade do Minho, Portugal, 2003.

[Bea85] J. E. Beasley, *Algorithms for unconstrained two–dimensional guillotine cutting*, J. Oper. Res. Soc. **36** (1985), no. 4, 297–306.

[Bel00] G. Belov, *A cutting plane algorithm for the one-dimensional cutting stock problem with multiple stock lengths*, Diploma thesis, Dresden University, 2000, in German.

[BHV00] C. Barnhart, C. A. Hane, and P. H. Vance, *Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems*, Oper. Res. **48** (2000), 318–326.

[BP02] E. Balas and M. Perregaard, *Lift-and-project for mixed 0-1 programming: recent progress*, Discrete Applied Mathematics **123** (2002), no. 1–3, 129–154.

[BS02] G. Belov and G. Scheithauer, *A cutting plane algorithm for the one-dimensional cutting stock problem with multiple stock lengths*, European Journal of Operational Research **141** (2002), no. 2, 274–294, Special issue on cutting and packing.

[BS03] ———, *A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting*, Technical report, Dresden University, 2003, URL: `www.math.tu-dresden.de/~capad`.

[CJP83] H. Crowder, E. Johnson, and M. W. Padberg, *Solving large-scale zero-one linear programming problems*, Operations Research **31** (1983), 803–834.

[CLSS87] F. Chauny, R. Loulou, S. Sadones, and F. Soumis, *A two–phase heuristic for strip packing: Algorithm and probabilistic analysis*, OR Letters **6** (1987), no. 1, 25–33.

[CMLW99] C. Cordier, H. Marchand, R. Laundy, and L. A. Wolsey, *bc-opt: A branch-and-cut code for mixed integer programs*, Mathematical Programming **86** (1999), no. 2, 335–354.

[cpl01] *ILOG optimization suite. Delivering a competitive advantage*, White paper, ILOG Corporation, 2001, URL: `http://www.ilog.com`.

[dC98] J. M. Valério de Carvalho, *Exact solution of cutting stock problems using column generation and branch-and-bound*, International Transactions in Operational Research **5** (1998), 35–44.

[dC02] ———, *LP models for bin-packing and cutting stock problems*, European Journal of Operational Research **141** (2002), no. 2, 253–273.

[DDI+98] G. Desaulniers, J. Desrosiers, I. Ioachim, M. M. Solomon, F. Soumis, and D. Villeneuve, *A unified framework for deterministic time constrained routing and crew scheduling problems*, Fleet management and logistics (T. G. Crainic and G. Laporte, eds.), Kluwer Academic Publishers, Norwell, MA, 1998, pp. 57–93.

[DF92] H. Dyckhoff and U. Finke, *Cutting and packing in production and distribution*, Physica Verlag, Heidelberg, 1992.

[DP03] Z. Degraeve and M. Peeters, *Optimal integer solutions to industrial cutting stock problems: Part 2, benchmark results*, INFORMS Journal on Computing **15** (2003), 58–81.

[DSD84] J. Desrosiers, F. Soumis, and M. Desrochers, *Routing with time windows by column generation*, Networks **14** (1984), 545–565.

[DST97] H. Dyckhoff, G. Scheithauer, and J. Terno, *Cutting and packing*, Annotated Bibliographies in Combinatorial Optimization (M. Dell'Amico, F. Maffioli, and S. Martello, eds.), John Wiley & Sons, Chichester, 1997, pp. 393–412.

[Dyc81] H. Dyckhoff, *A new linear approach to the cutting stock problem*, Oper. Res. **29** (1981), no. 6, 1092–1104.

[Dyc90] H. Dyckhoff, *A typology of cutting and packing problems*, European Journal of Operational Research **44** (1990), 145–160.

[Fal96] E. Falkenauer, *A hybrid grouping genetic algorithm for bin packing*, Journal of Heuristics **2** (1996), no. 1, 5–30.

[Far90] A.A. Farley, *A note on bounding a class of linear programming problems, including cutting stock problems*, Oper. Res. **38** (1990), no. 5, 922–923.

[FW00] H. Foerster and G. Waescher, *Pattern reduction in one-dimensional cutting stock problems*, International Journal of Production Research **38** (2000), no. 7, 1657–1676.

[GG61] P. C. Gilmore and R. E. Gomory, *A linear programming approach to the cutting-stock problem*, Operations Research **9** (1961), 849–859.

[GG63] ———, *A linear programming approach to the cutting-stock problem (Part II)*, Operations Research **11** (1963), 863–887.

[GG65] ———, *Multistage cutting stock problems of two and more dimensions*, Oper. Res. **13** (1965), 94–120.

[GG66] ———, *The theory and computation of knapsack functions*, Oper. Res. **14** (1966), 1045–1075.

[GS01] C. Gomes and B. Selman, *Algorithm portfolios*, Artificial Intelligence **126** (2001), 43–62, URL: `www.ilog.com` (technical papers).

[Hae75] R. W. Haessler, *Controlling cutting pattern changes in one-dimensional trim problems*, Operations Research **23** (1975), 483–493.

[Hif99] M. Hifi, *The strip cutting/packing problem: incremental substrip algorithms-based heuristics*, Working paper, 1999.

[Hif01] ———, *Exact algorithms for large-scale unconstrained two and three staged cutting problems*, Computational Optimization and Applications **18** (2001), no. 1, 63–88.

[HM03] M. Hifi and R. M'Hallah, *Strip generation algorithms for constrained two-dimensional two-staged cutting problems*, Working paper, 2003.

[Hol02] O. Holthaus, *Decomposition approaches for solving the integer one-dimensional cutting stock problem with different types of standard lengths*, European Journal of Operational Research **141** (2002), 295–312.

[HR01] M. Hifi and C. Roucairol, *Approximate and exact algorithm for constrained (un)weighted two-dimensional two-staged cutting stock problems*, Journal of combinatorial optimization **5** (2001), 465–494.

[ilo02] *ILOG Solver 5.1*, User's manual, ILOG Corporation, 2002, URL: `http://www.ilog.com`.

[JRZ99] M. P. Johnson, C. Rennick, and E. Zak, *One-dimensional cutting stock problem in just-in-time environment*, Pesquisa Operacional **19** (1999), no. 1, 145–158.

[JT00] M. Jünger and S. Thienel, *The ABACUS system for branch-and-cut-and-price algorithms in integer programming and combinatorial optimization*, Software: Practice and Experience **30** (2000), no. 11, 1325–1352.

[Kan60] L. V. Kantorovich, *Mathematical methods of organizing and planning production*, Management Science **6** (1960), 363–422, (in Russian 1939).

[Kar02] V. Kartak, *A grouping method and problem reduction by column generation*, Draft paper, 2002, Produced in 2001/2002 during a postgraduate scholarship in Dresden.

[Kup98] J. Kupke, *Lösung von ganzzahligen Verschnittproblemen mit Branch-and-Price*, Diplomarbeit, Universität zu Köln, 1998.

[LD02] M. E. Lübbecke and J. Desrosiers, *Selected topics in column generation*, Les Cahiers du GERAD G-2002-64, Montréal, Canada, 2002.

[LL02] A. N. Letchford and A. Lodi, *Strengthening Chvátal-Gomory cuts and Gomory fractional cuts*, Operations Research Letters **30** (2002), no. 2, 74–82.

[LM02] A. Lodi and M. Monaci, *Integer linear programming models for 2-staged two-dimensional knapsack problems*, Mathematical Programming, Ser. B **94** (2002), 257–278.

[LS99] J. T. Linderoth and M. W. P. Savelsbergh, *A computational study of strategies for mixed integer programming*, INFORMS Journal on Computing **11** (1999), 173–187.

[MA92] R. Morabito and M. Arenales, *Staged and constrained two-dimensional guillotine cutting problems: A new approach*, Tech. report, Notas do ICMSC 126, Universidale de Sao Paulo, Sao Carlos, S.P., Brazil, 1992.

[Mar01] A. Martin, *General mixed-integer programming: computational issues for branch-and-cut algorithms*, Computat. Comb. Optimization, LNCS (M. Jünger and D. Naddef, eds.), vol. 2241, 2001, pp. 1–25.

[MBKM01] E. A. Mukhacheva, G. Belov, V. Kartak, and A. S. Mukhacheva, *One-dimensional cutting stock problem: Numerical experiments with the sequential value correction method and a modified branch-and-bound method*, Pesquisa Operacional **20** (2001), no. 2, 153–168.

[MPT99] S. Martello, D. Pisinger, and P. Toth, *Dynamic programming and strong bounds for the 0-1 knapsack problem*, Management Science **45** (1999), 414–423.

[MT90] S. Martello and P. Toth, *Knapsack problems – algorithms and computer implementations*, John Wiley and Sons, Chichester et al., 1990.

[MZ93] E. A. Mukhacheva and V. A. Zalgaller, *Linear programming for cutting problems*, International Journal of Software Engineering and Knowledge Engineering **3** (1993), no. 4, 463–477.

[NST99] C. Nitsche, G. Scheithauer, and J. Terno, *Tighter relaxations for the cutting stock problem*, European Journal of Operational Research **112** (1999), 654–663.

[NW88] G. L. Nemhauser and L. A. Wolsey, *Integer and combinatorial optimization*, John Wiley and Sons, New York, 1988.

[Pee02] M. Peeters, *One-dimensional cutting and packing: new problems and algorithms*, Ph.D. thesis, Katholieke Universiteit Leuven, Belgium, 2002.

[PPSG00] V. Parada, R. Palma, D. Sales, and A. Gómes, *A comparative numerical analysis for the guillotine two-dimensional cutting problem*, Annals of Operations Research **96** (2000), 245–254.

[RS02] J. Rietz and G. Scheithauer, *Tighter bounds for the gap and non-IRUP constructions in the one-dimensional cutting stock problem*, Optimization **51** (2002), no. 6, 927–963.

[Sch02a] J. E. Schoenfield, *Fast, exact solution of open bin packing problems without linear programming*, Draft, US Army Space & Missile Defense Command, Huntsville, Alabama, USA, 2002.

[Sch02b] H. Schreck, *A sequence problem in practical trim optimization*, Proceedings of the sixteenth triennial conference of the International Federation of Operational Research Societies, 2002, p. 121.

[SKJ97] A. Scholl, R. Klein, and C. Jürgens, *BISON: A fast hybrid procedure for exactly solving the one-dimensional bin-packing problem*, Computers and Operations Research **24** (1997), no. 7, 627–645.

[ST95a] G. Scheithauer and J. Terno, *A branch-and-bound algorithm for solving one-dimensional cutting stock problems exactly*, Applicationes Mathematicae **23** (1995), no. 2, 151–167.

[ST95b] ———, *The modified integer round-up property of the one-dimensional cutting stock problem*, European Journal of Operational Research **84** (1995), 562–571.

[Sta88] H. Stadtler, *A comparison of two optimization procedures for 1– and 1,5–dimensional cutting stock problems*, OR Spektrum **10** (1988), 97–111.

[STMB01] G. Scheithauer, J. Terno, A. Müller, and G. Belov, *Solving one-dimensional cutting stock problems exactly with a cutting plane algorithm*, Journal of the Operational Research Society **52** (2001), 1390–1401.

[SW99] P. Schwerin and G. Wäscher, *A new lower bound for the bin-packing problem and its integration into MTP*, Pesquisa Operacional **19** (1999), no. 2, 111–130.

[TLS87] J. Terno, R. Lindemann, and G. Scheithauer, *Zuschnittprobleme und ihre praktische Lösung*, Verlag Harri Deutsch, Thun und Frankfurt/Main, 1987.

[UYI03] S. Umetani, M. Yagiura, and T. Ibaraki, *One dimensional cutting stock problem to minimize the number of different patterns*, European Journal of Operational Research **146** (2003), no. 2, 388–402.

[Vah96] R. Vahrenkamp, *Random search in the one-dimensional cutting stock problem*, European Journal of Operational Research **95** (1996), 191–200.

[Van99] F. Vanderbeck, *Computational study of a column generation algorithm for bin packing and cutting stock problems*, Mathematical Programming, Ser. A **86** (1999), no. 3, 565–594.

[Van00a] ———, *Exact algorithm for minimizing the number of setups in the one-dimensional cutting stock problem*, Operations Research **48** (2000), no. 6, 915–926.

[Van00b] ———, *On Dantzig-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm*, Operations Research **48** (2000), no. 1, 111–128.

[Van01] ———, *A nested decomposition approach to a 3-stage 2-dimensional cutting stock problem*, Management Science **47** (2001), 864–879.

[VW96] F. Vanderbeck and L. A. Wolsey, *An exact algorithm for IP column generation*, Operations Research Letters **19** (1996), 151–159.

[Wan83] P. Y. Wang, *Two algorithms for constrained two-dimensional cutting stock problems*, Oper. Res. **31** (1983), 573–586.

[WG96] G. Wäscher and T. Gau, *Heuristics for the one-dimensional cutting stock problem: A computational study*, OR Spektrum **18** (1996), 131–144.

[Wol98] L. A. Wolsey, *Integer programming*, Wiley Interscience Series in Discrete Mathematics and Optimization, John Wiley and Sons, Chichester, 1998.

[YBS99] H. H. Yanasse, J. C. Becceneri, and N. Y. Soma, *Bounds for a problem of sequencing patterns*, Pesquisa Operacional **19** (1999), no. 1, 249–278.

[Yue91a] M. Yue, *A simple proof of the inequality* $FFD(L) \leq 11/9OPT(L) + 1$, $\forall L$, *for the* FFD *bin-packing algorithm*, Acta Math. App. Sinica **7** (1991), 321–331.

[Yue91b] B. J. Yuen, *Heuristics for sequencing cutting patterns*, European Journal of Operational Research **55** (1991), 183–190.

[Zak02] E. Zak, *A counterpart of one-dimensional stock cutting: skiving stock problem*, Proceedings of the sixteenth triennial conference of the International Federation of Operational Research Societies, 2002, p. 121.

[ZB03] C. Zelle and R. E. Burkard, *A local search heuristic for the reel cutting problem in paper production*, SFB Report No. 257, Institute of Mathematics B, Technical University Graz, 2003.

List of Tables

Appendix

A.1 Example of an Idea Leading to Branch-and-Price

Sometimes it is possible to prove that an instance is non-IRUP in the following simple way [Kar02]. Let the LP value be z^1. Suppose we investigate the domain of optimum LP solutions by setting $x_t = 0$ for some basic variable x_t and resolving the LP which gives the LP value z^t. If $\lceil z^1 \rceil < \lceil z^t \rceil$ then column t *must* be in an integer optimum if the instance is IRUP. Thus, we can reduce the problem: $b' = b - a^t$ which gives a new LP value ${}^t z^1$. Now, if $\lceil {}^t z^1 + 1 \rceil > \lceil z^1 \rceil$ then the problem is non-IRUP.

This looks complicated at first sight but in fact it is branching on variable x_t: the first subproblem has $x_t \leq 0$ and the second has $x_t \geq 1$. If both subproblems have a local lower bound greater than the root LP bound $\lceil z^1 \rceil$ then it is also the global lower bound.

To compute the local bound in each subproblem, new columns may be needed. Thus, this scheme is branch-and-price.

A.2 Example of a Solution with 4 Open Stacks

Solutions of the first instance from class 4: the initial LP solution, a material-minimum solution, an SVC solution with at most 4 open stacks (Section 3.2).

```
The problem: L=10000 m=50  Piece type numbering from 0
6928:11 6923:87 6753:4 6623:21 6241:21 6131:33 6104:65 6102:75
5958:25 5678:87 5653:18 5245:6 4813:20 4809:15 4741:79 4415:11
4396:45 4305:6 4286:2 4239:95 4141:69 3910:49 3834:10 3820:74
3760:66 3661:87 3658:28 3550:16 3512:18 3465:30 3408:83 3353:34
3275:66 2922:93 2887:89 2848:5 2599:99 2392:12 2195:57 2114:93
1755:67 1618:33 1594:80 1506:22 1090:29 1060:66 965:75 551:52
438:20 204:89
```

File: m50l100l7000L10000b1b100f0r0-5 Instance No. 1 Wed Jul 2 10:03:51 2003
The initial LP solution: time=0.09 lpval=807.788 lpbnd=808

```
5.2: 36:3 38:1                  10: 8:1 22:1 49:1
8.12257: 30:2 39:1 45:1         18: 10:1 32:1 45:1
9: 28:2 33:1                    19: 21:1 29:1 36:1
0.5: 24:2 43:1 46:1             8.0837: 6:1 25:1 49:1
13: 24:1 31:1 34:1              45: 16:1 30:1 38:1
4.79221: 21:2 39:1              20: 1:1 36:1 48:1
2.9: 19:2 43:1                  2: 9:1 18:1
29.2: 19:1 20:1 42:1            9.3246: 26:1 39:3
6: 11:1 14:1                    1.75486: 30:1 32:2
6: 9:1 17:1                     13.5927: 1:1 44:1 46:2
27: 9:1 19:1                    21: 3:1 31:1
25.6017: 7:1 23:1               21.1246: 5:1 25:1 49:1
31.5097: 7:1 34:1 46:1          33: 19:1 20:1 41:1
41.1114: 6:1 39:1 40:1          16: 4:1 27:1 49:1
5: 4:1 25:1                     4: 2:1 44:2 45:1
46.1031: 1:1 33:1               13.0083: 13:1 21:1 45:1 49:1
11: 0:1 33:1                    11.092: 8:1 33:1 45:1
1.9917: 13:1 25:1 43:1          2.30415: 1:1 43:2
3.908: 8:1 23:1 49:1            15.8049: 6:1 33:1 46:1
11.8754: 5:1 26:1 49:1          22.2: 14:1 36:2
50.8: 14:1 25:1 42:1            5.88856: 7:1 40:1 45:2
20: 12:1 30:1 40:1              12: 7:1 37:1 43:1
5: 1:1 35:1 49:1                11: 15:1 29:1 39:1
52: 9:1 24:1 47:1               6.8: 20:1 26:1 38:1
44.4903: 23:1 32:1 34:1         3.70364: 21:2 44:2
```

File: m50l100l7000L10000b1b100f0r0-5 Instance No. 1 Wed Jul 2 10:03:52 2003
LP0 Time: 0.09 IPTime: 0.09 The best integer solution (808):

```
5: 36:3 38:1
8: 30:2 39:1 45:1
9: 28:2 33:1
13: 24:1 31:1 34:1
5: 21:2 39:1
3: 19:2 43:1
29: 19:1 20:1 42:1
6: 11:1 14:1
6: 9:1 17:1
27: 9:1 19:1
26: 7:1 23:1
31: 7:1 34:1 46:1
41: 6:1 39:1 40:1
5: 4:1 25:1
46: 1:1 33:1
11: 0:1 33:1
2: 13:1 25:1 43:1
4: 8:1 23:1 49:1
12: 5:1 26:1 49:1
51: 14:1 25:1 42:1
20: 12:1 30:1 40:1
5: 1:1 35:1 49:1
52: 9:1 24:1 47:1
44: 23:1 32:1 34:1
10: 8:1 22:1 49:1
18: 10:1 32:1 45:1
19: 21:1 29:1 36:1
8: 6:1 25:1 49:1
45: 16:1 30:1 38:1
20: 1:1 36:1 48:1
2: 9:1 18:1
9: 26:1 39:3
2: 30:1 32:2
14: 1:1 44:1 46:2
21: 3:1 31:1
21: 5:1 25:1 49:1
33: 19:1 20:1 41:1
16: 4:1 27:1 49:1
4: 2:1 44:2 45:1
13: 13:1 21:1 45:1 49:1
11: 8:1 33:1 45:1
2: 1:1 43:2
16: 6:1 33:1 46:1
22: 14:1 36:2
6: 7:1 40:1 45:2
12: 7:1 37:1 43:1
11: 15:1 29:1 39:1
7: 20:1 26:1 38:1
3: 21:2 44:2
1: 24:1 36:1 39:1 43:1
1: 21:1 34:1 44:1
```

SVC: max. open stacks 4.
Iter 146. Sol. found: (4 51[51] 811)

```
25: 8/1 29/1 47/1
5: 6/1 29/1 49/2
60: 6/1 25/1 49/1
19: 5/1 25/1 49/1
14: 5/1 23/1
60: 7/1 23/1
8: 4/1 25/1
13: 4/1 42/2 47/1
10: 7/1 22/1
4: 2/1 42/1 47/3
1: 9/1 42/2 47/2
6: 9/1 17/1
2: 9/1 18/1
78: 9/1 19/1
5: 7/1 24/1
10: 24/2 42/1 48/2
12: 24/2 37/1
17: 10/1 19/1
1: 10/1 42/2 44/1
16: 24/1 27/1 42/1 44/1
12: 16/2 44/1
1: 16/1 24/1 40/1
20: 3/1 40/1 42/1
1: 3/1 31/1
20: 16/1 31/1 38/1
11: 15/1 31/1 38/1
8: 30/1 38/3
1: 38/2 40/3
20: 12/1 30/1 40/1
15: 13/1 30/1 40/1
2: 30/1 31/1 41/2
19: 30/2 45/3
9: 26/2 41/1 45/1
10: 26/1 39/3
20: 20/1 39/2 41/1
44: 20/1 34/2
1: 34/1 40/4
5: 20/1 33/2
11: 0/1 33/1
72: 1/1 33/1
11: 1/1 43/2
23: 21/2 39/1
4: 1/1 35/1
1: 28/2 35/1
8: 28/2 46/3
3: 21/1 32/1 40/1 46/1
24: 14/1 32/1 46/2
6: 11/1 14/1
49: 14/1 36/2
13: 32/3
1: 36/1 40/1
```

Versicherung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt.

I affirm that I have written this dissertation without any inadmissible help from any third person and without recourse to any aids other than cited; all sources are clearly referenced. The dissertation has never been submitted in this or similar form before, neither in Germany nor in any foreign country.

Die vorgelegte Dissertation habe ich am Institut für Numerische Mathematik der Technischen Universität Dresden unter der wissenschaftlichen Betreuung von Herrn Dr.rer.nat. Guntram Scheithauer angefertigt.

I have written this dissertation at the Institute for Numerical Mathematics, Technical University Dresden, under the scientific supervision of Dr.rer.nat. Guntram Scheithauer.

Gleb Belov

Dresden, den 8. September 2003