Noname manuscript No.
(will be inserted by the editor)

Solving Difficult Optimization Problems

The Art of Modeling

Josef Kallrath[1,2]

[1] BASF Aktiengesellschaft, GVC/S (Scientific Computing) - B009, D-67056 Ludwigshafen, Germany (e-mail: josef.kallrath@web.de)
[2] University of Florida, Astronomy Dept., Gainesville, 32661 FL, USA (e-mail: kallrath@astro.ufl.edu)

The date of receipt and acceptance will be inserted by the editor

1 Introduction

We define difficult optimization problems as problems which cannot be solved to optimality or to any guaranteed bound by any standard solver within a reasonable time limit. The problem class we have in mind are mixed integer programming (MIP) problems. Optimization and especially mixed integer optimization is often appropriate and frequently used to model real world optimization problems. While its origin started in the 1950s models became larger and more complicated.

A reasonable general framework is mixed integer nonlinear programming (MINLP) problems. They are specified by the augmented vector $\mathbf{x}_{\oplus}^{\mathrm{T}} = \mathbf{x}^{\mathrm{T}} \oplus \mathbf{y}^{\mathrm{T}}$ established by the vectors $\mathbf{x}^{\mathrm{T}} = (x_1, ..., x_{n_c})$ and $\mathbf{y}^{\mathrm{T}} = (y_1, ..., y_{n_d})$ of n_c continuous and n_d discrete variables, an objective function $f(\mathbf{x}, \mathbf{y})$, n_e equality constraints $\mathbf{h}(\mathbf{x}, \mathbf{y})$ and n_i inequality constraints $\mathbf{g}(\mathbf{x}, \mathbf{y})$. The problem

$$\min \left\{ f(\mathbf{x}, \mathbf{y}) \left| \begin{array}{l} \mathbf{h}(\mathbf{x}, \mathbf{y}) = 0 \;\; \mathbf{h} : X \times U \to \mathbb{R}^{n_e} \;\; \mathbf{x} \in X \subseteq \mathbb{R}^{n_c} \\ \mathbf{g}(\mathbf{x}, \mathbf{y}) \geq 0, \, \mathbf{g} : X \times U \to \mathbb{R}^{n_i}, \, \mathbf{y} \in U \subseteq \mathbb{Z}^{n_d} \end{array} \right. \right\} \quad (1.1)$$

is called *Mixed Integer Nonlinear Programming* (MINLP) problem, if at least one of the functions $f(\mathbf{x}, \mathbf{y})$, $\mathbf{g}(\mathbf{x}, \mathbf{y})$ or $\mathbf{h}(\mathbf{x}, \mathbf{y})$ is nonlinear. The vector inequality, $\mathbf{g}(\mathbf{x}, \mathbf{y}) \geq 0$, is to be read component-wise. Any vector $\mathbf{x}_{\oplus}^{\mathrm{T}}$ satisfying the constraints of (1.1) is called a *feasible point* of (1.1). Any feasible point, whose objective function value is less or equal than that of all other feasible points is called an *optimal solution*. From this definition it follows that the problem might not have a unique optimal solution.

Depending on the functions $f(\mathbf{x}, \mathbf{y})$, $\mathbf{g}(\mathbf{x}, \mathbf{y})$, and $\mathbf{h}(\mathbf{x}, \mathbf{y})$ in (1.1) we get the following structured problems known as

acronym	*type of optimization*	$f(\mathbf{x},\mathbf{y})$	$\mathbf{h}(\mathbf{x},\mathbf{y})$	$\mathbf{g}(\mathbf{x},\mathbf{y})$	n_d
LP	Linear Programming	$\mathbf{c}^{\mathrm{T}}\mathbf{x}$	$\mathsf{A}\mathbf{x}-\mathbf{b}$	$\mathbf{x}$	0
QP	Quadratic Programming	$\mathbf{x}^{\mathrm{T}}Q\mathbf{x}+\mathbf{c}^{\mathrm{T}}\mathbf{x}$	$\mathsf{A}\mathbf{x}-\mathbf{b}$	$\mathbf{x}$	0
NLP	Nonlinear Programming				0
MILP	Mixed Integer LP	$\mathbf{c}^{\mathrm{T}}\mathbf{x}_{\oplus}$	$\mathsf{A}\mathbf{x}_{\oplus}-\mathbf{b}$	$\mathbf{x}_{\oplus}$	≥ 1
MIQP	Mixed Integer QP	$\mathbf{x}_{\oplus}^{\mathrm{T}}\mathsf{Q}\mathbf{x}_{\oplus}+\mathbf{c}^{\mathrm{T}}\mathbf{x}_{\oplus}$	$\mathsf{A}\mathbf{x}_{\oplus}-\mathbf{b}$	$\mathbf{x}_{\oplus}$	≥ 1
MINLP	Mixed Integer NLP				≥ 1

with a matrix A of m rows and n columns, i.e., $\mathsf{A}\in\mathcal{M}(m\times n,\mathbb{R})$, $\mathbf{b}\in\mathbb{R}^m$, $\mathbf{c}\in\mathbb{R}^n$, and $n=n_c+n_d$. Real-world problems lead much more frequently to LP and MILP than to NLP or MINLP problems. QP refers to quadratic programming problems. They have a quadratic objective function but only linear constraints. QP and MIQP problems often occur in applications of the financial service industries.

While LP problems as described in Pardalos (2001, [31]) or Anstreicher (2001, [1]) can be solved relatively easily (the number of iterations, and thus the effort to solve LP problems with m constraints grows approximately linearly in m), the computational complexity of MILP and MINLP grows exponentially with n_d but depends strongly on the structure of the problem. Numerical methods to solve NLP problems work iteratively and the computational problems are related to questions of convergence, getting stuck in bad local optima and availability of good initial solutions. Global optimization techniques can be applied to both NLP and MINLP problems and its complexity increases exponentially in the number of all variables entering nonlinearly into the model.

While the word *optimization*, in nontechnical or colloquial language, is often used in the sense of *improving*, the mathematical optimization community sticks to the original meaning of the word related to finding the *best value* either globally or at least in a local neighborhood. For an algorithm being considered as a (mathematical, strict or exact) optimization algorithm in the mathematical optimization community there is consensus that such an algorithm computes feasible points proven globally (or locally) optimal for linear (nonlinear) optimization problems. Note that this is a definition of a mathematical optimization algorithm and *not* a statement saying that computing a local optimum is sufficient for nonlinear optimization problems. In the context of mixed integer linear problems an optimization algorithm (Grötschel, 2004, [12] and Grötschel, 2005, [13]) is expected to compute a proven optimal solution or to generate feasible points and, for a maximization problem, to derive a reasonably tight, non-trivial upper bound. The quality of such bounds are quantified by the integrality gap – the difference between upper and lower bound. It depends on the problem, the purpose of the model and also the accuracy of the data what one considers to be a good quality solution. A few percent, say, 2 to 3%, might be acceptable for the example discussed Kallrath (2007, Encyclopedia: Planning). How-

ever, discussion based on percentage gaps become complicated when the objective function includes penalty terms containing coefficients without a strict economic interpretation. In such cases scaling is problematic. Goal programming as discussed in Kallrath & Maindl (2006, [23], pp. 294) might help in such situations to avoid penalty terms in the model. The problem is first solved with respect to the highest priority goal, then one cares about the next level goal, and so on.

For practical purposes it is also relevant to observe that solving mixed integer linear problems and the problem of finding appropriate bounds is often $\mathcal{NP}$ complete, which makes these problems hard to solve. A consequence of this structural property is that these problems scale badly. If the problem can be solved to optimality for a given instance, this might not be so if the size is increased slightly. While tailor-made optimization algorithm such as column generation, Branch&Price techniques can often cope with this situation for individual problems, this is very difficult for standard software.

We define difficult optimization problems as problems which cannot be solved to optimality or within a reasonable integrality gap by any standard MIP solver within a reasonable time limit. Problem structure, size, or both could lead to such behavior. However, in many cases these problems (typically mixed integer programming or non-convex optimization problems fall into this class) can be solved if they are individually treated and we resort to the art of modeling.

The art of modeling includes choosing the right level of detail implemented in the model. On the one hand, this needs to satisfy the expectations of the owner of the real world problem. On the other hand we are limited by the available computational resources. We give reasons why strict optimality or at least safe bounds are essential when dealing with real world problems and why we do not accept methods which do not generate both upper and lower bounds.

Mapping the reality also enforces us to discuss whether deterministic optimization is sufficient or whether we need to resort to optimization under uncertainty. Another issue is to check whether one objective function suffices or whether multiple criteria optimization techniques need to be applied.

Instead of solving such difficult problems directly as, for example, a stand alone mixed integer linear programming problem we discuss how the problems can be solved equivalently by solving a sequence of models.

Efficient approaches are:

- column generation with a master and subproblem structure,
- branch-and-price,
- exploiting a decomposition structure with a rolling time horizon,
- exploiting auxiliary problems to generate safe bounds for the original problem which then makes the original problems more tractable, or
- exhaustion approaches,

- hybrid methods, i.e., constructive heuristics and local search on subsets of the difficult discrete variables leaving the remaining variables and constraints in tractable MILP or MINLP problems which can be solved.

We illustrate various ideas using real world planning, scheduling and cutting stock problems.

1.1 Models and the Art of Modeling

We are here concerned with two aspects of modeling and models. The first one is to obtain a reasonable representation of the reality and mapping it onto a mathematical model, i.e., an optimization problem in the form of (1.1). The second one is to reformulate the model or problem in such equivalent forms which makes it numerically tractable.

1.1.1 Models The terms *modeling* or *model building* are derived from the word *model*. Its ethymological roots are the Latin word *modellus* (scale, [diminutiv of modus, measure]) and what was to be in the 16th century the new word *modello*. Nowadays, in scientific context the term is used to refer to a simplified, abstract or well structured part of the reality one is interested in. The idea itself and the associated concept is, however, much older. The classical geometry and especially Phythagoras around 600 B.C. distinguish between wheel and circle, between field and rectangle. Around 1100 D.C. a wooden model of the later Speyer cathedral was produced; the model served to build the real cathedral. Astrolabs and celestial globes have been used as models to visualize the movement of the moon, planets and stars on the celestial sphere and to compute the times of rises and settings. Until the 19th century mechanical models were understood as pictures of the reality. Following the principals of classical mechanics the key idea was to reduce all phenomena to the movement of small particles. Nowadays, in physics and other mathematical sciences one will talk about models if

- one, for reasons of simplifications, restricts oneself to certain aspects of the problem (*example*: if we consider the movement of the planets, in a first approximation the planets are treated as point masses),
- one, for reasons of didactic presentation, develops a simplified picture for the more complicated reality (*example*: the planetary model is used to explain the situation inside the atoms),
- one uses the properties in one area to study the situation in an analogue problem.

A model is referred to as a mathematical model of a process or a problem if it contains the typical mathematical objects (variables, terms, relations). Thus, a (mathematical) model represents a real world problem in the language of mathematics using mathematical symbols, variables, equations, inequalities and other relations.

It is very important when building a model to define and state precisely the purpose of the model. In science, we often encounter epistemological arguments. In engineering, a model might be used to construct some machines. In operations research and optimization, models are often used to support strategic or operative decisions. All models enable us to

- learn and understand situations which do not allow easy access (very slow or fast processes, processes involving a very small or very large region);
- avoid difficult, expensive or dangerous experiments; and
- analyze case studies and *What-If-When scenarios*.

Tailored optimization models can be used to support decisions (that is the overall purpose of the model). It is essential to have a clear objective describing what is a good decision. The optimization model should produce, for instance, optimal solutions in the following sense:

- to avoid unwanted side products as much as possible,
- to minimize costs, or
- to maximize profit, earnings before interest and taxes (EBIT), or contribution margin.

The purpose of a model may change over time.

To solve a real-world problem by mathematical optimization, at first we need to represent our problem by a *mathematical model*, that is, a set of mathematical relationships (e.g., equalities, inequalities, logical conditions) representing an abstraction of our real-world problem. This translation is part of the model building phase (which is part of the whole modeling process), and is not trivial at all because there is nothing we could consider an exact model. Each model is an acceptable candidate as long as it fulfills its purpose and approximates the real world accurately enough. Usually, a model in mathematical optimization consists of four key objects:

- data, also called the *constants* of a model,
- variables (continuous, semi-continuous, binary, integer), also called decision variables,
- constraints (equalities, inequalities), also called *restrictions*, and
- the objective function (sometimes even several of them).

The data may represent cost or demands, fixed operation conditions of a reactor, capacities of plants and so on. The variables represent the degrees of freedom, i.e., what we want to decide: How much of a certain product is to be produced, whether a depot is closed or not, or how much material will we store in the inventory for later use. Classical optimization (calculus, variational calculus, optimal control) treats those cases in which the variables represent continuous degrees of freedom, e.g., the temperature in a chemical reactor or the amount of a product to be produced. Mixed integer optimization involves variables restricted to integer values, for example

counts (numbers of containers, ships), decisions (yes-no), or logical relations (if product A is produced then product B also needs to be produced). The constraints can be a wide range of mathematical relationships: algebraic, analytic, differential or integral. They may represent mass balances, quality relations, capacity limits, and so on. The objective function expresses our goal: minimize costs, maximize utilization rate, minimize waste, and so on. Mathematical models for optimization usually lead to structured problems such as:

- *linear programming* (LP) problems,
- *mixed integer linear programming* (MILP) problems,
- *quadratic* (QP), and *mixed-integer quadratic programming* (MIQP)
- *nonlinear programming* (NLP) problems, and
- *mixed integer nonlinear programming* (MINLP) problems.

1.1.2 The Art of Modeling How do we get from a given problem to its mathematical representation? This is a difficult, non-unique process. It is a compromise between the degree of details required to model the problem and the complexity which is tractable. However, simplifications should not only seen as an unavoidable evil. They could be useful to develop understanding or serve as a platform with the client as the following three examples show:

1. At the beginning of the modeling process it can be useful to start with a "down-scaled" version to develop a feeling for the structure and dependencies of the model. This enable a constructive dialog between the modeler and the client. A vehicle fleet with 100 vehicle and 12 depots could be analyzed with only 10 vehicles and 2 depots to let the *model world* and the *real world* find to each other in a sequence of discussions.
2. In partial or submodels the modeler can develop a deep understanding of certain aspects of the problem which can be relevant to solve the whole problem.
3. Some aspects of the real world problem could be too complicated to model them complete or exactly. During the modeling process it can be clarified, using a smaller version, whether partial aspects of the model could be neglected or whether they are essential.

In any case it is essential that the simplifications are well understood and documented.

2 Tricks of the Trade for Monolithic Models

Using state-of-the art commercial solvers, *e.g.*, `XPressMP` [`XPressMP` is by Dash Optimization, http://www.dashoptimization.com] or `CPLEX` [`CPLEX` is by ILOG, http://www.ilog.com], MILP problems can be solved quite efficiently. In the case of MINLP and using global optimization techniques, the solution efficiency depends strongly on the individual problem and the model

formulation. However, as stressed in [21] for both problem types, MILP and MINLP, it is recommended that the full mathematical structure of a problem is exploited, that appropriate reformulations of models are made and that problem specific valid inequalities or cuts are used. Software packages may also differ with respect to the ability of *pre-solving techniques*, *default-strategies* for the Branch&Bound algorithm, *cut generation* within the Branch&Cut algorithm, and last but not least *diagnosing and tracing infeasibilities* which is an important issue in practice.

Here we collect a list of recommendation tricks which help to improve the solution procedure of monolithic MIP problems, i.e., stand-alone models which are solved by one call to a MILP or MINLP solver. Among them are:

- Using bounds instead of constraints, if the dual values are not necessarily required.
- Apply own pre-solving techniques. Consider for instance a set of inequalities

$$B_{ijk}\delta_{ijk} \leq A_{ijk} \quad ; \quad \forall\{i,j,k\} \tag{2.1}$$

  on binary variables δ_{ijk}. They can be replaced by the bounds

$$\delta_{ijk} = 0 \quad ; \quad \forall\{(i,j,k)\,|A_{ijk} < B_{ijk}\,\}$$

  or, if one does not trust the $<$ in a modeling language the bounds

$$\delta_{ijk} = 0 \quad ; \quad \forall\{(i,j,k)\,|A_{ijk} \leq B_{ijk} - \varepsilon\,\}$$

  where $\varepsilon > 0$ is a small number, say, of the order of 10^{-6}. If $A_{ijk} \geq B_{ijk}$ (2.1) is redundant. Note that due to the fact that we have three indices the number of inequalities can be very large.
- Exploit the *pre-solving techniques* embedded in the solver*; cf.* Martin (2001, [28]).
- Exploiting or eliminating symmetry: Sometimes, symmetry can lead to degenerate scenarios. There are situations, for instance, in scheduling that orders can be allocated to identical production units. Another example is the capacity design problem of a set of production units to be added to a production network. In that case, symmetry can be broken by requesting the capacities of the units are sorted in descending order, i.e., $c_u \geq c_{u+1}$. Menon & Schrage (2002, [29]) exploit symmetry in order allocation for stock cutting in the paper industry; a very pleasant paper to read.
- Use special types of variables for which tailor-made branching rules exists (this applies to semi-continuous and partial-integer variables as well as special ordered sets).
- Experiment with the various *strategies* offered by the commercial Branch&Bound solvers for the Branch&Bound algorithm
- Experiment with the *cut generation* within the commercial Branch&Cut algorithm among them Gomory cuts, knapsack cuts or flow cuts; cf. Martin (2001, [28]).

– Construction of own valid inequalities for certain substructures of problems at hand. Those inequalities may a priori be added to a model, and in the extreme case they would describe the complete convex hull. As an example we consider the mixed-integer inequality

$$x \leq C\lambda \quad , \quad 0 \leq x \leq X \quad ; \quad x \in \mathbb{R}_0^+ \quad , \quad \lambda \in \mathbb{N} \tag{2.2}$$

which has the valid inequality

$$x \leq X - G(K-\lambda) \quad \text{where } K := \left\lceil \frac{X}{C} \right\rceil \quad \text{and} \quad G := X - C\,(K-1) \tag{2.3}$$

This valid inequality (2.3) is the more useful, the more K and X/C deviate. A special case arising often is the situation $\lambda \in \{0,1\}$. Another example, taken from ([39], p. 129) is

$$A_1\alpha_1 + A_2\alpha_2 \leq B + x \qquad x \in \mathbb{R}_0^+ \qquad \alpha_1, \alpha_2 \in \mathbb{N} \tag{2.4}$$

which for $B \notin \mathbb{N}$ leads to the valid inequality

$$\lfloor A_1 \rfloor\, \alpha_1 + \left(\lfloor A_2 \rfloor\, \alpha_2 + \frac{f_2 - f}{1-f} \right) \leq \lfloor B \rfloor + \frac{x}{1-f} \tag{2.5}$$

where the following abbreviations are used:

$$f := B - \lfloor B \rfloor \quad , \quad f_1 := A_1 - \lfloor A_1 \rfloor \quad , \quad f_2 := A_2 - \lfloor A_2 \rfloor \tag{2.6}$$

The dynamic counterpart of valid inequalities added a priori to a model leads to cutting plane algorithms which avoid adding a large number of inequalities a priori to the model (note, this can be equivalent to finding the complete convex hull). Instead, only those useful in the vicinity of the optimal solution are added dynamically. For the topics of valid inequalities and cutting plane algorithms the reader is referred to books by Nemhauser & Wolsey (1988, [30]), Wolsey [39], Pochet & Wolsey (2006, [32]).

– Disaggregation in MINLP problems. Global optimization techniques are often based on convex underestimators. Univariate functions can be treated easier than multivariate terms. Therefore, it helps to represent the bilinear or multilinear terms by their disaggregated equivalences. As an example we consider x_1x_2 with given lower and upper bounds X_i^- and X_i^+ for x_i; $i = 1, 2$. Where ever we encounter x_1x_2 in our model we can replace is by

$$x_1x_2 = \frac{1}{2}(x_{12}^2 - x_1^2 - x_2^2)$$

and

$$x_{12} = x_1 + x_2 \quad .$$

The auxiliary variable is subject to the bounds $X_{12}^- := X_1^- + X_2^-$ and

$$X_{12}^- \leq x_{12} \leq X_{12}^+ \quad , \quad X_{12}^- := X_1^- + X_2^- \quad , \quad X_{12}^+ := X_1^+ + X_2^+ \quad .$$

This formulation has another advantage. It allows us to construct easily a relaxed problem which can be used to derive a useful lower bound. Imagine a problem $\mathcal{P}$ with the inequality

$$x_1 x_2 \leq A \quad . \tag{2.7}$$

Then

$$x_{12}^2 - X_1^- x_1 - X_2^- x_2 \leq 2A \tag{2.8}$$

is a relaxation of $\mathcal{P}$ as each point (x_1, x_2) satisfying (2.7) also fulfills (2.8). Note that an alternative disaggregation avoiding an additional variable is given by

$$x_1 x_2 = \frac{1}{4}\left[(x_1 + x_2)^2 - (x_1 - x_2)^2\right] \quad .$$

However, all of the creative attempts listed above may not suffice to solve the MIP using one monolithic model. That is when we should start looking at solving the problem by a sequence of problems. We have to keep in mind that to solve a MIP problem we need to derive tight lower and upper bounds with the gap between them approaching zero.

3 Decomposition Techniques

Decomposition techniques decompose the problem into a set of smaller problems which can be solved in sequence or in any combination. Ideally, the approach can still compute the global optimium. There are standardized techniques such as Benders Decomposition [*cf.*, Floudas (1995, [9], Chap. 6) or ###Dantzig-Wolfe Decomposition###. But often one should exploit the structure of an optimization to construct tailor-made decompositions. This is outlined in the following subsections.

3.1 Column Generation

The term *column* usually refers to variables in linear programming parlance. In the context of column generation techniques it has wider meaning and stands for any kind of objects involved in an optimization problem. In vehicle routing problems a column might, for instance, as described in subsection represent a subset of orders assigned to a vehicle. In network flow problems a column might represent a feasible path through the network. Finally, in cutting stock problems ([10],[11]) a column represents a pattern to be cut.

The basic idea of column generation is to decompose a given problem into a master and subproblem. Problems which otherwise could be nonlinear

can be completely solved by solving only linear problems. The critical issue is to generate master and subproblems which both can be solved efficiently. One of the most famous example is the elegant column generation approach by Gilmore & Gomory (1961, [10]) for computing the minimal number of rolls to satisfy the requested demand for smaller sized rolls. This problem, if formulated as one monolithic problem, leads to a MINLP problem with a large number of integer variables.

In simple cases, such as the ones described by Schrage (2006, [35], Sect. 11.7), it is possible to generate all columns explicitly, even within a modeling language. Often, the decomposition has a natural interpretation. If not all columns can be generated, the columns are added dynamically to the problem. Barnhart *et al.* [2] give a good overview on such techniques. A more recent review focussing on selected topics of column generation is [25]. In the context of vehicle routing problems, feasible tours have been added columns as needed by solving shortest path problem with time windows and capacity constraints using dynamic programming [7].

More generally, column generation techniques are used to solve well structured MILP problems involving a huge number, say several hundred thousand or millions, of variables, *i.e.*, columns. Such problems lead to large LP problems, if the integrality constraints of the integer variables are relaxed. If the LP problem contains so many variables (columns) that it cannot be solved with a direct LP solver (revised simplex, interior point method) one starts solving this so-called *master problem* with a small subset of variables yielding the *restricted master problem*. After the restricted master problem has been solved, a pricing problem is solved to identify new variables. This step corresponds to the identification of a non-basic variable to be taken into the basis of the simplex algorithm and coined the term *column generation*. The restricted master problem is solved with the new number of variables. The method terminates when the pricing problems cannot identify any new variables. The most simple version of column generation is found in the Dantzig-Wolfe decomposition [6].

Gilmore & Gomory ([10], [11]) were the first who generalized the idea of dynamic column generation to an integer programming (IP) problem: the cutting stock problem. In this case, the pricing-problem, *i.e.*, the subproblem, is an IP problem itself - and one refers to this as a *column generation algorithm*. This problem is special as the columns generated when solving the relaxed master problem are sufficient to get the optimal integer feasible solution of the overall problem. In general this is not so. If not only the subproblem, but also the master problem involves integer variables, the column generation part is embedded into a branch-and-bound method: this is called *branch-and-price*. Thus, branch-and-price is integer programming with column generation. Note that during the branching process new columns are generated; therefore the name *branch-and-price*.

3.1.1 Column Generation in Cutting Stock Problems This section describes the mathematical model for minimization of number of roles or trimloss and illustrates the idea of column generation.

Indices used in this model:

$p \in \mathcal{P} := \{p_1, \ldots, p_{N^{\mathrm{P}}}\}$ for cutting patterns (formats).
Either the patterns are directly generated according to a complete enumeration or they are generated by column generation.
$i \in \mathcal{I} := \{i_1, \ldots, i_{N^{\mathrm{I}}}\}$ given orders or widths.

Input Data We arrange the relevant input data size here:

B [L] width of the rolls (raw material roles).
D_i [-] number of orders for the width i.
W_i [L] width of order type i.

Integer Variables used in the different model variants:

$\mu_p \in \mathbb{N}_0 := \{0, 1, 2, 3, \ldots\}$ $[-]$ indicates how often pattern p is used.
If cutting pattern p is not used then is $\mu_p = 0$.
$\alpha_{ip} \in \mathbb{N}_0$ $[-]$ indicates how often width i is contained in pattern p.
This variable can take values between 0 and D_i depending on order situation.

Model The model contains a suitable object function

$$\min f(\alpha_{ip}, \mu_p) \quad ,$$

as well as the boundary condition (fulfillment of the demand)

$$\sum_p \alpha_{ip}\mu_p = D_i \quad , \quad \forall i \quad . \tag{3.9}$$

and the integrality constraints

$$\alpha_{ip} \in \mathbb{N}_0 \quad , \quad \forall\{ip\} \tag{3.10}$$

$$\mu_p \in \mathbb{N}_0 \quad , \quad \forall\{p\} \quad . \tag{3.11}$$

3.1.2 General Structure of the Problem In this form it is a mixed integer nonlinear optimization problem (MINLP). This problem class is difficult in itself. More seriously is, that we may easily encounter several million variables α_{ip}. Therefore the problem cannot be solved in this form.

Solution Method The idea of the dynamic column generation is based on the fact to decide in a master problem for a predefined set of patterns how often every pattern has to be used as well as calculating suitable input data for a sub-problem. In the mentioned sub-problem new patterns are calculated.

The master problem solves for the multiplicities of existing pattern and has the shape:

$$\min \sum_p \mu_p \quad ,$$

with the demand-fulfill inequality (note that it is allowed to produce more than requested)

$$\sum_i N_{ip}\mu_p \geq D_i \quad , \quad \forall i \quad . \tag{3.12}$$

and the integrality constraints

$$\mu_p \in \mathbb{N}_0 \quad , \quad \forall\{p\} \quad . \tag{3.13}$$

The sub-problem generates new patterns. Structurally it is a knapsack problem with object function

$$\min_{\alpha_i} 1 - \sum_p P_i\alpha_i \quad ,$$

where P_i are the dual values (pricing information) of the master problem (pricing problem) associated with (3.12) and α_i is an integer variables specifying how often width i occurs in the new pattern. We add the knapsack constraint respecting the width of the rolls

$$\sum_i W_i\alpha_i \leq B \quad , \quad \forall i \quad . \tag{3.14}$$

and the integrality constraints

$$\alpha_i \in \mathbb{N}_0 \quad , \quad \forall\{i\} \quad . \tag{3.15}$$

In some cases, α_i could be additionally bounded by the number, K, of knives.

3.1.3 Implementation Issues The critical issues in this method, in which we solve the master problem and sub-problem alternating, are the initialization of the procedure (a feasible starting point is to have one requested width in each initial pattern, but this is not necessarily a good one), excluding the generation of existing pattern by applying integer cuts, and the termination.

3.2 Column Enumeration

Column enumeration is a special variant of column generation and is applicable when a small number of columns is sufficient. This is, for instance, the case in real world cutting stock problems when it is known that the optimal solution have only a small amount of trimloss. This, usually, eliminates most of the pattern. Column enumeration naturally leads to a type of selecting columns or partitioning models. A collection of illustrative examples contained in Schrage (2006, [35], Sect. 11.7) covers several problems of grouping, matching, covering, partitioning, and packing in which a set of given objects has to be grouped into subsets to maximize or minimize some objective function. Despite the limitations with respect to the number of columns, column enumeration has some advantages:

- no pricing problem,
- easily applicable to MIP problems,
- column enumeration is much easier to implement.

In the online version of the vehicle routing problem described in Kallrath (2004, [22]) it is possible to generate the complete set, $\mathcal{C}_r$, of all columns, *i.e.*, subsets of orders $i \in \mathcal{O}$, $r = |\mathcal{O}|$, assigned to a fleet of n vehicles, $v \in \mathcal{V}$. Let $\mathcal{C}_r$ be the union of the sets, $\mathcal{C}_{rv}$, *i.e.*, $\mathcal{C}_r = \cup_{v=1...n}\mathcal{C}_{rv}$ with $C_r = |\mathcal{C}_r| = 2^r n$, where $\mathcal{C}_{rv}$ contains the subsets of orders assigned to vehicle v. Note that $\mathcal{C}_{rv}$ contains all subsets containing 1, 2, or r orders assigned to vehicle v. The relevant steps of the algorithm are:

1. Explicit generation of all columns $\mathcal{C}_{rv}$; followed by a simple feasibility test *w.r.t.* the availability of the cars.
2. Solution of the routing-scheduling problem for all columns $\mathcal{C}_{rv}$ using a tailor-made branch-and-bound approach (the optimal objective function values, $Z(\tau_c)$ or $Z(\tau_{cv})$, respectively, and the associated routing-scheduling plan are stored).
3. Solving the partitioning model:

$$\min_{\gamma_{cv}} \sum_{c=1}^{C_{rv}} \sum_{v=1}^{N^V} Z(\tau_{cv})\gamma_{cv} \quad , \tag{3.16}$$

s.t.

$$\sum_{c=1}^{C_r} \sum_{v=1}^{N^V} I_i(\tau_{cv})\gamma_{cv} = 1 \quad , \quad \forall i = 1, ..., r \tag{3.17}$$

ensures that each order is contained exactly once, the inequality

$$\sum_{c=1}^{C_r} \gamma_{cv} \leq 1 \quad , \quad \forall v \in \mathcal{V} \quad , \tag{3.18}$$

ensuring that at most one column can exists for each vehicle, and the integrality conditions

$$\gamma_{cv} \in \{0,1\} \quad , \quad \forall c = 1, ..., C_r \quad . \tag{3.19}$$

Note that not all combinations of index pairs $\{c, v\}$ exist; each c corresponds to exactly one v, and vice versa. This formulation allows us to find optimal solutions with the defined columns for a smaller number of vehicles. The objective function and the partitioning constraints are just modified by replacing

$$\sum_{v=1|v\in\mathcal{V}}^{N^V} \longrightarrow \sum_{v=1|v\in\mathcal{V}_*}^{N^V} \quad ,$$

the equations

$$\sum_{c=1}^{C_{rv}} \sum_{v=1|v\in\mathcal{V}_*}^{N^V} I_i(\tau_{cv})\gamma_{cv} = 1 \quad , \quad \forall i = 1, ..., r \quad ,$$

and the inequality

$$\sum_{c=1}^{C_{rv}} \gamma_{cv} \leq 1 \quad , \quad \forall v \in \mathcal{V}_* \quad ,$$

where $\mathcal{V}_* \subset \mathcal{V}$ is a subset of the set $\mathcal{V}$ of all vehicles. Alternatively, if it is not pre-specified which vehicles should be used but it is only required that not more than N_*^V vehicles are used, then the inequality

$$\sum_{c=1}^{C_r} \sum_{v=1|v\in\mathcal{V}}^{N^V} \gamma_{cv} \leq N_*^V \tag{3.20}$$

is imposed.

4. Re-constructing the complete solution and extracting the complete solution from the stored optimal solutions for the individual columns.

3.3 Branch&Price

Branch&Price (often coupled with Branch&Cut) are tailor-made implementations exploiting a decomposition structure. This efficient method for solving MIP problems with column generation has been well described by Barnhart *et al.* (1998, [2]) and has been covered by Savelsbergh (2001, [34]) in the first edition of the *Encyclopedia of Optimization*. Here, we give a list of more recent successful applications in various fields.

Cutting Stock: Vanderbeck (2000, [38]), Belov & Scheithauer (2006, [3])

Engine Routing and Industrial In-Plant Railroads: Lübbecke & Zimmermann (2003, [26])

Network design: Irnisch (2002, [16])
Lot-Sizing: Vanderbeck (1998, [38])
Scheduling (staff planning): Eveborn & M. Ronnqvist (2004, [8])
Scheduling of Switching Engines: Lübbecke & Zimmermann (2001, [24])
Supply chain optimization (pulp industry): Bredström et al. (2004, [5])
Vehicle Routing: Desrochers et al. (1992, [7]), Irnisch (2000, [15])

3.4 Rolling Time Decomposition

The overall methodology for solving the medium-range production scheduling problem is to determine the large and complex problem into smaller short-term scheduling subproblems in successive time horizons, *i.e.*, we decompose according to time. Large-scale industrial problems have been solved by Janak et al. (2006a, b; [18], [19]). A decomposition model is formulated and solved to determine the current horizon and corresponding products that should be included in the current subproblem. According to the solution of the decomposition model, a short-term scheduling model is formulated using the information on customer orders, inventory levels, and processing recipes. The resulting MILP problem is a large-scale complex problem which requires a large computational effort for its solution. When a satisfactory solution is determined, the relevant data is output and the next time horizon is considered. The above procedure is applied iteratively in an automatic fashion until the whole scheduling period under consideration is finished.

Note that the decomposition model determines automatically how many days and products to consider in the small scheduling horizon subject to an upper limit on the complexity of the resulting mathematical model. However, the complexity limit can be violated in order to ensure that each small scheduling filled to produce a final product.

4 An Exhaustion Method

This method combines aspects of an constructive heuristics and of exact model solving. We illustrate the exhausting method by the cutting stock problem described in Sect. 3.1.1; assigning orders in a scheduling problem would be another example. The elegant column generation approach by Gilmore & Gomory (1961, [10]) is known for producing minimal trimloss solutions with *many* patterns. Often this corresponds to setup changes on the machine and therefore is not desirable. A solution with a minimal a number of patterns minimizes the machine setup costs of the cutter. Minimizing simultaneously trimloss and the number of patterns is possible for small case of a few orders only exploiting the MILP model by Johnston & Salinlija (2004, [20]). It contains two conflicting objective functions. Therefore one could resort to goal programming. Alternatively, we produce several parameterized solutions leading to different number of rolls to be used and patterns to be cut from which the user extract the one he likes best.

```
 # of  #  output file  flag Wmax            comment
rolls pat
-------------------------------------------------------------------------------------------------------
  0    5                8   99   lower bound: minimal # of patterns
  30  10   pat00.out     9   99   lower bound: minimal # of rolls
  34   7   pat01.out     0   20
  31   9   pat02.out     1   15
  30   8   pat03.out     0   10   minimal number of rolls
  32   9   pat04.out     1    8
  30   8   pat05.out     0    6   minimal number of rolls
  31   8   pat06.out     1    4

 The best solution found contains 7 patterns !
 Solution with minimal trimloss contain 30 rolls !

 Improvement of the lower bound of pattern : 6 !
 Solutions with 6 patterns are minimal w.r.t.
 to the number of patterns.

 Found new solution with only 6 patterns and 36 rolls: patnew.out
  36   6   patnew.out  0  99
```

As the table above indicates we compute tight lower bounds on both trimloss and the number of patterns. Even for up to 50 orders feasible, near-optimal solutions are construct in less than a minute.

Note that it would be possible to use the Branch&Price algorithm described in Vanderbeck (2000, [38]) or Belov & Scheithauer (2006, [3]) to solve the one-dimensional cutting stock problem with minimal numbers of pattern. However, these methods are not easy to implement. Therefore, we use the following approaches which are much easier to program:

- V1: Direct usage of the model by Johnston & Salinlija (2004, [20]) for a small number, say, $N^I \leq 14$ of orders and $D_{\max} \leq 10$. In a preprocessing step we compute valid inequalities as well as tight lower and upper bounds on the variables.
- V2: Exhaustion procedure in which we generate successively new patterns with maximal multiplicities. This method is parametrized by the permissible percentage waste $W_{\max}$, $1 \leq W_{\max} \leq 99$. After a few patterns have been generated with this parameterization, it could happen that is is not possible to generate any more pattern with waste restriction. In this case the remaining unsatisfied order are generated by V1 without the $W_{\max}$ restriction.

4.1 Indizes and Sets

In this model we use the indices listed in Johnston & Salinlija (2004):

$i \in \mathcal{I} := \{i_1, \ldots, i_{N^{\mathrm{I}}}\}$ the sets of width.

$j \in | := \{j_1, \ldots, j_{N^{\mathrm{P}}}\}$ the pattern; $N^{\mathrm{J}} \leq N^I$.
The pattern are generated by V1, or dynamically by maximizing the multiplicities of a pattern used.

$k \in \mathcal{K} := \{k_1, \ldots, k_{N^{\mathrm{P}}}\}$ the multiplicity index to indicate how often a width is used in a pattern.
The multiplicity index can be restricted by the ratio of the width of the orders and the width of the given rolls.

4.2 Variables

The following integer or binary variables are used:

$a_{ijk} \in \mathbb{N}$ [−] specifies the multiplicity of pattern j.
The multiplicity can vary between 0 and $D_{\max} := \max\{D_i\}$. If pattern j is not used, we have $r_j = p_j = 0$.

$p_j \in \{0, 1\}$ [−] indicates whether pattern j is used at all.

$r_j \in \mathbb{N}$ [−] specifies how often pattern j is used.
The multiplicity can vary between 0 and $D_{\max} := \max\{D_i\}$. If pattern j is not used, we have $r_j = p_j = 0$.

$\alpha_{ip} \in \mathbb{N}$ [−] specifies how often width i occurs in pattern p.
This width-multiplicity variable can take all values between 0 and D_i.

$x_{ijk} \in \{0, 1\}$ [−] indicates whether width i appears in pattern j at level k.
Note that $x_{ijk} = 0$ implies $a_{ijk} = 0$.

4.3 The Idea of the Exhaustion Method

In each iteration we generate m at most 2 or 3 new patterns by maximizing the multiplicities of these pattern allowing not more than a maximum waste, $W_{\max}$. The solution generated in iteration m is preserved in iteration $m+1$ by fixing the appropriate variables. If the problem turns out to be infeasible (this may happen if $W_{\max}$ turns out to be restrictive) we switch to a model variant in which we minimize the number of patterns subject to satisfying the remaining unsatisfied orders.

The model is based on the inequalities (2,3,5,6,7,8,9) in Johnston & Salinlija (2004, [20]), but we add a few more additional ones or modify the existing ones. We exploit two objective functions: maximizing the multiplicities of the patterns generated

$$\max \sum_{j=1}^{\pi_u} r_j \quad ,$$

where π_u specifies the maximal number of patterns (π_u could be be taken from the solution of the column generation approach, for instance), or minimizing the number of patterns generated

$$\min \sum_{j=1}^{\pi_u} p_j \quad .$$

The model is completed by the integrality conditions

$$r_j, a_{ijk} \in \{0,1,2,3,\ldots\} \tag{4.21}$$

$$p_j, x_{ijk}, y_{jk} \in \{0,1\} \quad . \tag{4.22}$$

The model is applied several times with $a_{ijk} \leq \tilde{D}_i$, where $\tilde{D}_i$ is the number of remaining orders of width i. Especially, the model has to fulfill the relationships

$$k a_{ijk} > \tilde{D}_i \quad \Longrightarrow \quad a_{ijk} = 0 \quad , \quad x_{ijk} = 0$$

and

$$a_{ijk} \leq \left\lceil \frac{\tilde{D}_i}{k} \right\rceil \quad \text{bzw.} \quad a_{ijk} \leq \left\lceil \frac{\tilde{D}_i + S_i}{k} \right\rceil \quad ,$$

where S_i denotes the permissible overproduction.

The constructive method described so far provides an improved upper bound, π'_u, on the number of pattern.

4.4 Computing Lower Bounds

To compute a lower bound we apply two methods. The first method is to solve a bin packing problem which is equivalent to minimize the number of rolls in the original cutting stock problem described in Sect. 3.1.1 for equal demands $D_i = 1$. If solved with the column generation approach this method is fast and cheap, but the lower bound, π'_l, is often weak. The second method is to exploit the upper bound, π'_u, on the number of patterns obtained and to call the exact model as in V1. It is impressive how quickly the commercial solvers `CPLEX` and `XpressMP` improve the lower bound yielding $\pi^{"}_l$. For most examples with up to 50 orders we obtain $\pi'_u - \pi'_l \leq 2$, but in many cases $\pi'_u - \pi'_l = 1$ or even $\pi'_u = \pi^{"}_l$.

5 Primal Feasible Solutions and Hybrid Methods

We define hybrid methods as methods based any combination of exact MIP methods with constructive heuristics, local search, metaheuristics, or constraint programming which produces primal feasible solutions. Dive-and-fix, near-integer-fix, or fix-and-relax are such a hybrid methods. They are user

developed heuristics exploiting the problem structure. In their kernel they use a declarative model solved, for instance, by `CPLEX` and `XpressMP`.

In constructive heuristics we exploit the structure of the problem and compute a feasible point. Once we have a feasible point we can derive safe bounds on the optimum and assign initial values to the critical discrete variable which could be exploited by the `GAMS/CPLEX` *mipstart* option. Feasible points can sometimes be generated by appropriate sequences of relaxed models. For instance, in a scheduling problem $\mathcal{P}$ with due times one might relax these due times obtaining the relaxed model $\mathcal{R}$. The optimal solution, or even any feasible point of $\mathcal{R}$ is a feasible point of $\mathcal{P}$ if the due times are models with appropriate unbounded slack variables.

Constructive heuristics can also be established by systematic approaches of fixing crtitical discrete variables. Such approaches are *dive-and-fix* and *relax-and-fix*. In dive-and-fix the LP relaxation of an integer problem is to be solved followed by fixing a subset of fractional variables to suitable bounds. *Near-integer-fix* is a variant of dive-and-fix which fixes variables with fractional values to the nearest integer point. Note that these heuristics are subject to the risk of becoming infeasible.

The probability of becoming infeasible is less likely in relax-and-fix. In relax-and-fix following Pochet & Wolsey (2006, [32], pp.109) we suppose that the binary variables δ of a MIP problem $\mathcal{P}$ can be partitioned into R disjoint sets $S^1; \ldots; S^R$ of decreasing importance. Witin these subsets U^r with $U \subseteq \cup_{u=r+1}^{R} S^u$ for $r = 1; \ldots; R-1$ can be chosen to allow for somewhat more generality. Based on these partitions, R MIP problems are solved, denoted $\mathcal{P}^r$ with $1 \leq r \leq R$ to find a heuristic solution to $\mathcal{P}$. For instance in a production planning problem, S^1 might be all the δ variables associated with time periods in $\{1, \ldots, t_1\}$, S^u those associated with periods in $\{t_u + 1, \ldots, t_{u+1}\}$, whereas U^r would would be the δ variables associated with the periods in some set $\{t_r + 1, \ldots, u_r\}$.

In the first problem, $\mathcal{P}^1$, one only imposes the integrality of the important variables in $S^1 \cup U^1$ and relax the integrality on all the other variables in S. As $\mathcal{P}^1$ is a relaxation of $\mathcal{P}$, for a minimization problem, the solution of $\mathcal{P}^1$ provides a lower bound of $\mathcal{P}$. The solution values, δ^1, of the discrete variables are kept fixed when solving $\mathcal{P}^r$. This continues and in the subsequent $\mathcal{P}^r$, for $2 \leq r \leq R$, we additionally fix the values of the δ variables with index in S^{r-1} at their optimal values from $\mathcal{P}^{r-1}$, and add the integrality restriction for the variables in $S^r \cup U^r$.

Either $\mathcal{P}^r$ is infeasible for some $r \in \{1, \ldots, R\}$, and the heuristic failed, or else (x^R, δ^R) is a relax-and-fix solution. To avoid infeasibilities one might apply a smoothed form of this heuristic which allows for some overlap of U^{r-1} and U^r. Additional free binary variables in horizon $r-1$ allow to link the current horizon r with the previous one. Usually this suffices to ensure feasibility. Relax-and-fix comes in various flavours exploiting time decomposition or time partitioning structures. Other decompositions, for instance, plants, products, or customers, are possible as well.

Local search can be used to improve the solution obtained by the relax-and-fix heuristic. The main idea is to solve repeatedly the subproblem on small number of binary variables reoptimizing, for instance, the production of some products. The binary variables for resolving could be chosen randomly, or by a meta-heuristic such as simulated annealing. All binary variables related to them are released the other ones are fixed to the previous best values.

Another class of MIP hybrid method is established by algorithms which combine a MIP solver with another algorithmic method. A hybrid method obtained by the combination of mixed-integer and constraint logic programming strategies has been developed and applied by Harjunkoski *et al.* (2000, [14]) as well as Jain & Grossmann (2001, [17]) for solving scheduling and combinatorial optimization problems. Timpe (2002, [37]) solved a mixed planning and scheduling problems with mixed MILP branch-and-bound and constraint programming. Maravelias & Grossmann (2004, [27]) proposed a hybrid/decomposiiton algorithm for the short term scheduling of batch plants, and Roe *et al.* (2005, [33]) presented a hybrid MILP/CLP algorithm for multipurpose batch process scheduling, in which MILP is used to solve an aggregated planning problem while CP is used to solve a sequencing problem. Other hybrid algorithms combine evolutionary and mathematical programming methods, see, for instance, the heuristics by Till *et al.* (2005, [36]) for stochastic scheduling problems and by Borisovsky *et al.* (2006, [4]) for supply management problems.

Finally, one should not forget to add some algorithmic component, which for the minimization problem at hand, would generate some reasonable bounds to be provided in addition to the hybrid method. The hybrid methods discussed above provide upper bounds by constructing feasible points. In favourate cases, the MIP part of the hybrid solver provides lower bounds. In other case, lower bounds can be derived from auxiliary problems which are relaxations of the original problem, and which are easier to solve.

6 Summary

If a given MIP problem cannot be solved by an available MIP solver exploiting all its internal presolving techniques, one might reformulate the problem and getting an equivalent or closely related representation of the reality. Another aproach is to construct MIP solutions and bounds by solving a sequence of models. Alternatively, individual tailor-made exact decomposition techniques could help as well as primal heuristics such as relax-and-fix, or local search techniques on top of a MIP model.

Acknowledgements: I would like to thank Linus Schrage (University of Chicago) for his interest and constructive comments, Steffen Rebennack (University of Florida) for his careful proof-reading.

References

1. K. M. Anstreicher. Linear Programming: Interior Point Methods. In C. A. Floudas and P. Pardalos, editors, *Encyclopedia of Optimization*, volume 3, pages 189–191. Kluwer Academic Publishers, Dordrecht, Holland, 2001.
2. C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsberg, and P. H. Vance. Branch-and-price: column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998.
3. G. Belov and G. Scheithauer. A Branch-and-Crice Algorithm for One-Dimensional Stock Cutting and Two-Dimensional Two-Stage Cutting. *European Journal of Operational Research*, 171):85–106, 2006.
4. P. Borisovsky, A. Dolgui, and A. Eremeev. Genetic Algorithms for Supply Management Problem with Lower-bounded Demands. In A. Dolgui, G. Morel, and C. Pereira, editors, *Information Control Problems in Manufacturing 2006: A Proceedings volume from the 12th IFAC International Symposium. Vol. 3., St Etienne, France, 17-19 May 2006*, pages 521–526, Dordrecht, North-Holland, 2006. Elsevier.
5. D. Bredström, J. T. Lundgren, M. Rönqvist, D. Carlsson, and A. Mason. Supply Chain Optimization in the Pulp Mill Industry – IP models, Column Generation and Novel Constraint Branches. *European Journal of Operational Research*, 156):2–22, 2004.
6. B. Dantzig and P. Wolfe. The decomposition algorithm for linear programming. *Operations Research*, 8:101–111, 1960.
7. M. Desrochers, J. Desrosiers, and M. M. Solomon. A New Optimization Algorithm for the Vehicle Routing Problem with time Windows. *Operations Research*, 40(2):342–354, 1992, March-April.
8. P. Eveborn and M. Ronnqvist. Scheduler - A System for Staff Planning. *Annals of Operations Research*, 128:21–45, 2004.
9. C. A. Floudas. *Nonlinear and Mixed-Integer Optimization : Fundamentals and Applications*. Oxford University Press, Oxford, England, 1995.
10. P. C. Gilmore and R. E. Gomory. A Linear Programming Approach to the Cutting Stock Problem. *Operations Research*, 9:849–859, 1961.
11. P. C. Gilmore and R. E. Gomory. A Linear Programming Approach to the Cutting Stock Problem, Part II. *Operations Research*, 11:863–888, 1963.
12. M. Grötschel. Mathematische Optimierung im industriellen Einsatz. Lecture at Siemens AG, Munich, Germany, Dec 07, 2004.
13. M. Grötschel. private communication, 2005.
14. I. Harjunkoski, V. Jain, and I. E. Grossmann. Hybrid Mixed-integerconstraint Logic Programming Strategies for Solving Scheduling and Combinatorial Optimization Problems. *Computers and Chemical Engineering*, 24:337–343, 2000.
15. S. Irnich. A Multi-Depot Pickup and Delivery Problem with a Single Hub and Heterogeneous Vehicles. *European Journal of Operational Research*, 122:310–328, 2000.
16. S. Irnich. *Netzwerk-Design fr zweistufige Transportsysteme und ein Branch-and-Price-Verfahren für das gemischte Direkt- und Hubflugproblem*. Dissertation, Fakultät für Wirtschaftswissenschaften, RWTH Aachen, Aachen, Germany, 2002.
17. V. Jain and I. E. Grossmann. Algorithms for hybrid MILP/CP models for a class of optimization problems. *IFORMS Journal on Computing*, 13:258–276, 2001.

18. S. L. Janak, C. A. Floudas, J. Kallrath, and N. Vormbrock. Production Scheduling of a Large-Scale Industrial Batch Plant: I. Short-Term and Medium-Term Scheduling. *Industrial and Engineering Chemistry Research*, 45:8234–8252, 2006a.
19. S. L. Janak, C. A. Floudas, J. Kallrath, and N. Vormbrock. Production Scheduling of a Large-Scale Industrial Batch Plant: II. Reactive Scheduling. *Industrial and Engineering Chemistry Research*, 45:8253–8269, 2006b.
20. R. E. Johnston and E. Sadinlija. A New Model for Complete Solutions to One-Dimensional Cutting Stock Problems. *European Journal of Operational Research*, 153:176–183, 2004.
21. J. Kallrath. Mixed Integer Optimization in the Chemical Process Industry: Experience, Potential and Future Perspectives. *Chemical Engineering Research and Design*, 78(6):809–822, 2000.
22. J. Kallrath. *Online Storage Systems and Transportation Problems with Applications: Optimization Models and Mathematical Solutions*, volume 91 of *Applied Optimization*. Kluwer Academic Publishers, Dordrecht, Holland, 2004.
23. J. Kallrath and T. I. Maindl. *Real Optimization with SAP-APO*. Springer, Heidelberg, Germany, 2006.
24. M. Lübbecke and U. Zimmermann. Computer aided scheduling of switching engines. In W. Jäger and H.-J. Krebs, editors, *Mathematics—Key Technology for the Future: Joint Projects Between Universities and Industry*, pages 690–702. Springer-Verlag, Berlin, 2003.
25. M. E. Lübbecke and J. Desrosiers. Selected topics in column generation. *Oper. Res.*, 53(6):1007–1023, 2005.
26. M. E. Lübbecke and U. T. Zimmermann. Engine routing and scheduling at industrial in-plant railroads. *Transportation Sci.*, 37(2):183–197, 2003.
27. C. T. Maravelias and I. E. Grossmann. A Hybrid MILP/CP Decomposition Approach for the Continuous Time Scheduling of Multipurpose Batch Plants. *Computers and Chemical Engineering*, 28:1921–1949, 2004.
28. A. Martin. General Mixed Integer Programming: Computational Issues for Branch-and-Cut Algorithms. In D. Naddef and M. Juenger, editors, *Computational Combinatorial Optimization*, pages 1–25. Springer, Berlin, 2001.
29. S. Menon and L. Schrage. Order Allocation for Stock Cutting in the Paper Industry. *Operations Research*, 50(2):324–332, 2002.
30. G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley, New York, 1988.
31. P. M. Pardalos. Linear Programming. In C. A. Floudas and P. Pardalos, editors, *Encyclopedia of Optimization*, volume 3, pages 186–188. Kluwer Academic Publishers, Dordrecht, Holland, 2001.
32. Y. Pochet and L. A. Wolsey. *Production Planning by Mixed Integer Programming*. Springer, New York, 2006.
33. B. Roe, L. G. Papageorgiou, and N. Shah. A hybrid MILP/CLP Algorithm for Multipurpose Batch Process schedulin. *Computers and Chemical Engineering*, 29:1277–1291, 2005.
34. M. W. P. Savelsbergh. Branch-and-Price: Integer Programming with Column Generation. In C. A. Floudas and P. Pardalos, editors, *Encyclopedia of Optimization*, pages 218–221. Kluwer Academic Publishers, Dordrecht, Holland, 2001.
35. L. Schrage. *Optimization Modeling with LINGO*. LINDO Systems, Inc., Chicago, IL, 2006.

36. J. Till, , G. Sand, S. Engell, M. Emmerich, and L. Schönemann. A New Hybrid Algorithm for Solving Two-Stage Stochastic Problems by Combining Evolutionary and Mathematical Programming Methods. In L. Puigjaner and A. Espuña, editors, *Proc. European Symposium on Computer Aided Process Engineering (ESCAPE) - 15*, pages 187–192, Dordrecht, North-Holland, 2005. Elsevier.
37. C. Timpe. Solving mixed planning & scheduling problems with mixed branch & bound and constraint programming. *OR Spectrum*, 24:431–448, 2002.
38. F. Vanderbeck. Exact algorithm for minimising the number of setups in the one-dimensional cutting stock problem. 2000.
39. L. A. Wolsey. *Integer Programming*. Wiley, New York, US, 1998.