RC23589 (W0504-076) April 15, 2005
Mathematics

IBM Research Report

In Situ Column Generation for a Cutting-Stock Problem

Jon Lee
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

In Situ Column Generation for a Cutting-Stock Problem

Jon LEE

IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, U.S.A.
jonlee@us.ibm.com

Abstract. Working with an integer bilinear programming formulation of a one-dimen-sional cutting-stock problem, we develop an ILP-based local-search heuristic. The ILPs holistically integrate the master and subproblem of the usual price driven pattern-generation paradigm, resulting in a unified model that generates new patterns *in situ*. We work harder to generate new columns, but we are guaranteed that new columns give us an *integer* linear-programming improvement. The method is well suited to practical restrictions such as when a limited number of cutting patterns should be employed, and our goal is to generate a profile of solutions trading off trim loss against the number of patterns utilized. We describe our implementation and results of computational experiments on instances from a chemical-fiber company.

Introduction

We assume familiarity with the basics of integer linear programming (see [23], for example). Let m be a positive integer, and let $M := \{1, 2, \ldots, m\}$. Stock rolls of (usable) width $W_{\max}$ are available for satisfying demands d^i for rolls of width $w^i (< W_{\max})$, $i \in M$. The classical (one-dimensional) *CSP (cutting-stock problem)* is to minimize the trim loss created in covering the demands. Many models for the problem employ the idea of a cutting pattern. A *pattern* is a $z \in \mathbb{Z}_+^M$ satisfying

$$W_{\min} \leq \sum_{i \in M} w^i z^i \leq W_{\max} \ , \tag{1}$$

where $W_{\min}$ is a lower bound on the allowed *nominal trim loss* per stock roll (note that the total actual trim loss may be greater than the sum of the nominal trim losses, since we allow the possibility of over production for our solution). The variable z^i represents the number of rolls of width w^i that are obtained from a single stock roll when we employ this pattern. With this concept, most models seek to determine which patterns to use and how many times to repeat each pattern to cover the demand. This is the classical framework of Gilmore and Gomory (GG) [10] who applied a simplex-method based column-generation scheme to solve the LP relaxation, followed by rounding the column utilizations up.

We allow for some variations of the classical model to accommodate practical application. In actual instances of the CSP, the cutting machines have a limited number, say †, of (movable) knives. Ordinarily there are a pair of stationary knives at each end that cut the role to the standard width of $W_{\max}$. We do not count the boundary scrap trimmed by these stationary knives as trim loss. Hence we have the added restriction

$$\sum_{i \in M} z^i \leq \dagger + \frac{\sum_{i \in M} w^i z^i}{W_{\max}} \ , \tag{2}$$

or, equivalently,

$$\sum_{i \in M} (W_{\max} - w^i) z^i \leq \dagger W_{\max} \ , \tag{3}$$

since we want to allow $\dagger + 1$ pieces to be cut when there is no nominal trim loss for a pattern.

Also, widths up to some threshold ω are deemed to be *narrow*, and there is often a limit on the number C_{nar} of narrows permitted to be cut from a stock roll. So we add the restriction

$$\sum_{i \in M_{\text{nar}}} z^i \leq C_{\text{nar}} \ , \tag{4}$$

where $M_{\text{nar}} := \{i \in M \ : \ w_i \leq \omega\}$.

Additionally, we allow for the possibility of not requiring demand to be exactly satisfied. This is often customary for real applications, for example in the paper industry. Moreover, allowing this flexibility can lead to a significant decrease in trim loss. We assume that for some small nonnegative integers q^i and p^i, the customer will accept delivery of between $d^i - q^i$ and $d^i + p^i$ rolls of width w^i, for $i \in M$. We allow for over production, beyond $d^i + p^i$ rolls of width w^i, but we treat that as trim loss.

Finally, we limit the number of patterns allowed to some number $n_{\max}$. This kind of limitation can be quite practical for actual applications, but it is difficult to handle computationally (see [22] and the references therein). Heuristic approaches to this type of limitation include: variations on the simplex-method based GG approach (see [30], [7] and [14]), incremental heuristics for generating patterns (see [12] and [13]), local-search methods for reducing the number of patterns as a post-processing stage (see [15], [11] and [8]), linear-programming based local search (see [26]), heuristically-based local search (see [27]) and [28]). While a computationally intensive Branch, Cut and Price methodology is explored in [29] and later a Branch and Price approach in [2]. Our goal is find a good compromise method — more powerful than existing heuristics, but with less effort than exact ILP approaches. Such a tool can be used within an exact ILP approach on the primal side, and may also be used when other heuristics fail to give good solutions. So, we view our contribution as complementary to existing approaches. Moreover, often it is hard to exactly quantify the cost of changing changing cutting patterns. Our method is particularly well suited to generating a profile of solutions trading off trim loss against the number of patterns employed. Then an appropriate solution can be selected from such a Pareto curve.

Before getting into the specifics of our model and algorithm, we give a high-level view of our philosophy. As we have mentioned, most models and associated algorithms for CSPs employ the idea of a cutting pattern. With this idea, many methods follow the GG pattern-generation approach. Constraints involving coverage of demand using a known set of patterns are treated in a "master problem" and constraints describing feasible patterns are treated in a subproblem. As we seek a LP solution at the master level in the GG approach, we can communicate the necessary information (capturing demand coverage) to the subproblem via prices. But since we are really after an integer solution to the master, prices are not sufficient to describe optimal demand coverage based on known patterns to the subproblem. We address this shortcoming of the usual pattern-generation approach by formulating a holistic model that *simultaneously* seeks a small number of new patterns, their usages, and the usages of the current set of patterns. That is, the unified model generates new patterns *in situ*. Another benefit of this approach is that it is easily imbedded in a local-search framework which is ideal for working with constraints that are not modelled effectively in an ILP setting (like the cardinality constraint limiting the number of patterns that we may employ) — both finding a point solution and generating a profile of solutions.

In §1, we describe the bilinear model of our CSP. In §2, we modify the model by fixing some patterns and linearizing the remaining bilinear terms. In §3, we describe our computational methodology for solving the model of §2 and for generating a sequence of such models aimed at producing a profile of good solutions trading off trim loss against the number of patterns utilized. In §5, we describe the results on computational experiments. In §4, we report on preliminary experiments designed to examine the trade off between economizing on some variables and the quality of our solutions. Finally, in §6, we describe some extensions on our approach.

1 The Bilinear Model

Let $N := \{1, 2, \ldots, n\}$. We describe a set of n patterns by the columns of an $m \times n$ matrix Z. So, z^{ij} is the number of rolls of width w^i that is cut from a stock roll when we employ pattern number j. Let x^j be a nonnegative integer variable that represents the number of times pattern number j is employed toward satisfying the demands. We let the variable s^i be the part of the deviation (from demand d^i) of the production of width w^i that is within the allowed tolerance. The variable t^i is the part of the deviation that exceeds the upper tolerance and is treated as trim loss. Then we have the integer *bilinear* programming formulation which seeks to minimize trim loss:

$$\min\ W_{\max} \sum_{j \in N} x^j - \sum_{i \in M} w^i (d^i + s^i) \tag{5}$$

subject to

$$W_{\min} \le \sum_{i \in M} w^i z^{ij} \le W_{\max}\ , \quad \forall\ j \in N\ ; \tag{6}$$

$$\sum_{i \in M} (W_{\max} - w^i) z^{ij} \le \dagger W_{\max}\ , \quad \forall\ j \in N\ ; \tag{7}$$

$$\sum_{i \in M_{\text{nar}}} z^{ij} \le C_{\text{nar}}\ , \quad \forall\ j \in N\ ; \tag{8}$$

$$z^{ij} \ge 0 \text{ integer}\ , \quad \forall\ i \in M,\ j \in N\ ; \tag{9}$$

$$\sum_{j \in N} z^{ij} x^j - s^i - t^i = d^i\ , \quad \forall\ i \in M\ ; \tag{10}$$

$$-q^i \le s^i \le p^i\ , \quad \forall\ i \in M\ ; \tag{11}$$

$$t^i \ge 0\ , \quad \forall\ i \in M\ ; \tag{12}$$

$$x^j \ge 0 \text{ integer}, \quad \forall\ j \in N\ . \tag{13}$$

There are a few strategies for dealing with the bilinearity of (10). Allowing N to be so large that all possible patterns can be accommodated simultaneously, and applying decomposition (see Dantzig and Wolfe) [5, 6] in a certain manner, leads to the classical linear-programming based column-generation approach of GG [10].

In the next section, we describe an alternative approach, which does not rely on decomposition and thus does not involve an enormous number of variables. We seek a good (heuristic) solution to (5–13). Our approach is to partition N into two sets: $\bar{N}$ and $\hat{N}$. For $j \in \bar{N}$, we treat the pattern variables z^{ij} as fixed, and we treat the x^j as true variables. For $j \in \hat{N}$, we write all of the variables x^j and z^{ij} in binary expansion and then we linearize the binary products. We start with $\bar{N}$ empty, and $\hat{N}$ of a very limited size (say no more than 3 elements). While we have fewer than $n_{\max}$ patterns in $\bar{N}$, we repeatedly solve the problem, moving the resulting optimal patterns from $\hat{N}$ into $\bar{N}$. After each move of new patterns to $\bar{N}$, we can do local search, testing whether dropping a few patterns and re-generating offers an improvement.

2 Partial Linearization

Let β^i be an upper bound on the greatest number of bits used for representing z^{ij} in any solution to (6–9). For example, we can take

$$\beta^i := \left\lceil \log_2 \left(1 + \min\left\{ \lfloor W_{\max}/w^i \rfloor, \lfloor \dagger/(1 - w^i/W_{\max}) \rfloor \right\}\right) \right\rceil\ , \quad \forall\ i \in M \setminus M_{\text{nar}}\ ,$$

and

$$\beta^i := \left\lceil \log_2 \left(1 + \min\left\{ \lfloor W_{\max}/w^i \rfloor, \lfloor \dagger/(1 - w^i/W_{\max}) \rfloor, C_{\text{nar}} \right\}\right) \right\rceil \ , \quad \forall\ i \in M_{\text{nar}} \ .$$

Let β be an upper bound on the greatest number of bits used for representing x^j in an optimal solution to (5–13). For example, we can take $\beta = \lceil \log_2(1 + \sum_{j \in N} x^j) \rceil$, where (x, z, s, t) is any feasible solution to (5–13). It is desirable to have the β^i and β as small as possible.

Let $L^i := \{0, 1, \ldots, \beta^i - 1\}$, for all $i \in M$, and let $K := \{0, 1, \ldots, \beta - 1\}$. We define binary variables z_l^{ij}, for all $i \in M,\ j \in N,\ l \in L^i$, and binary variables x_k^j, for all $j \in N,\ k \in K$. These are just the bits in the binary representations of the z^{ij} and the x^j. For a given $j \in N$, we refer to the z^{ij} as the *integer-encoded pattern* j and the z_l^{ij} as the *binary-encoded pattern* j. So, letting

$$z^{ij} = \sum_{l \in L^i} 2^l z_l^{ij} \ , \quad x^j = \sum_{k \in K} 2^k x_k^j \ , \tag{14}$$

and

$$y_{kl}^{ij} := z_l^{ij} x_k^j \ , \tag{15}$$

we have

$$y^{ij} := \left(\sum_{l \in L^i} 2^l z_l^{ij} \right) \left(\sum_{k \in K} 2^k x_k^j \right) = \sum_{l \in L^i} \sum_{k \in K} 2^{l+k} y_{kl}^{ij} \ . \tag{16}$$

The *boolean-quadric constraints* (15) are well understood, and we apply the standard linearization (23) (see [24] for example).

So we work with the following ILP formulation:

$$\min\ W_{\max} \left(\sum_{j \in \bar{N}} x^j + \sum_{j \in \hat{N}} \sum_{k \in K} 2^k x_k^j \right) - \sum_{i \in M} w^i (d^i + s^i) \tag{17}$$

subject to

$$W_{\min} \le \sum_{i \in M} w^i \sum_{l \in L^i} 2^l z_l^{ij} \le W_{\max} \ , \quad \forall\ j \in \hat{N} \ ; \tag{18}$$

$$\sum_{i \in M} (W_{\max} - w^i) \sum_{l \in L^i} 2^l z_l^{ij} \le \dagger W_{\max} \ , \quad \forall\ j \in \hat{N} \ ; \tag{19}$$

$$\sum_{i \in M_{\text{nar}}} \sum_{l \in L^i} 2^l z_l^{ij} \le C_{\text{nar}} \ , \quad \forall\ j \in \hat{N} \ ; \tag{20}$$

$$z_l^{ij} \ge 0 \text{ integer} \ , \quad \forall\ i \in M,\ j \in \hat{N} \ , \ l \in L^i \ ; \tag{21}$$

$$\sum_{j \in \bar{N}} z^{ij} x^j + \sum_{j \in \hat{N}} \sum_{k \in K} \sum_{l \in L^i} 2^{k+l} y_{kl}^{ij} - s^i - t^i = d^i \ , \quad \forall\ i \in M \ ; \tag{22}$$

$$y_{kl}^{ij} \le z_l^{ij} \ , \quad y_{kl}^{ij} \le x_k^j \ , \quad z_l^{ij} + x_k^j \le 1 + y_{kl}^{ij} \ , \quad \forall\ i \in M \ , \ j \in \hat{N} \ , \ k \in K \ , \ l \in L^i \ ; \tag{23}$$

$$-q^i \le s^i \le p^i \ , \quad \forall\ i \in M \ ; \tag{24}$$

$$t^i \ge 0 \ , \quad \forall\ i \in M \ ; \tag{25}$$

$$x^j \ge 0 \text{ integer}, \quad \forall\ j \in \bar{N} \ ; \tag{26}$$

We note that the difficulty in working to solve instances of (17–26) centers on the boolean quadric part of the model when $\hat{N}$, K and the L^i are big.

3 Computational Methodology

For each $i \in M$, $j \in \hat{N}$, the variables and equations of (15) describe the *boolean quadric polytope* (see [24]) of the complete bipartite graph of Figure 1. Recognizing this, we can strengthen our formulation using known valid inequalities. Already, we can get some mileage by including simple ones that are not of the form (23). Specifically, we utilize all of the remaining facets of the boolean quadric polytopes of all of the 4-cycles of the graph:

$$x_k^j + z_l^{ij} + y_{k'l'}^{ij} \leq 1 + y_{kl}^{ij} + y_{kl'}^{ij} + y_{k'l}^{ij} \ , \tag{27}$$

$$y_{k'l}^{ij} + y_{kl'}^{ij} + y_{kl}^{ij} \leq x_k^j + z_l^{ij} + y_{k'l'}^{ij} \ , \tag{28}$$

$$\forall\ i \in M,\ j \in \hat{N},\ k,k' \in K,\ l,l' \in L^i \ .$$

Rather than try and tease these inequalities out of [24], it is easy to obtain them using a facet enumeration code such as Komei Fukuda's `cdd` [9] or Thomas Christof and Andreas Löbel's `PORTA` [3] (to calculate all of the facets of the boolean quadric polytope of a 4-cycle).

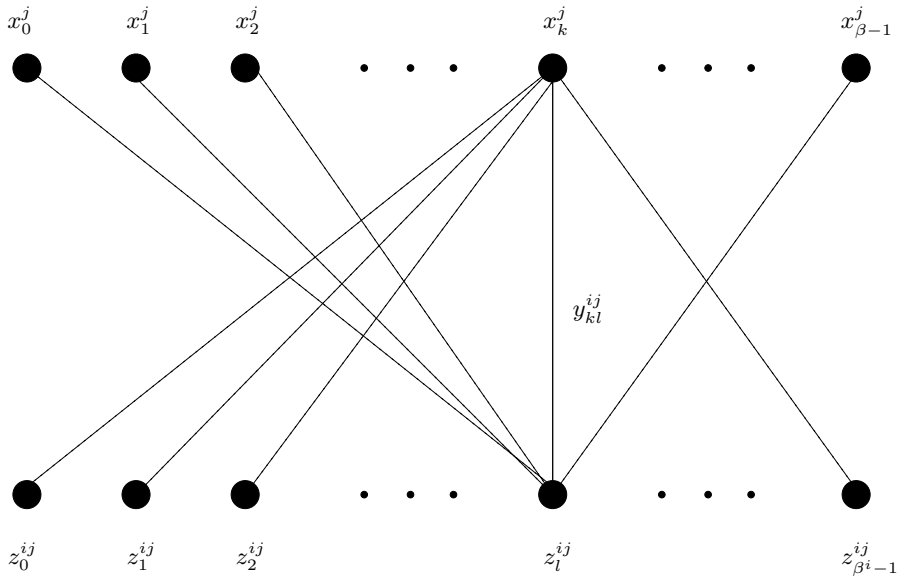

Fig. 1. Complete bipartite graph on (K, L^i)

An ugly fact of life for this formulation is the symmetry of permuting the patterns in a solution. This is not a problem within $\bar{N}$, since these (integer-encoded) patterns are fixed. The problem would be generating a (binary-encoded) pattern indexed in $\hat{N}$ that is already indexed in $\bar{N}$ (as an integer-encoded pattern). As is well known, such symmetry can be devastating for any branching procedure. The *pattern-exclusion constraints*

$$\sum_{i \in M,\ l \in L^i\ :\ z_l^{i\bar{\jmath}}=0} z_l^{ij} + \sum_{i \in M,\ l \in L^i\ :\ z_l^{i\bar{\jmath}}=1} \left(1 - z_l^{ij}\right) \geq 1 \ , \quad \forall\ j \in \hat{N},\ \bar{\jmath} \in \bar{N} \tag{29}$$

insure that we do not regenerate a (binary-encoded) pattern that is already indexed in $\bar{N}$ (as an integer-encoded pattern). We note that the primary reason for encoding patterns in binary was to accurately model multiplication; a side benefit is that we can easily exclude patterns from being re-generated.

We also need to do something to overcome symmetry within $\hat{N}$ (when $|\hat{N}| > 1$). One idea to overcome this is to order the implied integer-encoded patterns lexically, by imposing the $|\hat{N}|-1$ *lexicographic constraints*:

$$\sum_{i\in M}\sum_{l\in L_i} \epsilon^i 2^l z_l^{i1} \leq \sum_{i\in M}\sum_{l\in L_i} \epsilon^i 2^l z^{i2} \leq \cdots \leq \sum_{i\in M}\sum_{l\in L_i} \epsilon^i 2^l z^{i|\hat{N}|} \ , \tag{30}$$

for some $\epsilon > 0$. Choosing ϵ sufficiently small insures a true lexical ordering, but this causes numerical problems. A more moderate choice of ϵ seems to be sufficient to break enough of the symmetry without causing numerical difficulties. More sophisticated approaches for dealing with this symmetry would be to try to apply various ideas of Lee and Margot [16, 19, 18, 20, 21]. This would be particularly easy to carry out considering that we will take $|\hat{N}|$ to be small.

Next, we turn to the issue of sizing $\hat{N}$. We could take the extreme approach choosing $\hat{N} = \emptyset$, in which case we are just choosing how many times to use patterns from a fixed set that we have on hand. At the other extreme, we could let $\bar{N} = \emptyset$, and then we are really seeking to solve the full bilinear model (using binary-encoding). In the former case, we are being too restrictive. In the latter case, we are being too ambitious. So we take a practical approach, solving a sequence of models (17–30) as described in the `crawla` algorithm of Figure 2.

- *Initialization*: Start with $\bar{N} := \emptyset$. Choose ν_a and ν_s to be very small positive integers (say ≤ 3).
- *Augmentation Phase*: If we have $n_{\max}$ patterns in $\bar{N}$, then we STOP. Otherwise, if we have fewer than $n_{\max}$ patterns in $\bar{N}$, then we consider $\hat{N}$ to be of cardinality $\min\{\nu_a, n_{\max} - |\bar{N}|\}$, and we solve the model (17–30). We move the resulting optimal patterns from $\hat{N}$ (with binary encoding) into $\bar{N}$ (with integer encoding), and GOTO *Swapping Phase*.
- *Swapping Phase*: Do local search, testing whether dropping up to ν_s patterns from $\bar{N}$ and re-generating (through $\hat{N}$) via the model (17–30) offers an improvement. Repeat until there is no improvement; then GOTO *Augmentation Phase*.

Fig. 2. Algorithm `crawla`

We note that we are particularly motivated by problem instances for which (i) the demands d^i are small and for which (ii) the number of distinct patterns desired is small. For such instances, the number of variables in our ILP is manageable. Moreover, the classical approach of GG is ill suited to such instances, since it relies on rounding basic solutions of linear programs. In such solutions, (i) the rounding may produce solutions with significantly higher objective value (as a percentage) than the LP relaxation, and (ii) we may have as many as $|M|$ active patterns.

Finally, we note that our local-search approach can be seen as a type of *very-large scale neighborhood search technique* (see [1], for example).

4 Restricting β

Recall our notation that $\hat{N}$ indexes the new patterns that we seek to generate at each stage, and β is the number of bits that we allow in the binary representation of the usage x^j for a pattern indexed by each $j \in \hat{N}$. As it stands, the number of 0/1 bit variables for these usages is $\beta|\hat{N}|$, and we need to limit the number of these variables if we are to have some hope of quickly solving each of the ILPs along the way.

We have already suggested taking $|\hat{N}|$ to be quite small (no more than 3), but there is also the possibility of artificially limiting the number of 0/1 bit variables for these usages in some other way. For example, we can take β to be smaller than that required by the maximum possible usage of a single pattern. If we find a good pattern in this manner, then once it is transferred

to $\bar{N}$, it will enjoy unrestricted usage. Loosely speaking, we can regard $\hat{N}$ as 'direction seeking' and $\bar{N}$ as 'step taking'. Further in this spirit, we can allow differing numbers of bits for the various $j \in \hat{N}$. In this vein, it might be helpful to *not* break the symmetry within $\hat{N}$ so that a pattern might be allowed to occur more than once and make use of more bits for its usage.

We made some preliminary tests, varying β. We report on a representative example: a difficult randomly generated instance obtained from [25]. Specifically, we used 'DATA1' from 'type18', generated by Umetani using the problem generator of [31]; the instance has $m = 40$ and is characterized by w^i that are all greater than 1/5 of $W_{\max}$, and demands d^i as large as 205. In Figure 3 we see what happens as we let β range from 2 up through 8. We can readily see that for this data set, very good results are already achieved with β as low as 4 if we can tolerate 31 patterns — even though in the good solutions found, there are some patterns that are used close to 200 ($>> 2^4 - 1 = 15$) times. For fewer numbers of patterns allowed, there is significant benefit to being more generous with the bits.

A reasonable strategy seems to be one of dynamically varying β, starting fairly large and decreasing it as $|\bar{N}|$ increases. We adopted an adaptive version of such a strategy, which we used for the computational results reported in Section 5. Specifically, as a new column and its usage is generated, we monitored which was the highest order usage bit that was equal to one; then, at the next pass, we sized β to allow for one more bit than that. This is probably more generous with the bits than is needed, but it worked pretty well.

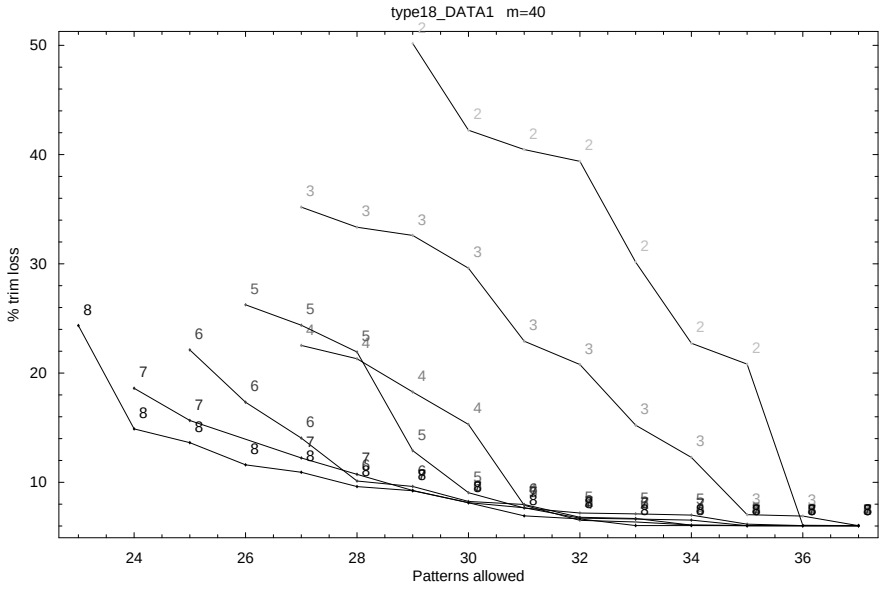

Fig. 3. Restricting β

5 Computational Experiments

We have developed an implementation using the optimization modeling/scripting software AMPL, and we have experimented using the ILP solvers CPLEX 9.0, Xpress-MP 14.27, and the open-source solver CBC (available from `www.coin-or.org`). Our implementation is quite practical for many real CSP instances. Faced with the most difficult instances or stringent running-time limits, one may regard our implementation as a prototype.

We set the search parameters $\nu_a = \nu_s = 1$, which (after experimenting initially also with $\nu_a = \nu_s = 2$) seemed to provide an adequately large search space at reasonable computational effort.

Obviously we can strengthen the formulation by using linear inequalities for each of the 0/1 knapsack constraints (18–20) (e.g., minimal cover inequalities (see Nemhauser and Wolsey [23], for example)). Also, the pattern-exclusion constraints (29) could potentially be aggregated and strengthened (see [17]). Rather than worry about strengthening the formulation in this manner ourselves, we relied on cuts of the ILP solver which we tuned through the solver's option settings.

In solving the ILPs, we did not explicitly include all of the 4-cycle inequalities (28). Rather, we adopted a "Cut&Branch" approach. We solved the LP at the root node using all of the 4-cycle inequalities. Then, we kept some number (a few hundred) of these that had the least slack at the LP optimum and proceeded to solve the resulting ILP.

In the early iterations of the augmentation phase, when $|\bar{N}|$ is quite small, the ILP may not have a feasible solution. We used a "combined Phase-I/Phase-II" methodology to get an initial feasible integer solution and then strive toward optimality. Specifically, we introduced an artificial variable into each of the demand constraints and we penalized these variables in the objective function. Since each iteration of the augmentation phase provides a feasible solution to the next iteration, after each such iteration we could permanently delete any artificial variables that had value 0. This is very simple to carry out, and it was quite beneficial in speeding up subsequent iterations.

Figures 4–6 give representative profiles of solutions for three instances. In Figure 4, a problem with 8 widths, we see that we can obtain trim loss equal to that of the GG lower bound using just 6 patterns rather than the 8 that the GG algorithm gives. Moreover, the solution obtained with just 4 patterns is not too much worse. In Figure 5, a problem with 18 widths, we see that we can obtain very good solutions using only 5 or 6 patterns. Finally, in Figure 6, we see that for a very difficult instance with 40 widths, we can already do quite well using just 29 patterns, and we can achieve a near-optimal solution using on 32 patterns.

Our primary test instances were obtained from Shunji Umetani. These 40 instances, publicly available from his webpage [25] arose at a chemical fiber company in Japan (see [28]). We aggregated demands d^i having the same width w^i — which is common in these data sets. This aggregation is important to do for our test purposes, since otherwise we will have an unfair advantage in being able to easily achieve solutions using a number of patterns equal to the number of distinct widths. Note that in practice it may not be practical to perform such aggregation due to existing post-processing systems.

We set each lower (resp., upper) demand tolerance q^i (resp., p^i) at five percent of demand, rounded down (resp., up). The first (resp. second) 20 instances have $W_{\max} = 5180$ ($W_{\max} = 9080$), and we arbitrarily set $W_{\min} = 3500$ ($W_{\min} = 7500$). In all cases, we set C_{nar} and † to large values so that the associated constraints (3–4) were not binding.

Our detailed results on the 40 chemical-fiber instances are summarized in Table 1. The columns labeled "**5180**" and "**9180**" indicate the names of the instances for the first and second 20 instances. The column labeled "**m**" indicates the number of distinct widths. We do not have a mechanism for generating optimal solutions for all choices of $n_{\max}$ as a basis of comparison, but there is the obvious lower bound of the GG lower bound, indicated in the column labeled "**GG-lb**", so we can see that we are able to generate very good solutions. A number n in the "*Absolute*" column labeled "**p%**" indicates that we have generated a solution using n patterns with percent trim loss that is within p% (difference wise) of the GG lower bound on the percent trim loss. For example, for the $m = 12$ problem **fiber19** and $W_{\max} = 5180$, the GG lower bound is 4.0547% trim loss, and we have a solution using just 6 patterns that gives trim loss less than 6.0547%. The columns labeled "*Relative*" contain the absolute numbers times 100 divided by m. These are averaged at the bottom. So we see that for the **5180** (resp., **9080**) problems, we get solutions within 2% of the GG lower bound percentage trim loss using an average of just

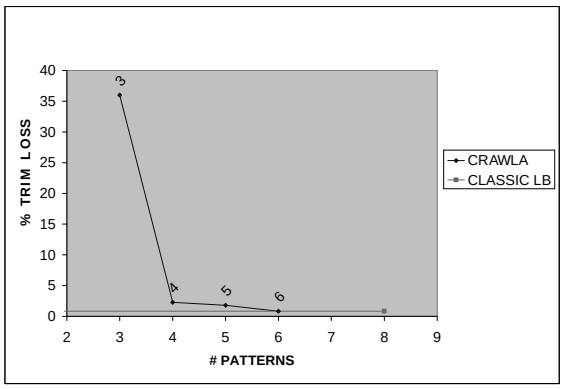

Fig. 4. 8 Widths

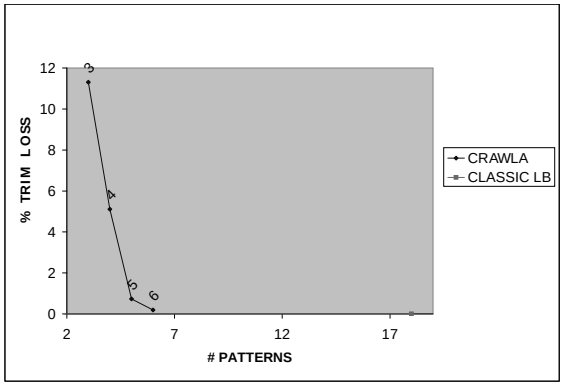

Fig. 5. 18 Widths

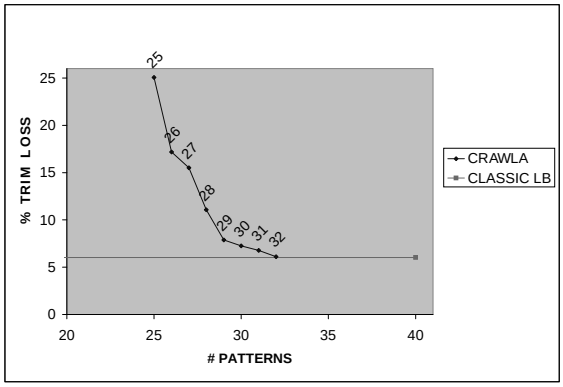

Fig. 6. 40 Widths

55% (42%) of the number of patterns that the GG method would employ. So, with a small sacrifice in trim loss, we can typically save half of the machine set ups.

With regard to running time, we have not implemented our methods with an eye to achieving the best performance possible. Our goal was to get our method to run fast enough to test out the ideas. Each profile of solutions took roughly 1 and 20 minutes, depending on the difficulty of the instance. This magnitude of running time is practical in some applied situations; for more demanding applications, what we have done may be regarded as just a prototype. In any case, using some novel methodology, we are able to generate solutions with near-optimal trim loss, using significantly fewer patterns than generated by the GG method, with modest computational effort. Moreover, we are obtaining not just a point solution, but rather a set of solutions trading off trim loss against the number of distinct patterns.

We made some further computational experiments with an eye toward comparing the quality of our solutions to those of a local-search heuristic that takes a different philosophy. More specifically, we chose as a basis of comparison, a local-search heuristic that uses pricing (i.e., dual variable) information: the ILS (Iterated Local Search) heuristic of [26]. Results for this heuristic on the same 40 chemical-fiber instances are available from Umetani's webpage. For these comparisons, we set the demand tolerances q^i and p^i to zero. We detail our results in Tables 2–4. In the tables, "**n**" refers to the number of distinct patterns in a solution, and the tabulated numbers in the columns labeled "**ILS**" and "**crawla**" are the percentage trim losses. It is not so easy to think of a good way to provide summary statistics to make a specific point. But we have highlighted certain wins for our methodology. Under "**crawla**" we have indicated with "**" runs for which the best trim loss that we obtained was lower than the best trim loss obtained by ILS (regardless of the number of patterns). Other notable data sets have a pair of numbers each adorned with a "*". Those pairs indicate situations where the best trim loss that we find is also found by ILS, but we find that trim loss with fewer (sometimes many fewer) patterns. Certainly on some of the runs, the solutions that the two methods find complement one another: **fiber09**,**11**,**13a**,**19**,**26** (**5180**), and **fiber11**,**13a** (**9080**). But even on these, our method tends to be the more effective when good solutions using very few patterns are sought. Overall, our results indicate that our methods are often very profitable on these chemical-fiber instances.

We note that the implementation of ILS runs in seconds or up to a very small number of minutes on these instances. For very demanding run-time goals, one should consider (i) making a more efficient implementation of our methods, (ii) using ILS instead, or (iii) only using our methods if the solutions of ILS are not satisfactory.

6 Extensions

Of course we could implement a Branch&Cut algorithm to solve the ILPs, rather than the Cut&Branch that we employed. But this would not be feasibly implemented within our paradigm of treating the ILP solver as a black box with a few knobs, so it would have to be the subject of a more involved computational study.

Also within a Branch&Cut framework, rather than instantiate the product variables $y_{kl}^{ij} := z_l^{ij} x_k^j$, we could implicitly account for them by applying appropriate cutting planes to the binary variables z_l^{ij} and x_k^j and the aggregate products y^{ij} defined by (16) (see [4] for the applicable cutting planes and a simple separation algorithm).

Finally, our general approach, or just parts of it, can be applied to other integer bilinear programs. It is mainly the practicality of our ideas that we have hoped to adequately demonstrate with our computational experiments (not that one algorithm will perform better than another). So it would be interesting to test these methods on other applications besides CSPs, and we hope that this paper ignites some work in that direction.

			Absolute							*Relative*						
5180	**m**	**GG-lb**	**0.5%**	**1%**	**2%**	**3%**	**5%**	**10%**	**20%**	**0.5%**	**1%**	**2%**	**3%**	**5%**	**10%**	**20%**
fiber06	6	1.3434	6	6	4	4	3	3	3	100	100	67	67	50	50	50
fiber07	4	3.7037	3	3	3	3	2	2	2	75	75	75	75	50	50	50
fiber08	4	2.8992	4	3	2	2	2	2	2	100	75	50	50	50	50	50
fiber09	7	6.2925	7	7	5	4	3	3	3	100	100	71	57	43	43	43
fiber10	6	2.4225	5	5	4	3	3	3	2	83	83	67	50	50	50	33
fiber11	7	2.1797	5	5	5	4	3	3	3	71	71	71	57	43	43	43
fiber13a	8	1.6253	6	5	4	4	3	3	3	75	63	50	50	38	38	38
fiber13b	9	0.7119	6	5	4	3	3	3	3	67	56	44	33	33	33	33
fiber14	10	0.7094	7	7	6	4	4	3	3	70	70	60	40	40	30	30
fiber15	7	2.4227	4	4	4	3	3	3	3	57	57	57	43	43	43	43
fiber16	13	0.1654	9	8	7	6	5	5	4	69	62	54	46	38	38	31
fiber17	12	1.3815	8	8	7	5	5	4	4	67	67	58	42	42	33	33
fiber18	12	0.9930	8	8	7	5	4	4	3	67	67	58	42	33	33	25
fiber19	12	4.0547	8	8	6	5	5	4	4	67	67	50	42	42	33	33
fiber20	17	0.0000	8	7	6	6	6	6	5	47	41	35	35	35	35	29
fiber23	15	0.4672	11	10	8	8	6	6	5	73	67	53	53	40	40	33
fiber26	15	1.6074	10	8	7	7	6	5	5	67	53	47	47	40	33	33
fiber28a	17	0.0046	9	8	7	6	6	6	5	53	47	41	35	35	35	29
fiber28b	19	0.0000	13	11	10	9	8	6	5	68	58	53	47	42	32	26
fiber29	19	0.0000	8	8	8	8	6	6	6	42	42	42	42	32	32	32
								Avgs:		**71**	**66**	**55**	**48**	**41**	**39**	**36**

			Absolute							*Relative*						
9080	**m**	**GG-lb**	**0.5%**	**1%**	**2%**	**3%**	**5%**	**10%**	**20%**	**0.5%**	**1%**	**2%**	**3%**	**5%**	**10%**	**20%**
fiber06	6	0.0416	4	4	3	3	2	2	2	67	67	50	50	33	33	33
fiber07	4	0.9626	2	2	2	2	2	2	1	50	50	50	50	50	50	25
fiber08	4	0.7859	3	3	3	3	2	2	2	75	75	75	75	50	50	50
fiber09	7	0.3917	5	5	5	5	4	3	2	71	71	71	71	57	43	29
fiber10	6	0.3951	5	5	3	3	2	2	2	83	83	50	50	33	33	33
fiber11	7	0.4467	5	5	4	4	3	3	3	71	71	57	57	43	43	43
fiber13a	8	0.0549	6	5	4	4	3	3	2	75	63	50	50	38	38	25
fiber13b	9	0.1067	4	4	4	4	2	2	2	44	44	44	44	22	22	22
fiber14	10	0.0000	5	3	3	3	3	3	2	50	30	30	30	30	30	20
fiber15	7	0.6861	3	3	3	3	3	2	2	43	43	43	43	43	29	29
fiber16	13	0.0000	6	6	6	6	4	4	3	46	46	46	46	31	31	23
fiber17	12	0.0000	7	6	4	4	3	3	2	58	50	33	33	25	25	17
fiber18	12	0.0000	6	5	4	4	3	3	3	50	42	33	33	25	25	25
fiber19	12	0.5954	7	6	4	4	4	3	3	58	50	33	33	33	25	25
fiber20	17	0.0000	5	4	4	4	4	3	3	29	24	24	24	24	18	18
fiber23	15	0.0000	7	7	6	5	5	4	3	47	47	40	33	33	27	20
fiber26	15	0.4087	5	5	4	4	4	4	3	33	33	27	27	27	27	20
fiber28a	17	0.0000	7	6	5	5	4	4	3	41	35	29	29	24	24	18
fiber28b	19	0.0238	7	7	6	5	5	4	4	37	37	32	26	26	21	21
fiber29	19	0.0000	7	7	6	5	5	4	4	37	37	32	26	26	21	21
								Avgs:		**53**	**50**	**42**	**42**	**34**	**31**	**26**

Table 1. Performance: Chemical Fiber Instances

5180	n	ILS	crawla
fiber06	7	5.19	
	6	5.19	
	5	5.19	
	4	8.28	8.28
	3	8.28	14.47
	2		48.5
fiber07	10	4.98	
	9	4.98	
	8	4.98	
	7	4.98	
	6	8.16*	
	3		8.16*
	2		11.34
	1		68.61
fiber08	10	4.47	
	9	4.47	
	8	4.47	
	7	4.47	
	6	4.47	
	3		4.46**
	2		5.67
fiber09	9	9.27	
	8	9.27	
	7	9.27	
	6	11.29	
	5	11.29*	
	4		11.29*
	3		23.43
	2		47.71
fiber10	14	4.20	
	13	4.20	
	12	4.20	
	11	4.20	
	10	4.20*	
	5		4.20*
	4		5.69
	3		7.18
	2		22.06
fiber11	12	6.06	
	11	6.06	
	10	4.52	
	9	4.52	
	8	6.06*	
	5		6.06*
	4		7.59
	3		10.67
fiber13a	14	2.41	
	13	4.24	
	12	2.41	
	11	4.24	
	10	4.24	
	4		13.38
	3		28.01
fiber13b	9	6.59	
	8	6.59	
	7	6.59	
	6	6.59	
	5	6.59*	
	4		6.59*
	3		10.27
	2		105.83

5180	n	ILS	crawla
fiber14	11	5.45	
	10	5.45	
	9	5.45*	
	8	7.61	
	7	9.76	
	5		5.45*
	4		9.76
	3		14.06
	2		63.56
fiber15	14	4.76	
	13	4.76	
	12	4.76	
	11	4.76*	
	10	6.56	
	4		4.76*
	3		10.17
	2		49.91
fiber16	10	7.68	
	9	6.46	
	8	8.91	
	7	8.91	4.01**
	6	27.26	5.24
	5		11.35
	4		13.80
	3		45.62
fiber17	9	5.46	
	8	6.69	
	7	7.92	
	6	17.28	4.24**
	5	22.63	6.69
	4		7.92
	3		29.99
fiber18	11	5.10*	
	10	6.16	
	9	5.10	
	8	10.41	
	7	11.47	
	6		5.10*
	5		6.16
	4		7.22
	3		18.90
fiber19	25	4.98	
	24	5.77	
	23	4.98	
	22	4.98	
	21	4.98	
	9		5.77
	8		6.56
	7		7.35
	6		8.14
	5		9.72
	4		14.45
fiber20	8	13.41	
	7	19.71	7.11**
	6	26.02	10.26
	5	54.37	22.86
	4	57.52	54.37

Table 2. Comparison: Chemical Fiber Instances (5180)

5180	**n**	**ILS**	**crawla**
fiber23	15	4.98	
	14	2.84	
	13	6.41	
	12	9.27	
	11	17.84	
	9		2.84**
	8		4.27
	7		5.70
	6		7.84
	5		19.98
fiber26	29	3.39	
	28	2.31	
	27	3.39	
	26	3.39	
	25	3.93	
	8		4.47
	7		5.01
	6		7.16
	5		15.78
	4		78.24
fiber28a	11	9.70	
	10	10.91	
	9	10.91	
	8	18.14	4.88**
	7	26.58	7.29
	6		9.70
	5		13.32
	4		28.99
fiber28b	15	5.69*	
	14	6.55	
	13	6.55	
	12	8.27	
	11	6.55	
	8		5.69*
	7		8.27
	6		15.14
	5		34.90
fiber29	13	11.03	
	12	9.40	
	11	14.30	
	10	9.40	
	9	12.66	4.50**
	8		9.40
	7		14.29
	6		24.09
	5		32.26

Table 3. Comparison: Chemical Fiber Instances (5180, continued)

9080	n	ILS	crawla
fiber06	5	8.46	
	4	8.46	
	3	8.46	8.46
	2		19.30
	1		73.53
fiber07	4	5.95	
	3	5.95	
	2	5.95	5.95
	1		17.10
fiber08	4	3.13	
	3	7.34	1.03**
	2	17.87	7.34
	1		32.60
fiber09	5	9.95	
	4	9.95	6.41**
	3	9.95	9.95
	2		27.69
fiber10	5	6.98	
	4	6.98	1.76**
	3	9.59	4.37
	2		14.81
fiber11	5	7.77	
	4	5.08	7.77
	3	10.47	10.47
	2		34.71
fiber13a	6	5.79	
	5	8.99*	
	4	12.20	8.99*
	3		15.40
	2		31.43
fiber13b	4	15.97	
	3	9.53*	
	2	22.42	9.53*
	1		138.39
fiber14	5	5.63	
	4	5.63*	
	3	43.35	5.63*
	2		24.49
fiber15	5	10.81	
	4	7.64	4.48**
	3	32.97	7.64
	2		23.47
fiber16	6	7.25	
	5	7.25	7.25
	4	13.68	9.39
	3		17.97
	2		45.86
fiber17	12	5.33	
	11	5.33	
	10	3.18*	
	5		3.18*
	4		5.33
	3		7.48
	2		26.83
fiber18	6	6.07	
	5	15.37	2.35**
	4	19.09	4.21
	3		7.93
	2		33.98
fiber19	8	3.77	
	7	6.54	
	6	6.54	
	4		3.27**
	3		6.10
	2		49.96
fiber20	7	15.97	
	6	15.97	
	5	15.97	10.45**
	4		15.97
	3		27.01
	2		137.46
fiber23	7	11.41	
	6	12.67	6.41**
	5	16.42	7.66
	4		11.41
	3		20.18
fiber26	9	6.66	
	8	7.61	
	7	8.55	
	6		1.94**
	5		3.83
	4		10.44
	3		33.09
fiber28a	8	9.88	
	7	9.88	
	6	18.33	
	5		5.66**
	4		11.99
	3		20.45
	2		60.60
fiber28b	12	3.93*	
	11	6.94	
	10	14.47	
	6		3.93*
	5		6.94
	4		14.47
	3		29.53
fiber29	13	5.90*	
	12	8.76	
	11	11.62	
	6		5.90*
	5		8.76
	4		20.21
	3		45.97

Table 4. Comparison: Chemical Fiber Instances (9080)

Acknowledgments

The author thanks Ronny Aboudi, Christoph Helmberg and Shunji Umetani for educating him about practical issues concerning CSPs, Laci Ladányi, Janny Leung, Robin Lougee-Heimer and François Vanderbeck for stimulating conversations on the CSP in general, and John J. Forrest both for his tuning of CBC and for his invisible hand. Special thanks to Dave Jensen for enabling the use of CBC from AMPL.

References

1. Ravindra K. Ahuja, Özlem Ergun, James B. Orlin, and Abraham P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1-3):75–102, 2002. Workshop on Discrete Optimization, DO'99 (Piscataway, NJ).
2. Gleb Belov and Guntram Scheithauer. The number of setups (different patterns) in one-dimensional stock cutting, 2003. Preprint, Dresden Univ., Institute for Numerical Mathematics.
3. Thomas Christof and Andreas Löbel. `www.zib.de/Optimization/Software/Porta`.
4. Don Coppersmith, Jon Lee, Janny Leung, and Oktay Günlük. A polytope for a product of real linear functions in 0/1 variables. *Revision of* IBM Research Report RC21568 (1999), 2003.
5. George B. Dantzig and Philip Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.
6. George B. Dantzig and Philip Wolfe. The decomposition algorithm for linear programs. *Econometrica*, 29:767–778, 1961.
7. Alan A. Farley and Kenneth V. Richardson. Fixed charge problems with identical fixed charges. *European Journal of Operations Research*, 18:245–249.
8. Hildegard Foerster and Gerhard Wäscher. Pattern reduction in one-dimensional cutting stock problems. *International Journal of Production Research*, 38:1657–1676, 2000.
9. Komei Fukuda. `www.ifor.math.ethz.ch/~fukuda/cdd_home/cdd.html`.
10. Paul C. Gilmore and Ralph E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9:849–859, 1961.
11. Constantine N. Goulimis. Optimal solutions for the cutting stock problem. *European Journal of Operations Research*, 44:197–208, 1990.
12. Robert W. Haessler. A heuristic programming solution to a nonlinear cutting stock problem. *Management Science*, 17(12):793–802, 1971.
13. Robert W. Haessler. Controlling cutting pattern changes in one-dimensional trim problems. *Operations Research*, 23(3):483–493, 1975.
14. Robert W. Haessler. A note on computational modifications to the Gilmore-Gomory cutting stock algorithm. *Operations Research*, 28(4):1001–1005, 1980.
15. Robert E. Johnston. Rounding algorithm for cutting stock problems. *Journal of Asian-Pacific Operations Research Societies*, 3:166–171, 1986.
16. Jon Lee. All-different polytopes. *Journal of Combinatorial Optimization*, 6:335–352, 2002.
17. Jon Lee. Cropped cubes. *Journal of Combinatorial Optimization*, 7(2):169–178, 2003.
18. Jon Lee and François Margot. On a binary-encoded ILP coloring formulation. IBM Research Report RC23324, September 2004.
19. Jon Lee and François Margot. More on a binary-encoded coloring formulation. In Daniel Bienstock and George L. Nemhauser, editors, *Integer Programming and Combinatorial Optimization (New York, 2004)*, volume 3064 of *LNCS*, pages 271–282, Berlin, 2004. Springer.
20. François Margot. Pruning by isomorphism in branch-and-cut. *Mathematical Programming, Series A*, 94(1):71–90, 2002.
21. François Margot. Exploiting orbits in symmetric ILP. *Mathematical Programming, Series B*, 98(1-3):3–21, 2003.
22. Colin McDiarmid. Pattern minimisation in cutting stock problems. *Discrete Applied Mathematics*, 98(1-2):121–130, 1999.
23. George L. Nemhauser and Laurence A. Wolsey. *Integer and combinatorial optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley, New York, 1988.
24. Manfred W. Padberg. The Boolean quadric polytope: Some characteristics, facets and relatives. *Mathematical Programming, Series B*, 45(1):139–172, 1989.

25. Shunji Umetani. `www.toyota-ti.ac.jp/Lab/Kikai/5k40/cad/umetani/index-e.html`.
26. Shunji Umetani, Mutsunori Yagiura, and Toshihide Ibaraki. An LP-based local search to the one dimensional cutting stock problem using a given number of cutting patterns. *IEICE Transactions on Fundamentals*, E86-A(5):1093–1102.
27. Shunji Umetani, Mutsunori Yagiura, and Toshihide Ibaraki. A local search approach to the one dimensional cutting stock problem to the pattern restricted one dimensional cutting stock problem. In Mauricio G.C. Resende and Jorge Pinho de Sousa, editors, *Metaheuristics: Computer Decision-Making*, pages 673–698. Kluwer, 2003.
28. Shunji Umetani, Mutsunori Yagiura, and Toshihide Ibaraki. One dimensional cutting stock problem to minimize the number of different patterns. *European Journal of Operations Research*, 146(2):388–402, 2003.
29. François Vanderbeck. Exact algorithm for minimising the number of setups in the one-dimensional cutting stock problem. *Operations Research*, 48(6):915–926, 2000.
30. Warren E. Walker. A heuristic adjacent extreme point algorithm for the fixed charge problem. *Management Science*, 22(5):587–696, 1976.
31. Gerhard Wäscher and Thomas Gau. CUTGEN1: A problem generator for the standard one-dimensional cutting stock problem. *European Journal of Operational Research*, 84:572–579, 1995.