Solving a Hard Cutting Stock Problem by Machine Learning and Optimisation

Steven D. Prestwich, Adejuyigbe O. Fajemisin(✉), Laura Climent, and Barry O'Sullivan

Department of Computer Science, Insight Centre for Data Analytics, University College Cork, Cork, Ireland
{steven.prestwich,ade.fajemisin,laura.climent, b.osullivan}@insight-centre.org

Abstract. We are working with a company on a hard industrial optimisation problem: a version of the well-known Cutting Stock Problem in which a paper mill must cut rolls of paper following certain *cutting patterns* to meet customer demands. In our problem each roll to be cut may have a different size, the cutting patterns are semi-automated so that we have only indirect control over them via a list of continuous parameters called a request, and there are multiple mills each able to use only one request. We solve the problem using a combination of machine learning and optimisation techniques. First we approximate the distribution of cutting patterns via Monte Carlo simulation. Secondly we cover the distribution by applying a k-medoids algorithm. Thirdly we use the results to build an ILP model which is then solved.

1 Introduction

The *Cutting Stock Problem* (CSP) [1] is a well-known NP-complete optimization problem in Operations Research. It arises from many applications in industry and a standard application is a paper mill. The mill produces rolls of paper of a fixed width, but its customers require rolls of a lesser width. The problem is to decide how many original rolls to make, and how to cut them, in order to meet customer demands. Typically, the objective is to minimise waste, which is leftover rolls or pieces of rolls. The problem can be modelled and solved by Integer Linear Programming (ILP), and for large instances column generation can be used.

We are working with a company on an industrial project and have encountered a hard optimisation problem. The application is commercially sensitive so we cannot divulge details, but the problem can be considered as a variant of the CSP. (We shall refer to "rolls" and "paper mills" but in fact the problem originates from another industry.) In this CSP variant, the choice of cutting pattern is semi-automated so the user has only partial control over it via a "request". A request is a vector of continuous variables so there are infinitely many possibilities, and their effect on the choice is complex. There are multiple paper mills, and each can use only one request. The rolls made by the mills are of different

A. Appice et al. (Eds.): ECML PKDD 2015, Part I, LNAI 9284, pp. 335–347, 2015.
DOI: 10.1007/978-3-319-23528-8_21

sizes even before they are cut. For each mill, either all or none of its rolls are cut. There are also demands to be met and costs to be minimised. For this paper the interest is in the application of machine learning techniques (multivariate distribution approximation and cluster analysis) to reduce this infinite nonlinear problem to a finite linear problem that can be solved by standard optimisation methods.

This paper is structured as follows. First, in Section 2 the cutting stock problem is described. Second, in Section 3, we define the framework associated with the extra difficult cutting stock problem treated in this paper. We also propose an Integer Linear Program for this problem in Section 4, and give a brief overview of an alternative metaheuristic approach. The machine learning approach presented for the problem is described in Section 5. The approaches are evaluated with a real-life application in Section 6. Finally, the conclusions are commented in Section 7.

2 Cutting Stock Problems

The cutting stock problem is a well-known optimisation problem that is often modelled as an ILP:

$$\begin{aligned} &\text{minimise} \sum_{i=1}^{n} c_i x_i \\ &\text{s.t.} \sum_{i=1}^{n} a_{ij} x_i \geq d_j \qquad \forall j \in \mathcal{M}, \forall x_i \in \mathbb{N} \end{aligned} \tag{1}$$

where $\mathcal{M}$ is the set of roll types and d_j is the demand for type j. There are n cutting patterns and x is a vector of decision variables which state how many times each pattern is used. The number of rolls of type j generated by pattern i is a_{ij}. The objective function is to minimize the total cost, where c_i is the cost associated with pattern i. The costs depend on the specifications of the problem. For instance, for some problems, such as the model described above, the costs are associated with the patterns used (e.g. some cutting machines incur certain cost), while for other problems the costs are associated with the amount of left-over material (typically called waste if it can not be sold in future orders), etc. For a literature review of cutting stock problems we recommend [2].

Variants of the CSP have been studied. The above problem is one-dimensional but two or three dimensions might be necessary [3]. The problem might be multi-stage, involving further processing after cutting [4], or might be combined with other problems, e.g. [5]. Additional constraints might be imposed because of user requirements. Widths might be continuous though restricted to certain ranges of values.

3 Problem Formalization

As mentioned above, our CSP problem has several extra difficulties from the standard CSP which makes it hard to model and solve. Instead of rolls of a fixed

original size which we can generate at will, we have a fixed number $r = 1 \dots R$ of rolls (possibly several hundred) each with its own dimensions $\sigma_r \in S$; the details of S are confidential and unimportant here, but it involves continuous values. The cutting patterns are vectors $\boldsymbol{p} \in \mathbb{N}^m$ of m integer variables, describing how many rolls of each of the m widths is cut from the original roll. However, we cannot directly choose the cutting pattern because the choice is semi-automated. We have only limited control over it via a *request* $\boldsymbol{v}$ which is a vector of n continuous variables (m and n might be different but are typically less than 20). Each v_i is restricted to the interval $[0, 1]$ and each vector is of length 1:

$$\sum_{i=1}^{n} v_i^2 = 1$$

A request $\boldsymbol{v}$ and a roll r are passed to an algorithm $\mathcal{A}$ which uses $\boldsymbol{v}$ and σ_r to select a cutting pattern $\boldsymbol{p}$. Considering the algorithm as a function $\mathcal{A} : [0, 1]^n \times S \to \mathbb{N}^m$, experiments reveal it to be quite a complex (nonlinear and discrete) function. We make no assumptions about the form of $\mathcal{A}$ (which is also confidential) and treat it as a black box. Unlike the standard CSP we have several paper mills $j = 1 \dots J$ (J might be as large as several hundred) each with its own set of R rolls (R might be different for each mill but we ignore this to simplify the description). For each mill, either all its rolls are cut into smaller rolls using the same request, or none of them are. Thus the rolls are partitioned into sets, each of which is treated as a unit and cut using the same request, though not necessarily the same cutting pattern. Finally, each mill's set of rolls has an intrinsic *value* V_j and we would like to satisfy demands using low-value rolls (in order to save the most valuable resources for future demands).

In summary, our problem is as follows. Given customer demand $\boldsymbol{d} \in \mathbb{N}^m$ for the m different roll sizes, we must select a subset of the mills with minimum total value, and a request $\boldsymbol{v}_j$ for each mill j, so that demand is met. This is an infinite nonlinear optimisation problem which we call the Continuous Semi-Automated Weighted Cutting Stock Problem (CSAWCSP):

$$\text{minimize} \sum_j V_j b_j$$

s.t.

$$\sum_{j=1}^{J} \sum_{r=1}^{R} b_j \mathcal{A}(\boldsymbol{v}_j, \sigma_r) \geq \boldsymbol{d} \tag{2}$$

$$b_j \in \{0, 1\} \qquad \forall j \tag{3}$$

$$\boldsymbol{v}_j \in [0, 1]^n \qquad \forall j \tag{4}$$

where b_j is a binary variable that is set to one iff mill j's rolls are cut.

This problem is very hard to solve because there are infinite number of possible requests $\boldsymbol{v}_j$, and because $\mathcal{A}$ is not a simple function. A metaheuristic approach is possible, searching in the space of b_j and $\boldsymbol{v}_j$ variable assignments with $\sum_j V_j b_j$

as the objective function and penalizing constraint violations, but in experiments this gave poor results. Instead we would like to transform the problem to make it solvable (at least approximately) by a standard method such as ILP.

Another possibility is to combine a metaheuristic approach with the ILP. First, the metaheuristic algorithm searches in the space of $\boldsymbol{v}_j$ for each j roll with the objective function of maximizing the similarity of the percentages of products of the pattern analyzed ($\mathcal{A}(\boldsymbol{v}_j, \sigma_r)$) with respect to the percentages of demanded products ($\boldsymbol{d}$). When the stopping criterion of the metahuristic has been reached (e.g. cut-off time), the best pattern found for each roll is provided to the ILP model. For implementing such approach, we used the metaheuristic introduced in [6], which is a Simulated Annealing Like Algorithm (SALA) called the Threshold Accepting Algorithm (TA) [7]. This metaheuristic algorithm iteratively generates new requests that are mapped by the non-linear function $\mathcal{A}$ into patterns. The objective function is to maximize the similarity of the percentages of product types obtained by a pattern with respect the percentages demanded. The results obtained with such technique were very poor: the ILP models of most of the instances evaluated in Section 6 did not have solutions (because the sum of all the unique patterns associated to each roll j did not satisfy the demand of at least one type of product); and for those few instances with solutions, their quality was far below that of the solutions found with our technique.

4 ILP Model for the CSAWCSP

To make the problem amenable to an ILP approach we reduce the infinite set of possible requests $\boldsymbol{v}_j$ to a representative finite set of requests $\boldsymbol{u}_{jk}$ ($k = 1 \ldots K$) for each mill j (we assume the same K for each mill to simplify the notation). We then precompute the total number of each roll size obtained for request k and mill j over all its rolls, storing the results in vectors of integer constants $\boldsymbol{c}_{jk} = \sum_{r=1}^{R} \mathcal{A}(\boldsymbol{u}_{jk}, \sigma_r)$ ($\forall j, k$). This eliminates the complexity of $\mathcal{A}$ and the infinite choice of requests, and we can now model the CSAWCSP as an ILP:

$$\text{minimize} \sum_j V_j b_j$$

s.t.

$$\sum_{k=1}^{K} x_{jk} = b_j \qquad \forall j \tag{5}$$

$$\sum_{j=1}^{J} \sum_{k=1}^{K} \boldsymbol{c}_{jk} x_{jk} \geq \boldsymbol{d} \tag{6}$$

$$b_j, x_{jk} \in \{0, 1\} \qquad \forall j, k \tag{7}$$

where $b_j = 1$ indicates that all mill j's rolls are cut, and $x_{jk} = 1$ indicates that they are cut using request k. If mill j is not selected then $b_j = 0$ which forces $x_{jk} = 0$ for $k = 1 \ldots K$.

This ILP can be solved by standard optimisation software. But to make this approach practical we must first select a finite set of requests $\boldsymbol{u}_{jk}$ that adequately covers all possible requests. More precisely, the possible sets of cut rolls $\boldsymbol{c}_{jk}$ must be adequately covered. This requires the generation of a finite set of vectors that approximately cover an unknown multivariate probability distribution.

5 Machine Learning Approach for the CSAWCSP

In this section we explain our approach to the problem of covering the unknown multivariate distribution in the CSAWCSP. An illustration of our approach is shown in Figure 1.

In scatter plot (a) the circle represents the hypersphere of possible requests $\boldsymbol{v}$, with a small random number of them selected shown as dots. Plot (b) shows the result of applying algorithm $\mathcal{A}$ to a mill's rolls using the different $\boldsymbol{v}$, to obtain a small set of $\boldsymbol{c}$ vectors. The space of $\boldsymbol{c}$ vectors might have a very different shape to that of the $\boldsymbol{v}$, as shown. As a consequence, a small random set of $\boldsymbol{v}$ might correspond to a very non-random small set of $\boldsymbol{c}$ vectors, showing the inadequacy of merely sampling a few requests.

Instead we sample a large number of $\boldsymbol{v}$ as shown in plot (c), with their corresponding $\boldsymbol{c}$ shown in plot (d): this represents the use of Monte Carlo simulation to approximate the distribution of the $\boldsymbol{c}$. We then select a small number of $\boldsymbol{c}$ via a k-medoids algorithm to approximately cover the estimated distribution, highlighted in plot (f). Finally we use a record of which $\boldsymbol{v}$ corresponds to which $\boldsymbol{c}$ to derive the non-random set of $\boldsymbol{v}$ highlighted in plot (e). Next we describe these phases of our approach in more detail.

5.1 Distribution Learning

Given a number of samples drawn from a distribution, the goal of distribution learning is to find the distribution from which the samples have been drawn with a high degree of certainty. Let $\mathcal{D}_n$ be a particular distribution class, and let $D \in \mathcal{D}_n$ be a distribution which has a support S, i.e. S is the range over which D is defined. To represent the probability distribution D over S, let G_D be a generator for D [10]. G_D is called a generator because, given a random input y, G_D simulates sampling from the distribution D and outputs an observation $G_D[y] \in S$.

Given independent random samples from D, as well as confidence and approximation parameters δ and ϵ respectively, the goal of any learning algorithm is to output an approximate distribution D' with a probability δ in polynomial time. The distance between this approximate distribution and the original distribution is $d(D, D')$, and can be measured in several ways. These include the Kolmogorov distance (from the Kolmogorov-Smirnoff test) [11], the Total Variation distance [12], and the Kullback-Leibler divergence [13]. When $d(D, D') \leq \epsilon$, G_D is called an ϵ-good generator.

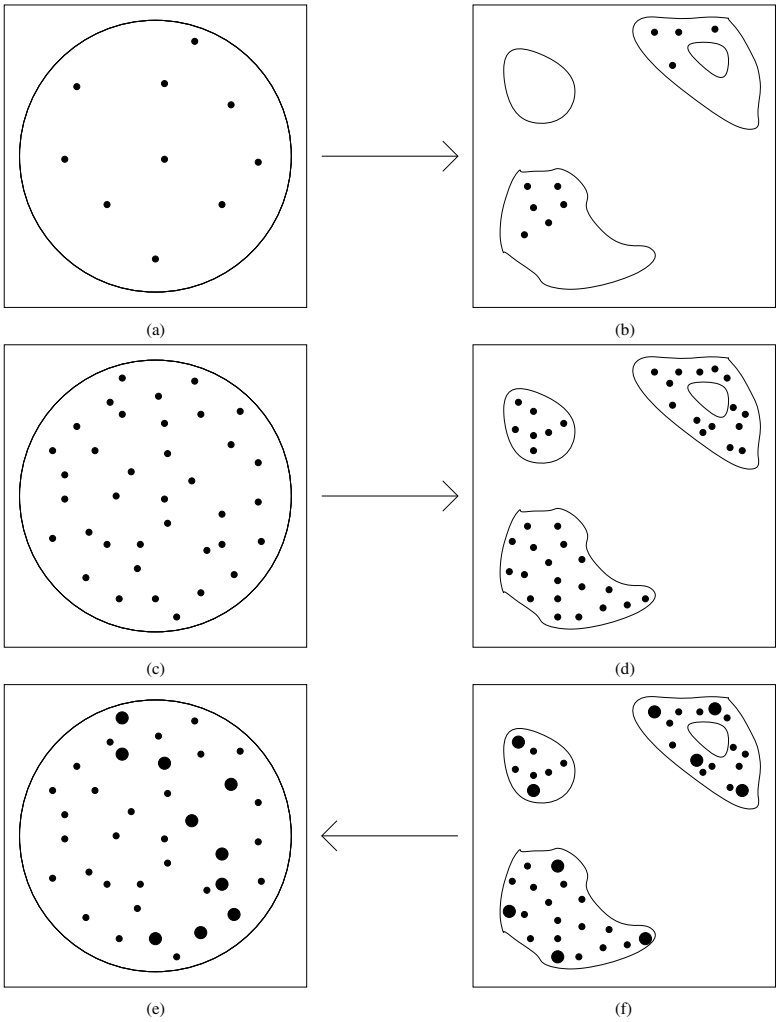

Fig. 1. Illustration of our approach

For our problem it is necessary to understand the relationship between the requests $\boldsymbol{v}$ and the corresponding vectors $\boldsymbol{c}$ which state how many of each roll is produced. Once this is known, given a set of demands for each of the different rolls, the requests required to produce the best cutting patterns for each roll can be determined. As the algorithm $\mathcal{A}$ is complex it is necessary to learn its distribution. $\mathcal{A}$ acts as the generator G_D.

We approximated the distribution of $\mathcal{A}$'s output using Monte Carlo sampling, by generating a large number of random request vectors $\boldsymbol{v}$ and obtaining corresponding vectors $\boldsymbol{c}$. To avoid bias we generated $\boldsymbol{v}$ by uniform sampling from a vector space, as opposed to simply randomising each component of $\boldsymbol{v}$.

5.2 Generating Uniformly Distributed Random Vectors

To generate a random vector $\boldsymbol{Y}$ that is uniformly distributed over an irregular n-dimensional region G, an acceptance-rejection method may be used [14].

For a regular region W, where W may be multidimensional in nature, we first of all generate a random vector $\boldsymbol{X}$, which is uniformly distributed in W. If $\boldsymbol{X} \in G$, we accept $\boldsymbol{Y} = \mathbf{X}$ as the random vector uniformly distributed over G. Otherwise, we reject $\boldsymbol{X}$, and generate a new random vector. In the case when G is an n-dimensional unit ball, i.e.

$$G = \left\{ x : \sum_i x_i^2 \leq 1 \right\} \tag{8}$$

we generate a uniformly distributed random vector $\boldsymbol{X} = (X_1, X_2, \ldots, X_n)^T$, and we accept $\boldsymbol{X}$ if it falls inside the n-ball. If it does not then $\boldsymbol{X}$ is rejected, otherwise it is projected onto the hypersphere. The algorithm used for this is taken from [9] and is described below.

Algorithm 1. Random Vector Generation

1. Generate n random variables $U_1, \ldots, U_n$ as iid variables from $U(0, 1)$.
2. Set $X_1 = 1 - 2U_1, \ldots, X_n = 1 - 2U_n$ and $R = \sum_{i=1}^{n} X_i^2$
3. If $R \leq 1$, accept $\boldsymbol{X} = (X_1, \ldots, X_n)^T$ as the desired vector; otherwise go to Step 1.

Using Algorithm 1, we generated a large number of request vectors $\boldsymbol{v}$ which were then passed to $\mathcal{A}$, together with σ_r $(\forall r \in R)$. $\mathcal{A}$ then returned the same number of cutting pattern vectors $\boldsymbol{c}$. For our problem this large number of patterns must be reduced to a smaller number. To do this, we used another well-known machine learning technique called k-medoids clustering.

5.3 *k*-Medoids Clustering

The k-medoids algorithm is a clustering algorithm used for partitioning a data set X into K homogeneous groups or clusters, i.e. $C = \{C_1, C_2, ..., C_K\}$ [15].

Unlike the k-means algorithm, partitioning is done around *medoids* (or *exemplars*) rather than *centroids*. This is vital for our problem because we require a small set of $\boldsymbol{c}$ that are each generated from some known $\boldsymbol{v}$. A medoid m_k is a data point in a cluster C_k which is most similar to all other points in that cluster.

A k-medoids algorithm seeks to minimize the function

$$\sum_{k=1}^{K} \sum_{i \in C_k} d\left(x_i, m_k\right) \tag{9}$$

where $d\left(x_i, m_k\right)$ is a distance metric measuring the dissimilarity between data entity x_i and the medoid m_k of the cluster [15]. Commonly used distance metrics are the Manhattan distance or Euclidean distance [16] and we use the latter. The most common algorithm for k-medoid clustering is the Partitioning Around Medoids (PAM) algorithm [15], presented at a high level in Algorithm 2.

Algorithm 2. Partitioning Around Medoids (PAM)

1. Select k out of n data entities as initial medoids $m_1, m_2, ...m_k$.
2. Assign each data entity $x_i \in X$ to cluster C_k of the closest medoid m_k as determined by $d\left(x_i, m_k\right)$.
3. For each medoid m_k, select a non-medoid y_i and swap y_i for m_k.
4. Calculate the distance $d\left(x_i, m_k\right)$
5. Select the configuration with the lowest value of $d\left(x_i, m_k\right)$.
6. Repeat steps 2 to 5 until the lowest possible value of $d\left(x_i, m_k\right)$ has been found with final medoids $m'_1, m'_2, ...m'_k$.

For very large datasets the CLARA algorithm, which is a combination of PAM and random sampling, is commonly used [17,18]. The speedup by CLARA over PAM is achieved by analysing data subsets of fixed size, which has the effect of making both computational and storage complexity linear, as opposed to quadratic [17,19]. The steps for CLARA are outlined in Algorithm 3 below.

Once the large set of cutting patterns $\boldsymbol{c}$ has been reduced to a much smaller representative set $\boldsymbol{c}'$, we can now solve an approximation to the hard cutting stock problem using ILP.

6 Emprirical Study

For empirically studying our approach, we have compared the solutions obtained by our approach for a certain range of k medoids with respect the lower optimality bounds (explained below) of several instances. For such purpose we used real data from our industrial partner. The total volume of the raw material analyzed is $1191.3m^3$ for 8 mills ($J = 8$) with each mill's rolls partitioned into a maximum of 4 different types of products. We generated and solved 20 instances of random demands in a 2.3 GHz Intel Core $i7$ processor. Monte Carlo simulation and

Algorithm 3. Clustering LARge Applications (CLARA)

1. Randomly choose a number p of sub-datasets of fixed size from the large dataset X.
2. Partition each sub-dataset into k clusters using the PAM algorithm, getting a set $M = \{m_1, m_2, ..., m_k\}$ of k medoids.
3. Associate each data entity x_i of the original large dataset to its nearest medoid m_k.
4. Calculate the mean of the dissimilarities of the observations to their closest medoid using:

$$MeanDistance(M, X) = \frac{\sum_{i=1}^{n} d\left(x_i, rep\left(M, x_i\right)\right)}{|X|} \tag{10}$$

   where:
   $d(x_i, x_j)$ is the dissimilarity between two data points x_i and x_j,
   $rep(M, x_i)$ returns the closest medoid m_k to x_i from the set M, and
   $|X|$ is the number of items in X.
5. Repeat steps 1 to 4 q times, selecting the set of medoids $M' = \{m'_1, m'_2, ...m'_k\}$ with the minimum $MeanDistance(M, X)$.

clustering were done in Java and R (using the CLARA algorithm) respectively on a 3.0 GHz Intel Xeon Processor with 8 GB of RAM. For solving the integer linear programming model, we used the CPLEX 12.6 solver with a time cut-off of 1 hour.

To approximate the unknown multivariate distribution, we generated $10,000$ random request vectors for each mill. Then, we obtained the same number of corresponding cutting patterns by applying the algorithm $\mathcal{A}$ (see Section 3). For 8 mills, this process resulted in total of $80,000$ cutting patterns. The total time required for generating all the cutting patterns was 2 hours and 30 minutes. Note that, in time-sensitive applications, a lower number of random cutting patterns could be generated to reduce this overhead. Next we applied k-medoids clustering to cover the distribution.

For evaluating the effect of the number of medoids used to cover the distribution, we varied k from 25 to 200 in steps of 25 units. The time taken to generate the clusters can be found in Table 1. In addition, Figure 2 shows the total clustering times (for all the mills). Note that the clustering times increase exponentially from 40.54 seconds for $k = 25$, to $2,651.08$ seconds ($\sim$ 44 minutes) for $k = 200$. Thus for real-life problems it is very important to make an appropriate selection of the parameter k (especially in on-line applications).

Once all the medoids were obtained we used them as input parameters for the ILP model (see Section 4). Then we solved the 20 instances of random demands. Furthermore, we also applied a relaxed ILP model with the objective of calculating the lower bound of optimality of the instances analyzed. In this variation, we consider feasible any linear combination of cutting patterns. However, it might occur that the combination selected is not feasible for the real life problem. For this reason, this measurement is a lower bound of optimality, since in the lat-

Table 1. Clustering times.

	Time (sec)							
Mill	k = 25	k = 50	k = 75	k = 100	k = 125	k = 150	k = 175	k = 200
0	5.13	14.47	31.40	56.80	107.00	158.85	229.90	310.47
1	4.94	15.68	35.66	59.88	95.85	156.30	234.19	313.49
2	4.99	14.62	27.59	56.59	94.38	154.96	235.22	313.99
3	5.06	17.59	36.63	62.06	102.29	167.75	266.71	356.86
4	5.17	16.04	33.82	60.64	99.78	164.89	240.40	331.72
5	5.43	16.83	33.04	63.48	95.85	171.37	255.47	351.97
6	4.99	15.77	33.21	61.18	98.10	163.63	241.04	327.87
7	4.83	16.27	34.37	67.03	95.04	168.46	251.94	344.71
Total Time	40.54	127.27	265.72	487.66	788.29	1306.23	1954.87	2651.08

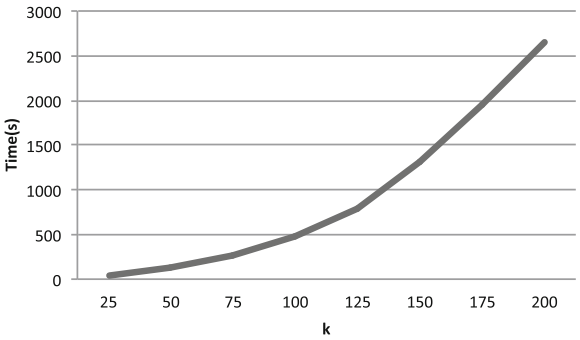

Fig. 2. Total clustering times.

ter case, the optimal solution is greater than this bound. The lower optimality bound is very useful since it allows us to stop the search for a better solution once we have reached such bound (since it can be ensured that this is the optimal solution). For this reason, we incorporated such lower optimality bound in the model solved by CPLEX.

Figure 3 shows the percentage difference between the solutions obtained with our approach and the lower optimality bound. It can be observed that as k increases, the percentage difference decreases, following an exponential inverse function. This suggest that increasing the medoids is more effective when the original medoids are fewer. Furthermore, it can be observed that there is a "saturation" point in which it is not possible to further improve the quality of the solutions. For this instance, the saturation point is located approximately at $k = 125$ since higher values of k provide almost the same results. For this reason, for these instances, the best option is to select $k = 125$, since it is not worth it to spend more time computing higher k values. Note that for this case, the difference in percentage with the lower optimality bound is $\sim 0.4\%$, which indicates that we succeeded in finding optimal and close-optimal solutions.

We also would like to comment how these percentages differences are translated into economical earnings. On average, for these instance, the raw material that was required to satisfy the demands when using $k = 25$ was almost €800 more expensive than when using $k = 125$. Needless to mention, the benefits in the real life application that will involve to use the approach presented in this paper with a great enough value of k.

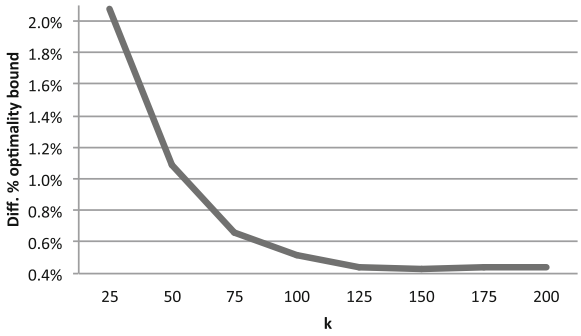

Fig. 3. Percentage Optimality Difference

In Table 2 and Figure 4 we show the mean times for solving the 20 instances for the interval of values of k selected. Note that these times also increase in a non-linear fashion, from a solution time of 1.506 seconds for $k = 25$ to 407.475 seconds ($\sim$ 7 minutes) for $k = 200$. We would like to point out that there is a correlation with the saturation point in the optimality with respect the saturation point in the computation times for solving the ILP model. Note that for higher values of $k = 125$, the increment of computation time is little appreciable.

Table 2. Mean ILP solution times.

k	Average Time (sec)
25	1.506
50	188.454
75	369.16
100	240.120
125	410.091
150	398.914
175	411.180
200	407.475

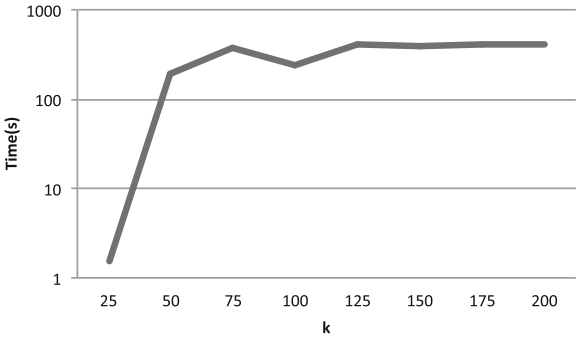

Fig. 4. Mean ILP solution times (logarithmic scale of base 10).

7 Conclusions and Future Work

In this paper, we combined two well-known machine learning techniques to solve a hard optimisation problem. This problem, which we called the CSAWCSP, arose from a real-world application and is a more complicated variant of the traditional CSP. We used Monte Carlo simulation to approximate an unknown multivariate distribution. We generated a large number of random request vectors, which were then provided to an algorithm in order to generate a corresponding number of cutting patterns. Subsequently, we applied k-medoids clustering to cover the distribution.

To study the effect of the number of medoids on the solution, we increased k steadily, and observed that there is a particular value of k (saturation point) above which, no improvements to the solution could be made. In this way, we succeeded in finding optimal and close-optimal solutions for the type of cutting stock problem analyzed in this paper. Regarding the computation time, we observed that linear increases in the value of k led to exponential increases in clustering time, and that for values above the k saturation point, the ILP solution times were relatively constant.

In the future, we aim to improve the sampling approach in order to reduce the number of request vectors needed to be generated. Furthermore, we intend to use an adaptive clustering, which might be more effective in reducing the clustering time. By applying these two new improvements, we expect to reduce computation times. This will be especially beneficial for on-line real-life applications.

Acknowledgments. This publication has emanated from research conducted with the financial support of Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289.

References

1. Kantorovich, L.V.: Mathematical methods of organizing and planning production. Management Science **6**(4), 366–422 (1960)
2. Cheng, C.H., Feiring, B.R., Cheng, T.C.E.: The cutting stock problem - a survey. International Journal of Production Economics **36**(3), 291–305 (1994)
3. Gilmore, P.C., Gomory, R.E.: Multistage Cutting Stock Problems of Two and More Dimensions. Operations Research **13**, 94–120 (1965)
4. Furini, F., Malaguti, E.: Models for the two-dimensional two-stage cutting stock problem with multiple stock size. Computers & Operations Research **40**(8), 1953–1962 (2013)
5. Hendry, L.C., Fok, K.K., Shek, K.W.: A cutting stock scheduling problem in the copper industry. Journal of the Operational Research Society **47**, 38–47 (1996)
6. Murphy, G., Marshall, H., Bolding, M.C.: Adaptive control of bucking on harvesters to meet order book constraints. Forest Products Journal and Index **54**(12), 114–121 (2004)
7. Dueck, G., Scheuer, T.: Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. Journal of computational physics **90**(1), 161–175 (1990)
8. Sawilowsky, S.S.: You think you've got trivials? Journal of Modern Applied Statistical Methods **2**(1), 218–225 (2003)
9. Kroese, D.P., Taimre, T., Botev, Z.I.: Handbook of Monte Carlo Methods, Wiley Series in Probability and Statistics. John Wiley and Sons, New York (2011)
10. Kearns, M., Mansour, Y., Ron, D., Rubinfeld, R., Schapire, R., Sellie, L.: On the Learnability of Discrete Distributions. ACM Symposium on Theory of Computing (1994)
11. Chakravarti, I.M., Laha, R.G., Roy, J.: Handbook of Methods of Applied Statistics, vol. I. John Wiley and Sons, pp. 392–394 (1967)
12. Adams, C.R., Clarkson, J.A.: On definitions of bounded variation for functions of two variables. Transactions of the American Mathematical Society **35**, 824–854 (1933)
13. Kullback, S., Leibler, R.A.: On information and sufficiency. Annals of Mathematical Statistics **22**(1), 79–86 (1951). doi:10.1214/aoms/1177729694. MR 39968
14. Rubinstein, R.Y., Kroese, D.P.: Simulation and the Monte Carlo Method, 2nd edn. John Wiley & Sons (2008)
15. de Amorim, R.C., Fenner, T.: Weighting features for partition around medoids using the Minkowski metric. In: Proceedings of the 11th International Symposium in Intelligent Data Analysis, pp. 35–44, October 2012
16. Maechler, M., Rousseeuw, P., Struyf, A., Hubert, M., Hornik, K., Studer, M., Roudier: cluster: Cluster Analysis Extended Rousseeuw et al. R package version 2.0.1, January 2015. http://cran.r-project.org/web/packages/cluster/cluster.pdf
17. Kaufman, L., Rousseeuw, P.J.: Finding Groups in Data: An Introduction to Cluster Analysis. John Wiley & Sons Inc, New York (1990)
18. Wei, C., Lee, Y., Hsu, C.: Empirical comparison of fast clustering algorithms for large data sets. In: Proceedings of the 33rd Hawaii International Conference on System Sciences (2000)
19. Nagpaul, P.S.: 7.1.2 Clustering Large Applications (CLARA). In: Guide to Advanced Data Analysis using IDAMS Software. http://www.unesco.org/webworld/idams/advguide/Chapt7_1_2.htm (access date: October 02, 2015)