Technical Report

Nr. 1/2019

Cutting Optimal Pieces from Production Items

Michael Kirchhof, Oliver Meyer, Claus Weihs

technische universität dortmund

fakultät statistik

Cutting Optimal Pieces from Production Items

Michael Kirchhof * Oliver Meyer † Claus Weihs ‡

February 4, 2019

Abstract

In the process of manufacturing various products, a larger production item is first produced and subsequently smaller parts are cut out of it. In this report we present three algorithms that find optimal positions of production pieces to be cut out of a larger production item. The algorithms are able to consider multiple quality parameters and optimize them in a given priority order. They guarantee different levels of optimality and therefore differ in their required computing time and memory usage. We assemble these algorithms with respect to each's specific benefits and drawbacks and in adaption to the given computational resources. If possible, the process is sped up by splitting the search for pieces on the whole production item into several local searches. Lastly, the approach is embedded into an application with a graphical user interface to enable its use in the industry.

1 Introduction

In many industrial applications, products are not individually manufactured, but extracted from larger production items, e.g. pieces cut from a roll of foil. In the age of Industry 4.0, modern production machines keep track of various quality parameters of the items they produce. These gathered data offer an opportunity to diminish waste and improve the products' quality by determining how many individual pieces that comply with certain quality thresholds can be cut from the production item and where the optimal cutting positions are found.

One of the use cases in which the quality of the cut products gives a huge competitive advantage is the production of long coated electrode foils. From coated foils of several hundred meters in length with varying quality, short electrodes are cut and assembled into Rechargeable Lithium-Ion Batteries. The parameters of the coated foil, like the uniformity of the eletrolyte or the residual humidity, influence the electro-chemical characteristics of the finished batteries. Another use case is found in the recycling of locally damaged products. For example, from a pipeline that has dents on a limited area, several shorter pipelines can be cut and further used as opposed to discarding the whole pipeline.

In this paper, we present three algorithms that jointly compute the optimal cutting positions. The first algorithm determines the maximum number of pieces that can be cut from a given production line, with respect to upper and lower specification limits for the observed quality parameters, by proceeding over the line from the start to the end and picking pieces as soon as possible. As this does not yield the highest quality pieces, a second algorithm originating from the 0-1 multi objective knapsack problem further reduces the set of possible piece selections by improving the pieces with respect to all quality parameters. The third algorithm simplifies the second for cases in which computational resources are limited.

*Fakultät Statistik, TU Dortmund, email: michael.kirchhof@tu-dortmund.de
†Fakultät Statistik, TU Dortmund, email: meyer@statistik.tu-dortmund.de
‡Fakultät Statistik, TU Dortmund, email: weihs@statistik.tu-dortmund.de

In the following section, the optimization problem and its constraints are defined. Next, the three algorithms as well as an approach to combine them are described. Moreover, the discretization and a way to split the problem into subproblems is outlined. In the fourth section, an overview of a graphical user interface that allows to apply these algorithms on data is given. Finally, we conclude on our findings and discuss the opportunities and restrictions of the algorithms as well as possibilities for further improvement.

Parts of this report have been submitted to the proceedings of ECDA conference 2018.

2 Optimization Problem

Along the entire length P of a production item, I quality parameters are reported periodically by the production machines. For the i-th quality parameter, the quality $q_i(p)$ of the production item at position $p, 0 \leq p \leq P$, is defined as the measurement of parameter i that is closest to p. This augments the finite number of available measurement points to a function that approximates the item's quality continuously over its entire length. For each parameter, a lower and an upper specification limit $\mathsf{LSL}_i, \mathsf{USL}_i$ are provided. This allows us to define the relative area of a piece that is outside those specification limits, hereafter simply referred to as amount of outliers, in parameter i for a piece of length l that ends at position p. This can be formalized as:

$$r_i(p) = \frac{1}{l} \int_{p-l}^{p} \mathbb{1}_{\{q_i(x) < \mathsf{LSL}_i \vee q_i(x) > \mathsf{USL}_i\}}(x) dx \,. \tag{1}$$

To take the inaccuracy of the measuring systems into account, a specific percentage of outliers $\alpha_i, 0 \leq \alpha_i \leq 1$, is allowed in the i-th quality parameter of each piece. If any quality parameter exceeds this limit, the piece is considered as not in order (n.i.o.), which is expressed by defining the cost $c_i(p)$ of parameter i for a piece that ends at position p as:

$$c_i(p) = \begin{cases} \infty & , \exists i = 1, \ldots, I : r_i(p) > \alpha_i \\ r_i(p) & , \text{else} \end{cases} . \tag{2}$$

The goal of the optimization can be split into two parts: First, the maximum number S of valid pieces that can be cut out of the production item has to be determined. Second, the sum of costs in the first quality parameter when choosing S pieces has to be minimized. If several selections of pieces lead to the same sum of costs, the second to the I-th quality parameter have to be minimized in sequential order. This can be summarized as priority based optimization. During the optimization, the pieces to be cut out are not allowed to overlap. This is expressed by the constraint that if a piece ending at p is cut out, no more pieces ending at positions between $p-l$ and $p+l$ may be picked.

3 Algorithms

3.1 Maximization of Piece Count

To detect the maximum number of valid pieces in a given production item, we use a naïve algorithm. The key idea is to start at the beginning of an item and set the possible ending p of the first piece to the required length of the pieces l. Then check whether a piece ending at that position is valid in all quality parameters. If this is the case, the count of valid pieces is increased by 1 and the ending position is increased by l. If the current piece is invalid, the position is increased by the maximum difference of measured and allowed outliers per parameter. These steps are repeated until the end of the foil is reached. The algorithm is described in pseudo code in Algorithm 1. Note that in some situations the stepsize can converge to zero, causing the algorithm to accidently stop. Thus, a minimal increment is implemented in line 9.

Algorithm 1 Naïve Count Maximization

Require: $r_1(\cdot), \ldots, r_I(\cdot)$, $\alpha_1, \ldots, \alpha_I$, P, l, I

Define $pieceCount \leftarrow 0$
Define $pos \leftarrow l$
while $pos < P$ **do**
    if $\forall i = 1, \ldots, I : r_i(p) \leq \alpha_i$ **then**
        $pieceCount \leftarrow pieceCount + 1$
        step $\leftarrow l$
    else
        step $\leftarrow \max_{i=1,\ldots,I} r_i(p) - \alpha_i$
        step $\leftarrow \max(\text{step}, 1e-16) \cdot l$
    $pos \leftarrow pos + \text{step}$
return $pieceCount$

3.2 Discretization and Splitting into Subproblems

Having found the maximum number of valid pieces, the next task is to find the positions of the pieces that minimize the amount of outliers. To make this optimization problem solvable with tolerable effort, it is discretized and – if possible – split into subproblems as described below. Note that the choice of discretization and splitting procedures has an influence on the optimization result. The procedure proposed below should therefore be considered as a (recommended) tradeoff between running time and optimality.

The discretization takes place in such a way that it is not allowed anymore to end a piece at any position p, but only at a finite number of J predefined positions $p_j, j = 1, \ldots, J$.

In order to adapt to the density of recorded measurements, we choose J as the number of measurement points. The possible ending points are each placed in the middle of two adjacent measurement points and additionally on the very last measurement position. This placement corresponds to the leaps of the quality function q_1. It forces the algorithms to end sheets at the end of constant areas of q_1 so that an i.o. or n.i.o. area is either entirely or not at all included. Furthermore, to enable the algorithms to find solutions that are at least as good as the solution of Algorithm 1, its found ending positions of pieces are also included as possible cutting points.

One drawback of the algorithms described in the following sections is that their runtime is quadratic with respect to the number of positions J. It can therefore be beneficial to divide the search area into several smaller areas and compute each's optimal selection of pieces in parallel. However, dividing the search area at certain break positions entails that no piece can overlap these positions. If pieces of the globally best solution do overlap them, the algorithms will not be able to find that solution. To minimize this risk, we use the result of Algorithm 1 to determine the break positions. In order to grant the optimization algorithms as many piece positions to choose from as possible, break positions should be placed at the end of gaps between two pieces. The reason here is as follows: Suppose there is a gap between the s-th and $(s+1)$-th piece which Algorithm 1 chose. Recalling that Algorithm 1 selects pieces as soon as possible, it must have been impossible to select piece $s+1$ earlier due to too many outliers. This implies that none of the pieces $s+1, \ldots, S$ can be shifted to a position preliminary of the start of sheet $s+1$. Thus, the search area can be restricted without a loss in quality of pieces $s+1, \ldots, S$. On the other side, this specific break position gives the biggest possible room for searching pieces $1, \ldots, s$. We currently choose every such end of a gap as break position in order to parallelize the problem as much as possible, but we are looking forward to a deeper analysis on the quality loss depending on the number and choice of breaking points.

3.3 Optimized Selection Regarding All Parameters

Discretizing the problem enables a new perspective on the problem: Each position p_j has the costs $c_i(p_j) \geq 0, i = 1, \ldots, I$, and the aim is to choose S out of the J positions in order to minimize the overall costs. Later, S will be the number of pieces found by Algorithm 1, but in general it can be chosen arbitrarily.

This is similar to the 0-1 multi objective knapsack problem (MOKP) in which a subset of J objects with different utility values and a weight has to be put into a knapsack with a given weight limit so that the sum of the chosen utility values is maximized. In the past decades, several approaches to this problem have been published using approximations (Erlebach et al, 2001) or dominance relations to compute a set of possible solutions that each can not be further improved in at least one value parameter (Bazgan et al, 2009). While this can be used to choose a priority-optimum of all non-dominated solutions a posteriori, the runtime complexity for high dimensional value parameters (I) or big numbers of objects (J) makes this approach unsuitable for our given application.

As a result, it is beneficial to point out the two main differences between the MOKP and the piece selection problem of this paper. In contrast to the MOKP, in the piece optimization the weight of each p_j is 1, and a knapsack of capacity S has to be filled. This offers a big runtime opportunity against using a generalized MOKP solver. On the other hand, the piece selection requires the further constraint that selecting a piece that ends at a position p_{j^*} prohibits selecting further pieces that end at positions p_j with a distance of $\left|p_j - p_{j^*}\right| < l$ because of the pieces' lengths.

The constraint outlined above is taken into account by creating sets of allowed predecessors $\varphi_{j,s}$. Each of these sets gives the permitted predecessors of the piece ending at position p_j when this piece and $s-1$ other pieces have to be chosen. This a priori allows to prohibit predecessors of each position according to the length of the pieces. Moreover, these sets will be used when iterating over the quality parameters to mark which piece paths are optimal in the quality parameters considered in previous iterations.

In analogy to the single objective 0-1 knapsack problem (Martello and Toth, 1990, p. 38), the cost $d_{i,s}(p_j)$ in quality parameter i of picking the piece that ends at position p_j as well as the $s-1$ best possible pieces out of the permitted preceeding pieces is defined recursively as:

$$d_{i,s}(p_j) = \begin{cases} c_i(p_j) + \min\limits_{a \in \varphi_{j,s}} (d_{i,s-1}(p_a)) & \text{, if } s > 1 \wedge \varphi_{j,s} \neq \emptyset \\ \infty & \text{, if } s > 1 \wedge \varphi_{j,s} = \emptyset \\ c_i(p_j) & \text{, else} \end{cases} . \tag{3}$$

This formula is computed in Algorithm 2 by using dynamic programming. Iterating over the number of pieces s and the current position p_j, the matrix ownCost$[j,s]$ is filled with the corresponding costs of $d_{i,s}(p_j)$ (lines 6 to 14). At the same time, the sets of valid predecessors valPred$[j,s]$ are updated so that for each piece only the predecessors with minimal costs are kept as valid predecessors. After each iteration of the outer loop, only paths of pieces that are optimal in regards to the corresponding quality parameter are left as input for the optimization of the next quality parameter (lines 15 to 23). This ensures the optimization in the given order of the quality parameters. After all quality parameters are considered, a selection of S pieces that is optimal in terms of the optimization goal is given out as optSelection (lines 24 to 30). Note that in case several selections have the same quality, this section can be modified to return a list of all of these selections, which might however entail a sharp increase of the running time.

A numerical example of the algorithm is given in Figure 1. Here, the task is to choose $S = 4$ out of $J = 9$ endpoints with $I = 2$ quality parameters and predefined costs $c_i(p_j)$. Due to the pieces' length there is the constraint that picking one endpoint prohibits to pick its direct predecessor. The two tables show the ownCost matrix in each iteration. The lines and circles show which predecessors are still valid after the corresponding iteration. After the first iteration,

Algorithm 2 Dynamic Piece Optimization

Require: valPred$[1,\ldots,J,1,\ldots,S]$, $c_1(\cdot),\ldots,c_J(\cdot)$, I, J, S

/ Iterate over all quality parameters in their priority order */*
for all quality parameters $i = 1,\ldots,I$ **do**
  Define ownCost$[1,\ldots,J][1,\ldots,S] \leftarrow \infty$
  for all piece counts $s = 1,\ldots,S$ **do**
    for all positions p_j with indices $j = 1,\ldots,J$ **do**
      if valPred$[j,s] \neq \emptyset$ **then**
        if $s = 1$ **then**
          ownCost$[j,s] \leftarrow c_i(p_j)$
        else
          / Pick the piece ending at p_j and its cheapest predecessors */*
          minCost $\leftarrow \min_{a \in \text{valPred}[j,s]}$ ownCost$[a,s-1]$
          cheapestPreds $\leftarrow \arg\min_{a \in \text{valPred}[j,s]}$ ownCost$[a,s-1]$
          ownCost$[j,s] \leftarrow c_i(p_j) +$ minCost
          valPred$[j,s] \leftarrow$ cheapestPreds
  / Delete selections that are not optimal */*
  cheapestSelections $\leftarrow \arg\min_{j=1,\ldots,J}$ ownCost$[j,S]$
  for $j \notin$ cheapestSelections **do**
    valPred$[j,S] \leftarrow \emptyset$
  / Delete subselections that have no valid successor */*
  for piece counts $s = S-1,\ldots,1$ **do**
    for all position indices $j = 1,\ldots,J$ **do**
      if $j \notin \bigcup$valPred$[\cdot,s+1]$ **then**
        valPred$[j,s] \leftarrow \emptyset$
/ Return one optimal selection of pieces */*
Declare optSelection$[1,\ldots,S]$
cheapestSelections $\leftarrow \arg\min_{j=1,\ldots,J}$ ownCost$[j,S]$
optSelection$[S] \leftarrow$ one arbitrary element of cheapestSelections
for piece counts $s = S,\ldots,2$ **do**
  optSelection$[s-1] \leftarrow$ one arbitrary element of valPred[optSelection$[s],s]$
return optSelection

there are four selections with optimal costs in the first parameter, those are the pieces ending at positions p_j with $j \in \{1,4,6,8\}$, $j \in \{2,4,6,8\}$, $j \in \{1,5,7,9\}$ or $j \in \{2,5,7,9\}$. Out of these possible selections, the pieces ending at positions $p_j, j \in \{2,5,7,9\}$, optimize the costs of the second parameter and are therefore selected in the last iteration. Their total cost is 9 in the first and 10 in the second parameter.

3.4 Optimized Selection Regarding First Parameter

Maintaining the predecessors structure in Algorithm 2 needs much memory and it requires running time to repeatedly search for the cheapest predecessor within it. However, the structure is redundant if there is only one quality parameter. In such a case, a dynamic programming approach of the single objective 0-1 knapsack problem is sufficient. The functionality is similar to Algorithm 2 and pseudocode is found in (Martello and Toth, 1990, p. 38).

This algorithm can also be used after shrinking a multi-parameter optimization problem into one surrogate quality parameter if the problem would be too complex otherwise. Because the first quality index is the most important, we propose selecting its cost c_1 as surrogate. Note that $c_1 = \infty$ if any quality parameter indicates an n.i.o. piece. Therefore,

the selected pieces still have fewer than α_i outliers in each parameter, but only the amount of outliers in the first parameter is optimized. In the following sections, this one dimensional approach with c_1 as surrogate parameter is referred to as Algorithm 3.

3.5 Combining the Algorithms

While it is desirable to apply Algorithm 2 on the entire problem to achieve a selection of pieces that is optimal in all parameters, it is sometimes not bearable due to the high demand of computing time and memory usage. To address this challenge, the algorithms are combined as visualized in Figure 2.

First, Algorithm 1 is used to detect the maximum number of pieces. Then, the problem is translated into a discretized version and if possible split into several subproblems as described in section 3.2. Note that the subproblems can be solved in parallel. Next, one of the two optimization algorithms is chosen for each subproblem. Algorithm 2 is chosen if there are multiple quality parameters and the size of the predecessors structure will not exceed the available memory. Elsewise, Algorithm 3 is applied to obtain a piece selection. The selected algorithm is handed over the number of desired pieces S by counting the pieces Algorithm 1 found in the corresponding subproblem area. In the end, the pieces found in all subproblems are combined and returned.

4 User Interface and Further Features

The approach outlined above is implemented in an application with a graphical user interface (GUI) to make it directly accessible to domain experts. In this section, we give a short overview on the GUI and some further functionalities of the application in its current state.

The GUI which handles the input of data and further parameters is shown in the top part of Figure 3. It is divided into three parts. In the first part, one or more files are inputted. To enable processing raw files, a filetype can be chosen to call a custom function to extract, transform and load the data. In the second part, the user can choose the quality parameters that were extracted from the files and set specification limits. To match the different quality parameters' positions, each quality parameter has to be associated with a position column which is also extracted in the first part. In the last part, the user can hand over more settings, in this case only the piece length. Once all fields are filled, clicking on "Analyze..." will run the algorithms described in the preceding sections. For efficient usage, the tool tries to fill in the fields automatically according to past user inputs.

The bottom half of Figure 3 shows the output presented to the user. The central element is a plot that shows the quality measurements alongside with the found pieces. This plot can also be customized. Below it, a brief summary of the analysis results is given. To allow further usage of the results, different save functions are implemented. Some of these export parts of the result, such as the plot, the summary or the sheet positions, and others further refine the result, for example by creating quality charts. As with the filetypes, additional save functions can easily be integrated to fit the application to the user's individual needs.

j	1	2	3	4	5	6	7	8	9
$c_1(p_j)$	3	3	6	2	1	0	0	4	5
$d_{1,1}(p_j)$	(3)	(3)	6	2	1	0	0	4	5
$d_{1,2}(p_j)$	∞	∞	9	(5)	(4)	2	1	4	5
$d_{1,3}(p_j)$	∞	∞	∞	∞	10	(5)	(4)	6	6
$d_{1,4}(p_j)$	∞	∞	∞	∞	∞	∞	10	(9)	(9)
$c_2(p_j)$	2	1	0	2	3	7	4	1	2
$d_{2,1}(p_j)$	2	(1)	∞	∞	∞	∞	∞	∞	∞
$d_{2,2}(p_j)$	∞	∞	∞	3	(4)	∞	∞	∞	∞
$d_{2,3}(p_j)$	∞	∞	∞	∞	∞	10	(8)	∞	∞
$d_{2,4}(p_j)$	∞	∞	∞	∞	∞	∞	∞	11	(10)

Figure 1: Example of Dynamic Piece Optimization for $I = 2$ quality parameters to be optimized, $J = 9$ possible pieces and $S = 4$ pieces to be selected. Optimal selections after each iteration are highlighted.

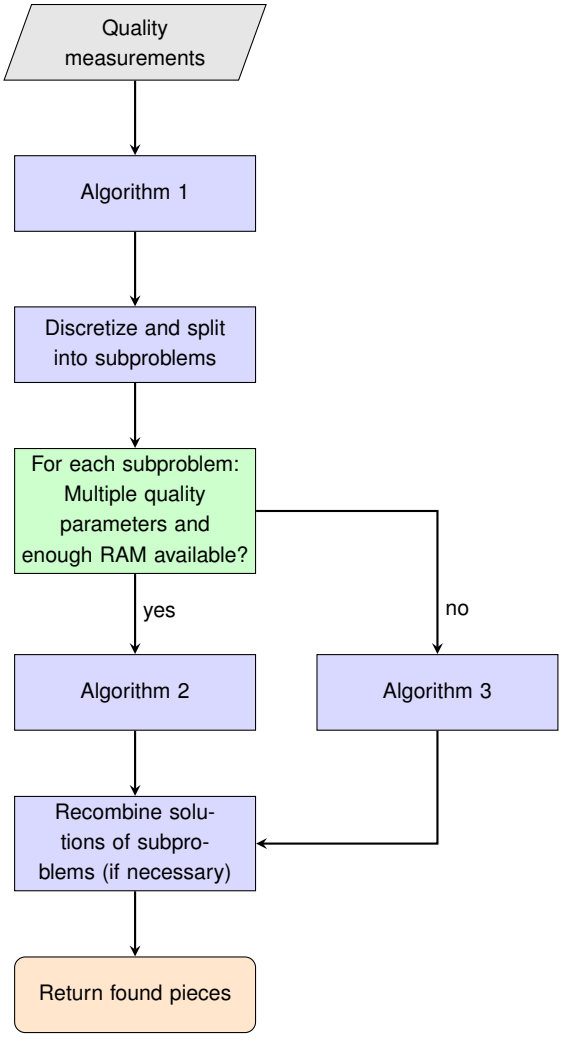
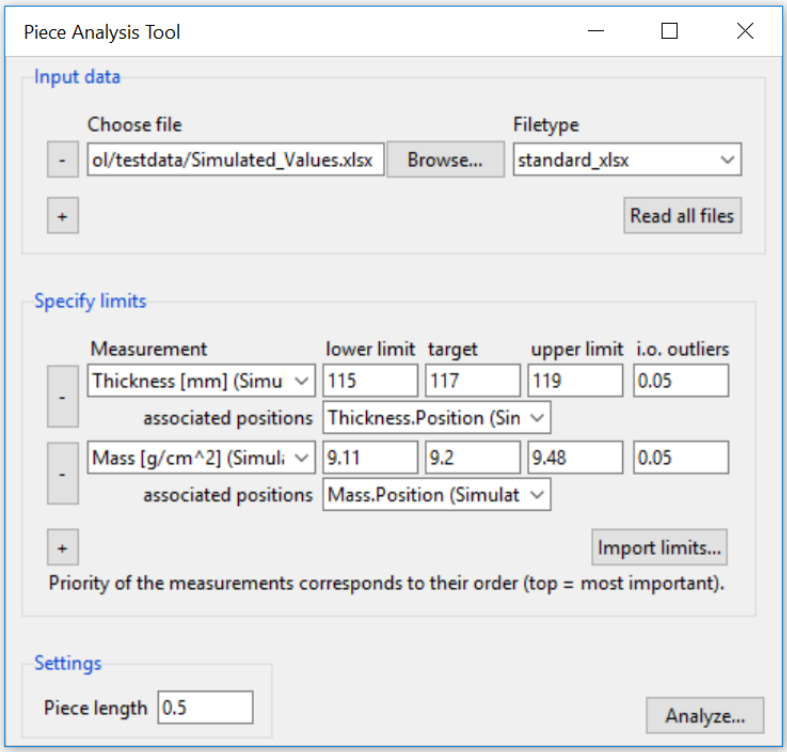


Figure 2: Combination of all algorithms as program flowchart



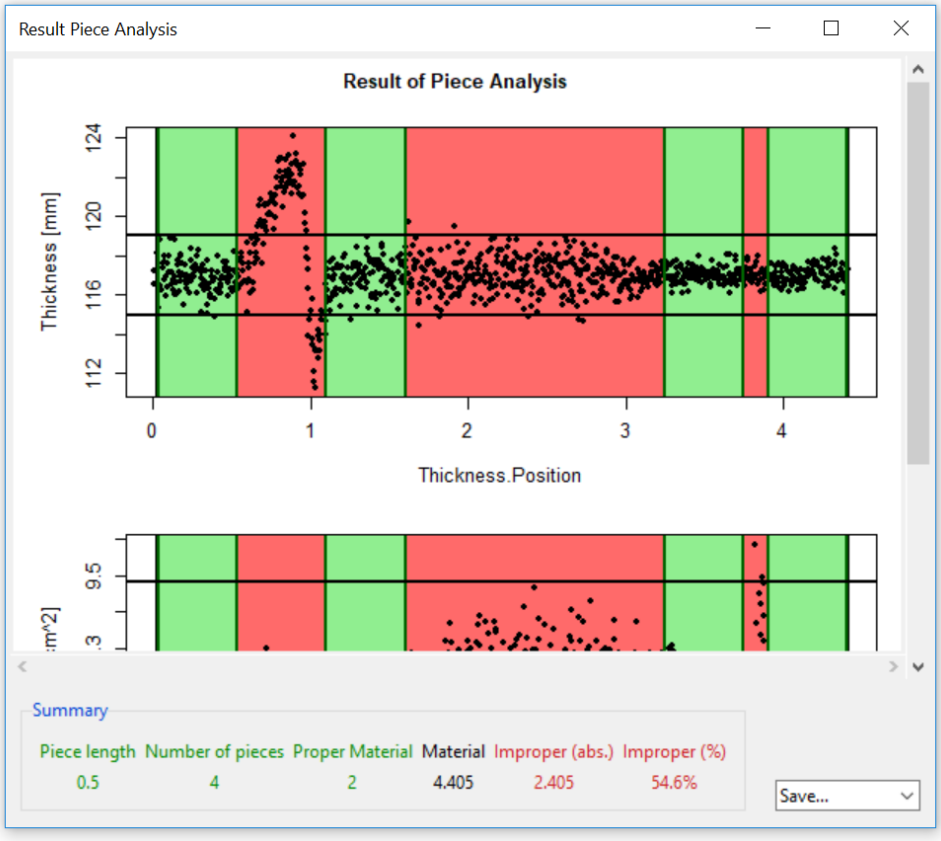


Figure 3: Input GUI (top) and Output GUI (bottom)

5 Conclusion

We presented three algorithms that search for optimal positions to cut pieces from a larger production item considering multiple quality parameters measured frequently along the length of the production item. In assemblage, they automatically adapt to the available computational resources to suggest cutting positions that lead to the highest possible quality. They are embedded in an application with a graphical user interface that can easily be extended to the accruing raw production data and to the desired output format.

A major advantage of this application is its wide area of application and its usability by domain experts apart from data scientists. One possible drawback of the tool is the discretization and division into subproblems used for speeding up the computing time. It influences the selection of pieces and might therefore lead to solely approximately optimal solutions. Depending on the degree of desired optimality and the available system resources, our proposed tradeoff may have to be adjusted. Another possible refinement can be made regarding the object that saves the predecessor structure in the second algorithm as the array that is currently used contains some dispensable entries.

We are looking forward to include indices that represent the quality of a piece not only with respect to the upper and lower specification limits, but also to the desired target value. Additionally, other summary indices may be used when it is necessary to shrink the parameters into one dimension. Lastly, we strive to further analyze the effect of dividing the problem into several subproblems on the optimality of the returned solution.

Acknowledgements
Financial support from the German Federal Ministry of Education and Research (BMBF) under the grant 03XP0076A (Projectcluster: ProZell, Project: QS-Zell) is gratefully acknowledged.

References

Bazgan C, Hugot H, Vanderpooten D (2009) Solving efficiently the 0-1 multi-objective knapsack problem. Computers & Operations Research 36(1):260 – 279, DOI 10.1016/j.cor.2007.09.009

Erlebach T, Kellerer H, Pferschy U (2001) Approximating multi-objective knapsack problems. In: Algorithms and Data Structures, Springer, Berlin, Heidelberg, pp 210–221, DOI 10.1007/3-540-44634-6_20

Martello S, Toth P (1990) Knapsack Problems: Algorithms and Computer Implementations. John Wiley & Sons, ISBN 9780471924203