Laser Cutting Path Planning using CP

Mikael Z. Lagerkvist, Martin Nordkvist, and Magnus Rattfeldt

Tomologic AB – Sweden
firstname.lastname@tomologic.com

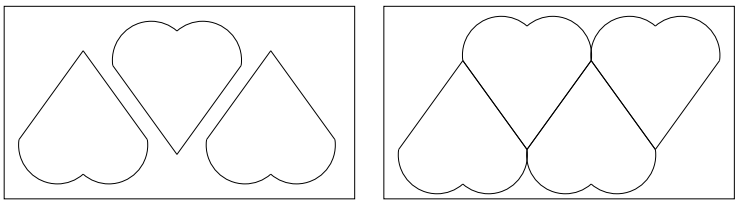
Abstract. Sheet metal cutting using lasers is ubiquitous in the industry, and is used to produce everything from home decorations to excavator scoops. Metal waste is costly for the industry, both in terms of money, but also in terms of an increased environmental footprint. Tomologic develops a unique optimisation system that can reduce this waste drastically. This paper presents a CP approach to the *Laser Cutting Path Planning Problem* (*LCPPP*), a very hard important sub problem within the Tomologic optimisation system. A solution to the LCPPP is, given a packing of some details on a metal sheet, an ordering of the cuts necessary to separate the details from the sheet. The problem is complicated by physical factors such as heat from the laser beam, or details moving or flexing. In the paper, we explain the problem in detail and present our CP approach that we developed for solving the problem. The possibility (in CP) of custom search heuristics turned out to be crucial to be able to solve the problem efficiently, as these could be made to guide the search to good first solutions.

1 Introduction

Most people have come across the problem of planning different shapes (hearts, Christmas trees, stars, etc) on gingerbread dough, and trying to minimise the dough waste that needs to be rolled out again. See Fig. 1 on the following page for an example with hearts where, in 1(a), only three hearts fit but, when aligning the hearts together as in 1(b), one more heart can be made to fit. Now, replace the dough by metal sheets, and the technology to separate the shapes (or details) from those metal sheets by laser cutting machines. Then, aligning the details as in Fig. 1(b) is not trivial anymore, and the waste cannot simply be "rolled out" again, but the recycling process is very costly.

The sheet metal cutting market is huge: the number of active laser cutting machines is estimated to be around 50,000 globally, each such machine consumes around 1,500 tonnes of raw material each year, and the amount of metal waste is typically between 20 and 50 percent [1]. So *any* (general) decrease in waste means great savings!

Tomologic develops a unique optimisation system that can reduce this global metal waste considerably, by deploying a technology that makes alignments such as those in Fig. 1(b) possible. This is of great importance not only for the manufacturing industry, for which there are obvious cost savings, but also for the whole world, since the industry's environmental footprint can be made smaller.



(a) (b)

Fig. 1. How many hearts can be obtained from the gingerbread dough?

Tomologic's solution is based on technical knowledge of, given a packing of some details on a metal sheet, how to plan the cutting paths of the laser beam to separate aligned details, and still ensuring a high quality of the end products. In this paper, we formalise this very important and hard combinatorial sub problem that must be solved within the Tomologic optimisation system, and describe a constraint programming approach that we developed for solving it. The main contributions of this paper are:

- the introduction of a new problem domain in the context of a real life industrial problem of great importance;
- a constraint programming approach for the problem, including a formal model of variables and constraints, as well as customised search heuristics for solving the model.

In the following, we first discuss background and context in Sect. 2, after which we introduce the Laser Cutting Path Planning Problem in Sect. 3. We then present our constraint programming model in Sect. 4, where we start by describing the decision variables of the problem, followed by problem constraints as well as implied ones. Section 5 describes the search heuristics and optimisation goal, and Sect. 6 gives an overview of the implementation. Finally, in Sect. 7 we discuss current status and constraint programming impacts on the application development.

2 Optimisation for Sheet Metal Cutting

One of the large problems faced by the manufacturing industry today is metal waste. This is inevitable when, out of large metal sheets, using lasers or related techniques to produce anything from home decorations to excavator scoops. Such metal waste needs to be (i) transported from the manufacturing shops to metal recycling facilities (often overseas); (ii) melt down and restored to new raw material (for example new metal sheets); (iii) transported back to the manufacturing shops for further processing. This means increased costs, both in terms of money, but also in terms of increased environmental footprints for the end products. So the objective when optimising sheet metal cutting is very easy to understand:

Given a set of production details and a number of metal sheets, find a packing of the details on the sheets that minimises the overall metal waste.

2.1 Current Technology

The traditional technology that is used for planning production details on metal sheets is *nesting* [2], where the details are planned on the sheets using two-dimensional irregular shape packing algorithms. Current state-of-the-art nesting software can produce sophisticated plans, but suffers from one important limitation:

To ensure quality of the production details, any two adjacent details must be separated by a safety distance.

This safety distance depends on the type and thickness of the metal sheets and, of course, means that large amounts of waste in the form of metal skeletons are unavoidable. For example, using the traditional nesting technology for solving the hearts problem shown in Fig. 1 on the preceding page, the solution in 1(b) is not possible, as the laser cutting machine would not be capable of cutting those aligned shapes safely.

However, by using a safety distance, the *only* condition (disregarding any optimisation criteria) that needs to be taken into account when developing nesting algorithms, is the geometric non-overlapping constraint on all details. Given any packing that fulfils this condition, the details are cut in isolation in some order, without affecting each other.

2.2 The Tomologic Optimisation System

Tomologic introduces a completely new technology for planning production details on metal sheets. This technology is based on the observation that, under some conditions, the safety distance between details can often be omitted. This means that details can be aligned and separated by the width of the laser beam only, and that cutting paths can be shared between several details. Tomologic's knowledge of when this is safe to do is based on many years of hands on experience of manual production planning for, and operation of, laser cutting machines.

However, the alignment of production details complicates the problem considerably since (i) there are many more conditions to take into account in addition to the geometric non-overlapping constraint, such as when and how two details can be aligned; and (ii) the cutting path planning is much more complicated, since the order of the cuts now depends on the packing.

Although complicating the problem, the alignment of production details also means that the waste can be reduced considerably. For example, it is often the case that waste in the form of metal skeletons (coming from the use of a safety distance) is replaced by much less waste in the form of *metal frames* (see Fig. 2 on the following page, for example). Furthermore, the alignment of production

Fig. 2. Tomologic's technology (left) compared to the traditional nesting technology (right).

details also means that sophisticated cutting patterns can be deployed, which can decrease the time and energy necessary to drive the laser beam.

So the Tomologic optimisation system must solve two interacting problems, the first one being how to find a packing of the production details on the metal sheets, while the second one being how to plan the cutting paths given such a packing of details. In this paper we focus on the second problem, that we call the Laser Cutting Path Planning Problem, presented in the next section.

3 The Laser Cutting Path Planning Problem

Given a *packing* of a set of production details on a metal sheet, the *Laser Cutting Path Planning Problem* (*LCPPP*) is the problem of finding an order of the cuts necessary to separate the details from the sheet. In order to discuss this in greater detail, we need to introduce some terminology.

A packing consists of a number of *clusters*, each such cluster contains a number of details that are connected (directly or indirectly) to each other through *alignment cuts* (two sides of different details separated by the width of the laser beam only). Such clusters are separated by a safety distance. This is in contrast with the traditional nesting technology, where each cluster can contain at most one detail. A *pocket* is an area within a cluster that is not a detail, but completely surrounded by at least two connected details.

A *cutting path* describes the movement of the laser beam while it is turned on. This is analogous to paper pencil drawing, from the time that the pencil

first touches the paper until it is lifted again. Given a cluster, we call a complete sequence (that separates each detail in the cluster from any other detail or the rest of the metal sheet) of such cutting paths a *cutting plan* for the cluster. A *piercing* is the process of creating a small hole in the metal sheet at the start of each cutting path. Due to additional heat produced by the laser beam in this process, there must be some space between piercings and details, or the details may suffer from defects. This means that after each piercing, and before starting the actual cut (that is, the cut separating the relevant detail from the rest of the sheet), there must be a short *lead-in* cut.

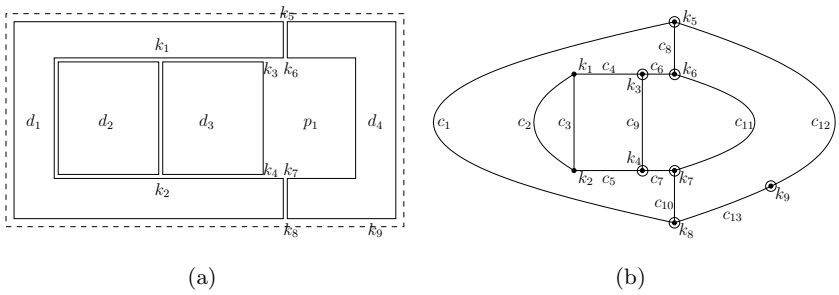
To reason about solutions to the LCPPP, we represent each cluster as a graph: the *cut graph* of the cluster. The edges of a cut graph represent cuts; either cuts separating details from the rest of the metal sheet, or cuts separating two details from each other (alignment cuts). The nodes of a cut graph represent the *connections* where two or more cuts meet (the *incoming cuts* of the connections). A cut graph is generated by identifying the cuts and connections of the cluster. In addition to natural connections that occur at the endpoints of alignment cuts, additional connections are introduced at positions that are well suited for piercings.

Example 1. Consider the instance of the LCPPP shown in Fig. 3(a) on the next page, consisting of one cluster containing four details (labeled $d_1, \ldots, d_4$), and one pocket (labeled p_1), to be separated from a metal sheet (its edges shown dashed). To separate the details from the metal sheet, thirteen cuts must be made, in some order. These cuts are labeled $c_1, \ldots, c_{13}$ in the cut graph of Fig. 3(b), and should be interpreted as follows. Cut c_1 separates d_1 from the metal sheet; alignment cut c_2 separates details d_1 and d_2 from each other; alignment cut c_3 separates details d_2 and d_3 from each other; alignment cuts c_4 and c_5 separate details d_1 and d_3 from each other; cut c_6 and c_7 separate detail d_1 from pocket p_1; alignment cuts c_8 and c_{10} separate details d_1 and d_4 from each other; cut c_9 separates detail d_3 from pocket p_1; cut c_{11} separates detail d_4 from pocket p_1. Finally, cuts c_{12} and c_{13} separate d_4 from the metal sheet.

Each cut starts and ends in two out of nine connections labeled $k_1, \ldots, k_9$ (these connections are also shown on the details in (a) for clarity). Possible cutting path starting connections are identified in the cut graph by additional circles. The connection k_9 was introduced as an additional such possible starting connection.

A possible cutting plan for this instance is: c_4 starting in k_3; $c_5 \to c_2 \to c_3$ starting in k_4; $c_8 \to c_6 \to c_9 \to c_7$ starting in k_5; $c_{10} \to c_{11}$ starting in k_8; $c_{13} \to c_1 \to c_{12}$ starting in k_9.

A *solution* to the LCPPP is a cutting plan for each cluster that separates the production details from the rest of the metal sheet, and still ensuring production reliability of those details. This is achieved by imposing additional constraints on cutting plans. We may also impose optimisation criteria on cutting plans, for example with respect to improved detail quality or lower cutting time. These constraints and optimisation criteria are discussed in the context of our constraint programming approach in the following three sections.



(a) (b)

Fig. 3. An instance of the Laser Cutting Path Planning Problem.

4 A Constraint Programming Model

4.1 Assumptions and Notation

We consider an instance of the LCPPP where cuts $\mathcal{C} = \{c_1, \ldots, c_n\}$ with connections $\mathcal{K}$ must be made to separate a number of production details from a metal sheet. For simplicity, we assume a single cluster; problems involving several such clusters are beyond the scope of this paper. Given this, we use

- *cut part* as a collective name for a detail or a pocket;
- incoming(k) to denote the incoming cuts to connection $k \in \mathcal{K}$; and
- arrays indexed by cuts as placeholders for our decision variables.

By abuse of notation we will sometimes use variable array names to denote sets or functions, and write formulas on elements, subsets, or function applications of such array names. For example, we use

- cutorder(P) to denote the cut order variables (defined below) of any of the cut parts in P; and
- cutparts(x) to denote the cut parts (at most two) that x is a cut order variable of.

This is exemplified further in Ex. 2 below.

4.2 Decision Variables and Their Domains

Cut order variables. We use an array cutorder$[c_1, \ldots, c_n]$ of *cut order variables* to represent the order in which the cuts are made, where the domain of each such variable is $1..n$. Furthermore, we let cutorder$[\bot] = -\infty$.

Cut start variables. We use an array cutstart$[c_1, \ldots, c_n]$ of *cut start variables* to represent cutting path starting points, where the domain of each such variable is the starting connections of the corresponding cut, and $\bot$ (meaning that the corresponding cut does not start a cutting path).

Predecessor variables. We use an array $\text{pred}[c_1, \ldots, c_n]$ of *predecessor variables* to represent the predecessors of the cuts, where the domain of each such variable is its adjacent cuts, and $\perp$ (meaning that the corresponding cut does not have a predecessor, since it starts a cutting path).

Example 2. Recalling the instance of Ex. 1 on page 5, the initial variable domains are as follows (only showing the domains for c_1, c_2, and c_{13}):

$$\begin{aligned}
\text{cutorder}[c_1, \ldots, c_{13}] &= \big[\,1..13, 1..13, \ldots, 1..13\,\big] \\
\text{cutstart}[c_1, \ldots, c_{13}] &= \big[\,\{k_5, k_8, \perp\}, \{\perp\}, \ldots, \{k_8, k_9, \perp\}\,\big] \\
\text{pred}[c_1, \ldots, c_{13}] &= \big[\,\{c_8, c_{10}, c_{12}, c_{13}, \perp\}, \{c_3..c_5, \perp\}, \ldots, \{c_1, c_{10}, c_{12}, \perp\}\,\big]
\end{aligned}$$

Now, the cutting plan given in Ex. 1 is equivalent to the assignments:

$$\begin{aligned}
\text{cutorder}[c_1, \ldots, c_{13}] &= [12, 3, 4, 1, 2, 6, 8, 5, 7, 9, 10, 13, 11] \\
\text{cutstart}[c_1, \ldots, c_{13}] &= [\perp, \perp, \perp, k_3, k_4, \perp, \perp, k_5, \perp, k_8, \perp, \perp, k_9] \\
\text{pred}[c_1, \ldots, c_{13}] &= [c_{13}, c_5, c_2, \perp, \perp, c_8, c_9, \perp, c_6, \perp, c_{10}, c_1, \perp]
\end{aligned}$$

Let $\text{cutorder}[c_i] = x_i$ for $1 \leq i \leq n$. The cut order variables of d_2 and $\{d_4, p_1\}$ respectively are :

$$\begin{aligned}
\text{cutorder}(\{d_2\}) &= \{x_2, x_3\} \\
\text{cutorder}(\{d_4, p_1\}) &= \{x_6, \ldots, x_{13}\}
\end{aligned}$$

The cut parts of x_1 and x_3 respectively are:

$$\begin{aligned}
\text{cutparts}(x_1) &= \{d_1\} \\
\text{cutparts}(x_3) &= \{d_2, d_3\}
\end{aligned}$$

4.3 Problem Constraints

We present the constraints first in English and then formally, possibly followed by an explanation.

Basic graph constraints. These constraints ensure that cutorder, cutstart and pred are correctly related.

(a) *Any given cut order can only be assigned once.*

$$\text{alldifferent}(\text{cutorder})$$

(b) *The cut order of a predecessor must be one less than the cut it precedes.*

$$\underset{c \in \mathcal{C}}{\forall} \text{cutorder}[c] = \text{cutorder}[\text{pred}[c]] + 1 \iff \text{pred}[c] \neq \perp$$

(c) *A starting cut must not have a predecessor.*

$$\underset{c\in\mathcal{C}}{\forall} \text{cutstart}[c] \neq \bot \iff \text{pred}[c] = \bot$$

(d) *A starting cut must have a correctly directed successor.*

$$\underset{c\in\mathcal{C}}{\forall} \Big(\text{cutstart}[c] = k \wedge k \neq \bot \Rightarrow \underset{d\in\text{incoming}(k)}{\forall} \text{pred}[d] \neq c \Big)$$

Each cut starting a cutting path in a connection k must not precede any of k's adjacent cuts. Otherwise, the cutting path would contain cuts with opposite directions (which is not possible, since a cutting path can have at most one start where it pierces the metal sheet).

Constraints ensuring production reliability. These constraints ensure that important properties from the physical reality of laser cutting are maintained.

(e) *For some sets $K \subset \mathcal{K}$ of conflicting connections, at most one of those connections can start a cutting path.*

$$\begin{gathered} \left(\underset{k\in K}{\forall} \underset{c\in\text{incoming}(k)}{\forall} b_{ck} \iff \text{cutstart}[c] = k \right) \\ \wedge \\ \text{count}(b) \leq 1 \end{gathered}$$

The counting is done using additional boolean variables.

(f) *A cut separating two cut parts must not be the final cut for both parts.*

$$\underset{x\in\text{cutorder}}{\forall} \big(|\text{cutparts}(x)| = 2 \Rightarrow \max(\text{cutorder}(\text{cutparts}(x))) > x \big)$$

For each cut order variable x that corresponds to a cut c separating two cut parts p and q, the maximum cut order for *any* cut order variable of p or q must be greater than x. Otherwise, c is the final cut for both p and q.

(g) *For some pairs of sets of cuts $A, B \subset \mathcal{C}$, all cuts of A must be cut before the final cut of B.*

$$\max(\text{cutorder}(A)) < \max(\text{cutorder}(B))$$

(h) *For some sets of adjacent cuts $A \subset \mathcal{C}$ all sharing the same connection, no more than M pairs of those cuts may pass that connection consecutively.*

$$\begin{gathered} \left(\underset{c<d\in A}{\forall} b_{dc} \iff (\text{pred}[c] = d \vee \text{pred}[d] = c) \right) \\ \wedge \\ \text{count}(b) \leq M \end{gathered}$$

The counting is done using additional boolean variables.

(i) *For some cut triplets (a, b, c) sharing a connection k, when a and b are cut consecutively, they must be cut after c.*

$$\begin{gathered} (\text{pred}[a] = b \vee \text{pred}[b] = a) \\ \Rightarrow \\ \text{cutorder}[c] < \min(\text{cutorder}[a], \text{cutorder}[b]) \end{gathered}$$

4.4 Implied Constraints

(j) *The cut order of a predecessor must be strictly less than the cut it precedes.*

$$\underset{c \in \mathcal{C}}{\forall} \text{cutorder}[\text{pred}[c]] < \text{cutorder}[c]$$

This constraint is implied by (b), and uses the property of cutorder$[\bot] = -\infty$.

(k) *For all connections with exactly three incoming cuts, at most one pair of those cuts can pass through it consecutively.*

$$\underset{k \in \mathcal{K}:|\text{incoming}(k)|=3}{\forall} \left(\begin{array}{c} \left(\underset{c<d \in \text{incoming}(k)}{\forall} b_{cd} \iff (\text{pred}[c] = d \vee \text{pred}[c] = d) \right) \\ \wedge \\ \text{count}(b) \leq 1 \end{array} \right)$$

For each connection k with exactly three incoming cuts, the number of distinct pairs of its cuts for which either is the predecessor of the other, can be at most one, since there are only three cuts in total. The counting is done using additional boolean variables for each distinct pair of cuts. This constraint is implied by the local properties around connections with three connected cuts.

(l) *The directed graph described by the predecessor variables consists of a set of simple paths.*

$$\text{mirrored} = [m_o, \ldots, m_{n-1}] \tag{1}$$

$$\underset{0 \leq i < n}{\forall} m_i = \begin{cases} x & \text{if } p_i = c_x \\ -(i+1) & \text{if } p_i = \bot \end{cases} \tag{2}$$

$$\text{alldifferent}(\text{mirrored}) \tag{3}$$

The implied constraint is a path constraint [3], and the above decomposition models the constraint. The variables used in (1) above are similar to the pred variables, the difference being that the cutting path starting point marker is indicated by unique negative values. The constraints in (2) can be implemented with element constraints since it is a total functional relation [4]. Replacing the cutting path starting point marker means that for any solution, all variables will be assigned different values, enforced by (3). This is in contrast with the pred variables, where all cuts starting paths are assigned $\bot$.

5 Optimisation and Search

Real world instances of the LCPPP can be quite large. It is not unreasonable to expect instances with thousands of variables and more than ten thousand constraints. This has several consequences for solving such instances to optimality, including high memory usage and long solving times. However, our goal is to find

a good enough solution quickly, and not to find and prove the optimal solution. If no solution is found in a reasonable time frame, we consider that particular sub problem (or cluster) to be infeasible, and discard it as a potential solution. In our context, a reasonable time frame is a few seconds of running time.

In the following sub sections, we describe the general optimisation goal for the LCPPP, and the custom search heuristics that we developed.

5.1 Optimisation Goal

The overall goal of solving an instance of the LCPPP is to find a satisfying solution that has some combination of good properties in the context of sheet metal cutting using lasers. While the details of this goal is beyond the scope of this paper, we outline some general guide lines, in their order of importance.

- *Avoiding certain cut starts.* All cutting path starting connections are not equally good, but some starting connections may lead to undesirable marks.
- *Minimising the number of cutting paths.* Starting a new cutting path takes time, since it means that the metal sheet needs to be pierced.
- *Minimising the laser movement distance.* Moving the laser head between cutting paths takes time.

The first two goals are modelled as a minimisation problem over a sum using a valuation for each cutting path starting connection. While the third goal could be expressed in a similar way, our solution handles this more softly in combination with domain specific search patterns. These search patterns come from crucial domain knowledge of sheet metal cutting using lasers. Handling the third goal in this way works fine in our current approach, but it would likely have to be handled differently if a more general search heuristic was used, such as large neighbourhood search [5].

5.2 Custom Search Heuristics

The decision variables of the model in Sect. 4 are rather low level, while they are used to describe high level concepts such as graphs and their properties. So any single assignment to a variable has a low propagation impact, since it does not meaningfully constrain the solution space. In addition, most assignments have no or next to no impact on the optimisation goal. As a consequence, using standard constraint programming search techniques, either simple ones such as fail first, or more complicated ones such as weighted degree [6] or activity based search [7], is not effective enough. As a consequence of this, in order to find good enough solutions to the LCPPP quickly, we have implemented a set of custom search heuristics comprised by what we call *actions* and *strategies*. These custom search heuristics then drive the search towards good first solutions.

In the following, a *search node* is a partially instantiated solution, and a *choice* is a set of alternatives that restrict such a search node further.

Actions, strategies, and heuristics. An *action* is a function that accepts a search node, and returns a choice, or nothing if the action does not apply to the search node.

A *strategy* is a list of actions, and a specification of how to conduct the search among these actions. The specification indicates the maximum number of dispatches of each action, if the action should create choices or assignments (single alternative choices), or any limits that should be imposed on the search. For example, by limiting the number of times an action can be dispatched to one, we can create a sub list of strategies that must succeed on its first dispatch, or fail the whole strategy upon backtracking. A strategy will run the first applicable action that is available. If no action is applicable, the strategy is finished.

A *heuristic* is a list of strategies. Given a suitable set of strategies, an overarching heuristic that guides the search to good solutions can be defined.

Example actions. We have defined over 40 different actions that perform meaningful choices for an LCPPP instance. Some examples are:

- *Extend alignment cut.* Given an open ended cutting path of alignment cuts, extend it with a successor alignment cut.
- *Start left bottom alignment cut.* Start a new cutting path of alignment cuts, choosing the left-most bottom-most possibility.
- *Start corner aligned contour.* Start a new cutting path in a graph contour corner.
- *Assign top left order.* Assign the top left unassigned cut order variable its minimum value.

The actions can roughly be classified into actions that start new cutting paths, actions that extend current cutting paths, and actions that assign cut orders. Most defined actions have some geometric meaning, and are derived from practical experience of how to plan cutting paths.

Example strategies. New strategies that implement some desired behaviour are fairly easy to define by combining lists of actions. A typical such strategy is defined by an action starting a new cutting path, followed by a sequence of actions that extend the cutting path according to different properties. We have defined 15 different strategies so far for the LCPPP.

Example heuristics. An example of a typical heuristic is as follows:

1. Build open cutting paths of alignment cuts that can be extended in both directions.
2. Assign good starting points.
3. Extend internal cutting paths from the chosen starting points.
4. Extend external cutting paths on the graph contour.
5. Assign remaining starting points, extend remaining cutting paths, and assign remaining cut orders.

When the final step is reached, the (partial) solution typically already has the interesting features defined already. This means that we are only interested in the existence of a single solution, which is similar to the radiotherapy planning [8] problem, as well as the use cases for the once-combinator [9]. At the moment, we have defined two main heuristics.

Example 3. Consider the instance of the LCPPP described in Ex. 1 on page 5, and recall that k_1 and k_2 are not possible cutting path starting connections. Following the general outline of a heuristic above, the search could perform the following steps to reach the described solution.

1. Set $\text{pred}[c_2] = c_5$ (speculative choice). Since k_1 is not a possible cutting path starting connection, set $\text{pred}[c_3] = c_2$ (all other alternatives at this point would force k_1 as a starting connection).
2. Connections k_3, k_4, k_5 and k_8 are identified as good starting points around pocket p_1.
3. Set $\text{cutstart}[c_5] = k_4$ and $\text{cutstart}[c_4] = k_3$ for paths from the pocket. Assign values for the paths $c_8 \to c_6 \to c_9 \to c_7$ starting in k_5 and $c_{10} \to c_{11}$ starting in k_8, finishing up the assigned starting points.
4. Assign the graph contour cuts, choosing k_9 (as best alternative among the available connections) as the starting point.
5. At this point, only the order remains to be set. Starting from top left among non graph contour cuts, cuts are ordered with k_4 starting the first cut, followed by cuts starting from (in order) k_3, k_5, k_8, and k_9

6 Implementation

The model and the search has been implemented using the Gecode [10, 11] constraint programming system, version 3.7.3, as a stand-alone C++ application. The Tomologic optimisation system is implemented mostly in Java and Scala. Instead of integrating the C++ code using native calls, the model is run as a separate process. This ensures full separation between the Tomologic optimisation system and the CP application.

To facilitate the communication between Java and C++, a custom XML format is used for describing instances of the problem. In addition to the instance description, the XML definition also contains a list of the strategies that should be used. Each strategy is defined with the actions it contains and the search method to be used. This allows the application code to programmatically select the overall heuristic that is to be used for a particular instance, and to run the constraint model using different search strategies on the same instance.

The full implementation, including all supporting code such as visualisation and XML parsing consists of around 5000 lines of code and 1500 lines of documentation.

Variables. All variables are simple boolean or integer variables, and the largest domains for the integer variables are bounded by the number of cuts in the instance.

Constraints and propagators. The constraints used in the model are mostly standard simple constraints. These include arithmetic, logical, counting, element, and min/max constraints. Such constraints are directly available as propagators in most constraint programming systems. The only global constraint in the model is alldifferent. After experimentation, we have concluded that the appropriate consistency level to use for the alldifferent propagation on the cutorder variables when solving LCPPP instances is bounds consistency [12].

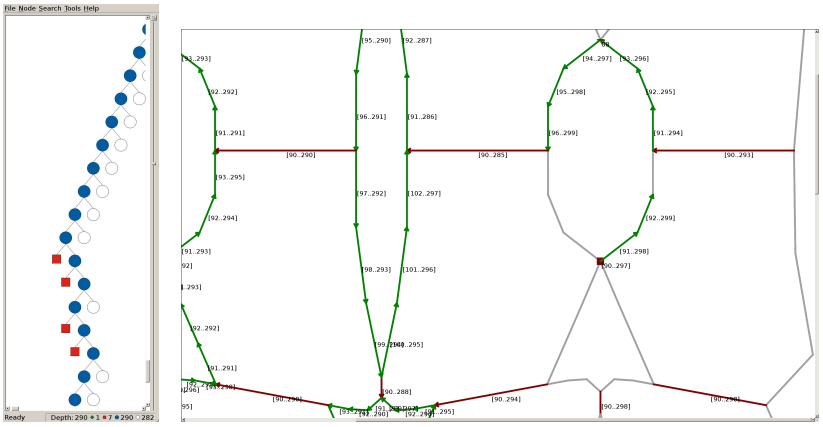
Search. The search strategies are implemented as Gecode branchers [13]. Each brancher contains a list of implementations of actions that produce descriptions of the choices to be made. To run the strategies, a custom search function is used, where a normal Gecode depth first search engine is created for each strategy that is started. Since Gecode returns a partially instantiated solution when all currently installed branchers are exhausted, that returned solution can be used as the root node for the search engine for the next strategy.

Graphical inspection support. Invaluable for understanding the search process in large instances is to have good visualisation support. The Gecode Gist search tree visualiser [14, 11] was used for understanding the search process. To understand partial solutions, a graphical visualisation was implemented that shows the currently assigned cutting paths and cut order domains. This gives a much more high level view of the current state, compared to just looking at the variables and their current domains. See Fig. 4 on the following page for an example Gist tree and visualised search state. The visualised search state shows the cutting paths under construction. For cuts that are known to be part of a specific cutting path, the cut is highlighted and their current cut order domain is displayed.

Parallel search. In most cases the search trees are quite deep, and exploration only visits a very small part of the state space; the final number of leaf nodes visited is *much* lower than the number of nodes in the explored search tree. This means that using parallel search would not be very effective, since it does not significantly speed up getting to the first solution. However, in the surrounding context of the LCPPP, the machine is fully loaded by other tasks and, hence, not using parallelism for the LCPPP is not an issue.

7 Constraint Programming Impact

Before the CP approach described in this paper was developed, we used a customised greedy algorithm for solving the LCPPP. At this time, the problem was not formalised, but based on the interaction between (non-CP) developers and our sheet metal cutting domain experts. The greedy algorithm quickly became difficult to maintain and, as more features in the form of additional constraints



(a) (b)

Fig. 4. Gist search tree (a) and visualisation of the current cutting paths (b). The cutting paths with assigned cut starts are shown as arrows indicating their directions. Green such cutting paths indicate pocket cuts, while red such cutting paths indicate alignment cuts. Grey cuts indicate cuts that do not yet belong to a cutting path. The ranges indicate the current domain for the respective cut order variables.

were introduced, the more often the algorithm had a hard time finding feasible solutions. It became clear that a more flexible and powerful approach was needed.

The formal modelling of the LCPPP that was necessary for the CP approach has been crucial for understanding the problem, and for gaining confidence in the generated solutions. Using CP as the vehicle for such a formalisation was very natural, since it allows the expression of the domain constraints and search heuristics in a reasonably high level.

Formalising, implementing, and testing a large and complicated constraint programming model such as the LCPPP requires a significant amount of time and experience with constraint technology. In total this took about four months, which included one constraint programming expert responsible full time, discussions with two sheet metal cutting domain experts, and one additional constraint programming expert, helping with constraint formulations and implementation.

In the process of formalising and understanding the LCPPP, it was possible to restructure and reimplement the previously used greedy algorithm by using an architecture inspired by the CP approach. (This was also a necessity since, during the development of the CP approach, having *some* reasonably working solution was crucial.) This has had the effect that the greedy algorithm can now handle many more cases and is much more robust, so its performance has increased drastically as a direct consequence of developing the CP approach. Due to this and to other practical reasons, the maintenance of the CP approach has stopped, and is currently not used in production. However, even though it is

not used in production anymore for solving the LCPPP, we strongly believe CP to be a key factor in the *process* leading to our current solution. Keeping this in mind, constraint programming may very well be our first approach in future applications.

Acknowledgements. We thank Magnus Gedda and Jim Wilenius for many fruitful discussions about details of the LCPPP, as well as the anonymous referees for constructive reviews.

References

1. Tomologic AB. Market Survey (2010)
2. Bennell, J.A., Oliveira, J.F.: A tutorial in irregular shape packing problems. Journal of the Operational Research Society **60**(S1) (2009) S93–S105
3. Beldiceanu, N., Carlsson, M., Rampon, J.X.: Global constraint catalog, working version as of April 24, 2013. http://www.emn.fr/z-info/sdemasse/gccat/
4. Stuckey, P.J., Tack, G.: Minizinc with functions. In Gomes, C., Sellmann, M., eds.: CP-AI-OR. Volume 7874 of LNCS., Springer (2013) 268–283
5. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In Maher, M.J., Puget, J.F., eds.: CP. Volume 1520 of Lecture Notes in Computer Science., Springer (1998) 417–431
6. Boussemart, F., Hemery, F., Lecoutre, C., Sais, L.: Boosting systematic search by weighting constraints. In de Mántaras, R.L., Saitta, L., eds.: ECAI, IOS Press (2004) 146–150
7. Michel, L., Hentenryck, P.V.: Activity-based search for black-box constraint programming solvers. In Beldiceanu, N., Jussien, N., Pinson, E., eds.: CPAIOR. Volume 7298 of Lecture Notes in Computer Science., Springer (2012) 228–243
8. Baatar, D., Boland, N., Brand, S., Stuckey, P.J.: CP and IP approaches to cancer radiotherapy delivery optimization. Constraints **16**(2) (2011) 173–194
9. Schrijvers, T., Tack, G., Wuille, P., Samulowitz, H., Stuckey, P.J.: Search combinators. Constraints **18**(2) (2013) 269–305
10. Gecode team: Gecode, the generic constraint development environment (2012) http://www.gecode.org/.
11. Schulte, C., Tack, G., Lagerkvist, M.Z.: Modeling and Programming with Gecode. (2012) Corresponds to Gecode 3.7.3.
12. López-Ortiz, A., Quimper, C.G., Tromp, J., van Beek, P.: A fast and simple algorithm for bounds consistency of the alldifferent constraint. In Gottlob, G., Walsh, T., eds.: IJCAI, Morgan Kaufmann (2003) 245–250
13. Schulte, C.: Programming branchers. In Schulte, C., Tack, G., Lagerkvist, M.Z., eds.: Modeling and Programming with Gecode. (2012) Corresponds to Gecode 3.7.3.
14. Schulte, C.: Oz explorer: A visual constraint programming tool. In Kuchen, H., Swierstra, S.D., eds.: PLILP. Volume 1140 of Lecture Notes in Computer Science., Springer (1996) 477–478