PERGAMON

Robotics and Computer Integrated Manufacturing 17 (2001) 33–47

Robotics and Computer Integrated Manufacturing

www.elsevier.com/locate/rcim

Automated design of sheet metal punches for bending multiple parts in a single setup

Ujval Alva, Satyandra K. Gupta*

Mechanical Engineering Department and Institute for Systems Research, University of Maryland, College Park, MD-20742, USA

Abstract

Sheet metal bending is a process in which bends are formed using a combination of a punch and a die. A very large number of mechanical products such as furniture panels, shelves, cabinets, housing for electro-mechanical devices, etc. are created by using the sheet metal bending process. Bending tools need to satisfy the following two criteria: (1) tools should be able to withstand bending forces, and (2) tool shapes should be such that there is no tool-part interference. In this paper, we describe a methodology for automatically designing shapes of bending punches for bending multiple parts in a single setup. We create parametric geometric models of punches. These parametric models describe the family of possible punch shapes. Using the part geometry and parametric punch shape models, we automatically generated constraints on tool parameters that eliminate the possibility of part-tool interference. We use mixed-integer techniques to identify parameters of punch shapes that result in the maximum punch strength. Finally, we perform strength analysis of the designed punch shape using finite element analysis methods, to verify that the designed punch shape is capable of withstanding the bending forces. © 2001 Elsevier Science Ltd. All rights reserved.

Keywords: Process planning; Sheet metal bending; Punch design

1. Introduction

Increasing emphasis on more personalized products and shrinking product lives is resulting in major changes in manufacturing practices. As we move towards mass customization, we will need ways to handle a wider variety of product mix on the shop floors. So far, very little attention has been paid in process planning systems to exploit commonality in tooling and fixturing across multiple parts. Most process planning systems currently handle one part at a time, attempting to find the best plan for every part. Such planners fail to identify commonality among parts and cannot select common tooling and shared fixtures that work for multiple parts. This results in more frequent tool changes and reduced overall throughput time. Most traditional process planning systems work by matching part features to existing manufacturing resources (i.e., tools and fixtures). We have developed a new two-step approach that allows us to perform tool design for multiple parts. The idea behind this approach is as follows. Rather than directly matching part features to manufacturing process, we first identify the constraints imposed by a part feature on the tooling that will be used to create that feature. In the second step, we gather all the constraints imposed by various features in various parts and perform constraint-driven tool design to identify the punch shape that works for multiple parts.

For a detailed description of sheet-metal bending processes, readers are referred to handbooks on this subject [1–4], (Fig. 1). In a typical problem, we are given a final part and a starting flat part. The flat part needs to be bent along the bend lines to create the final part. In this paper, we describe a methodology for automatically synthesizing the shapes of bending punches. Bending tools need to satisfy the following two criteria: (1) tools should be able to withstand bending forces, and (2) tool shapes should be such that there is no tool-part interference. In this paper, we describe a systematic methodology for automatically designing bending punches. We create parametric geometric models of punches. These parametric models describe the family of possible punch shapes. Using the part geometry and parametric punch shape

* Corresponding author.

E-mail addresses: ujval@eng.umd.edu (U. Alva), skgupta@eng.umd.edu (S.K. Gupta).

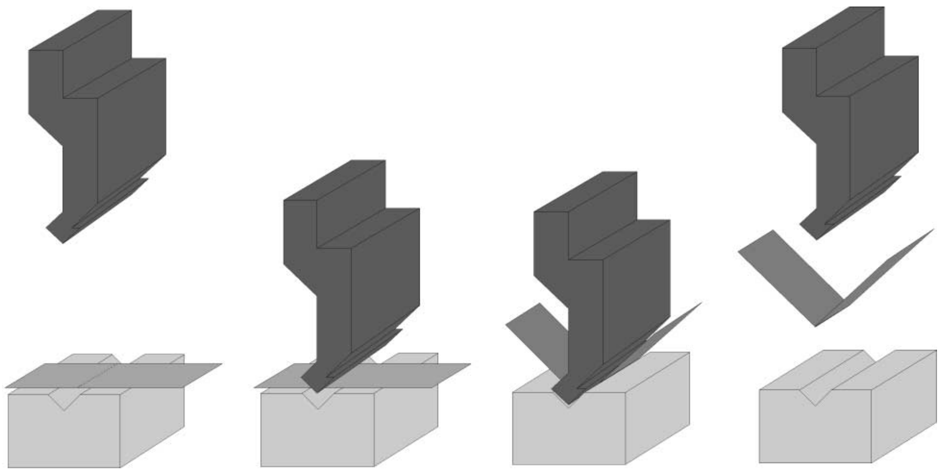


Fig. 1. Sheet metal bending.

models, we automatically generate constraints on tool parameters that eliminate the possibility of part-tool interference. We use mixed-integer programming techniques to identify parameters of punch shapes that result in the maximum punch strength. Finally, we perform strength analysis of the designed punch shape to verify that the designed punch shape is capable of withstanding the bending forces.

2. Punch design background

In order to perform satisfactorily, the punch shape should satisfy the following three criteria:

1. *Compatibility with bend geometry*: The punch radius should be compatible with the inside radius of the bend. The angle of the punch should be smaller than the bend angle.
2. *Punch strength*: The punch should be strong enough to withstand the bending forces. Bending forces depend on the part material, part thickness, and a variety of bending-related parameters. Whether a given punch shape will be able to withstand forces or not can be determined by identifying the stresses in the punch. We use the following formula to identify bending forces [5]. The formula for per unit length bending force (F) is given by,

$$F = \frac{(1.33)St^2}{L},$$

where L is the span of sheet metal, t is the sheet metal thickness, S is the nominal ultimate tensile stress of the part material. Given the bending force and the punch shape, several different techniques can be used to compute the stresses in the punch. Such possibilities include use of finite element analysis (FEA), or conservative analytical approximation of various portions of punch shapes. We are primarily concerned with the stresses in the main punch body and use FEA. Our approach to punch strength analysis will be described in Section 7.
3. *Interference*: The shape of punch should be such that there is no part-tool interference between the punch and any intermediate workpiece shape. Such interferences distort the workpiece and may cause damage to the pressbrake. Fig. 2 shows potential interference problems if the punch shape is not chosen carefully.

3. Related work

3.1. Sheet metal bending process planning

Process planning for the sheet metal bending operations involves tasks that include the selection of the tool, blank length calculation, calculation of force required to carry out the bending, bending sequence determination and other related tasks. Tool design/selection is an important part of process planning for small batch manufacturing in sheet metal bending. Representative work in the field of process planning includes work by Bourne et al. [6], Gupta et al. [7,8], de Vin et al. [9,10], Radin et al. [11], Nnaji et al. [12], Yut et al. [13] and Uzsoy [14]. Most previous work in sheet metal process planning has primarily focussed on using standard strength of material formulas and using a generate and test approach to select a punch from the database of available punches to eliminate part-tool interference. Unfortunately, to generate and test approach is time consuming and does not allow us to synthesize new punch shapes that can work for multiple parts.

3.2. Parametric shape optimization

Parametric design optimization is found to be a useful tool for the method of shape optimization. Solving the

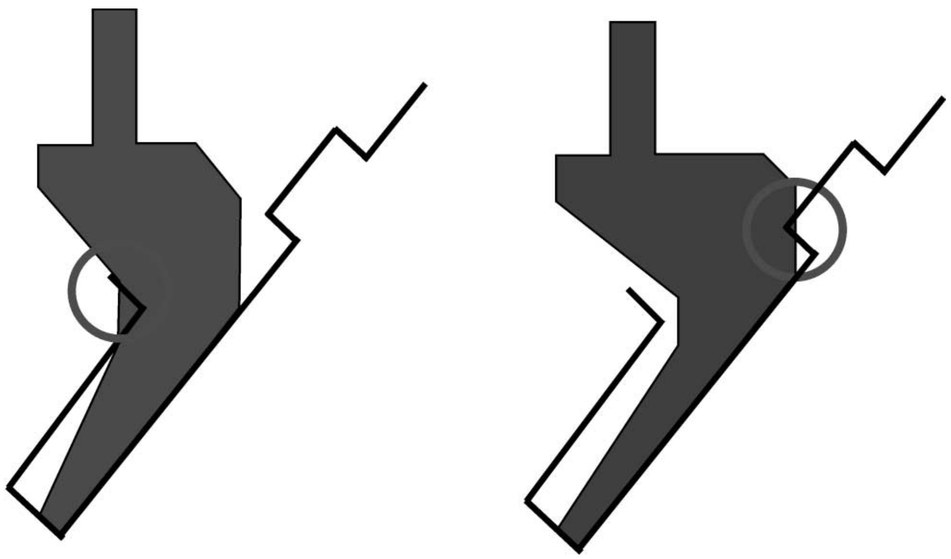


Fig. 2. Part-tool interference.

shape optimization problem involves finding the ideal shape of a body by satisfying the various constraints on its form. The input to such problems is a continuous function of one or more parameters. Using shape optimization in the early tool development process allows the tool designer to determine a near-optimal design without using time-consuming "trial and error" methods. Defining the shape of an object in terms of parameters helps in modeling the object and also the FEA of the object. Representative work that uses parametric shape optimization includes work by Braibant and Fleury [15], Rajan et al. [16], Rosen and Grosse [17], Schramm and Pilkey [18], Lindby and Santos [19], Noel et al. [20], Salagame and Belegundu [21], Zhang et al. [22], Salagame [23] and Vajna et al. [24]. Traditional parametric shape optimization techniques often utilize gradient search. Unfortunately, when we try to design punches for multiple parts, very few feasible solutions exist. Therefore, gradient search techniques do not work well for the punch design problem being addressed in this paper.

4. Overview of solution methodology

4.1. Problem statement

The problem being addressed in this paper involves identifying a common punch shape that can be used to bend multiple sheet metal parts. Traditionally, tool engineers determine a punch for each part that is to be fabricated. In the present manufacturing scenario where there are multiple parts in the product mix, the process of finding a tool for each part results in frequent tool changes. The aim of this paper is to present a punch design methodology that will help in designing a punch for bending multiple parts in a single setup.

We assume the following information is available before the process of punch design:

1. *Parametric model of the punch*. Fig. 3 shows a parametric model of a gooseneck punch. A gooseneck punch is being considered because of the flexibility it gives in terms of geometry. There are restrictions imposed on values of these parameters by the punch manufacturers. These restrictions depend on the size of the punch press, the vertical travel of the punch, the number of punch holders in the press, etc. These restrictions are the constraints that punch design should satisfy.
2. *Geometric models for a set of parts*. Currently our approach is restricted only to 2.5D parts. These types of parts are quite often referred to as sash-type parts in the sheet metal industry. The geometric models of the parts are defined completely by the dimensions of their faces and the bend angle between these faces. All the parts that are being considered in this paper have a bend angle of 90°.
3. *Operation sequences for each part in the given part set*. For each part, the operation sequence specifies the order in which the part will be bent. The sequence of bending operations is explicitly defined. This bending sequence is expressed in terms of the bend edges. Since each edge can be bent in two different orientations, the edge that is outside the press-brake is identified along with the bend line.

4.2. Overview of approach

The following approach is proposed to solve the problem described in the previous section. This approach has the following steps:

1. *Generate constraints on punch parameters*. The first step involves generating constraints on the punch

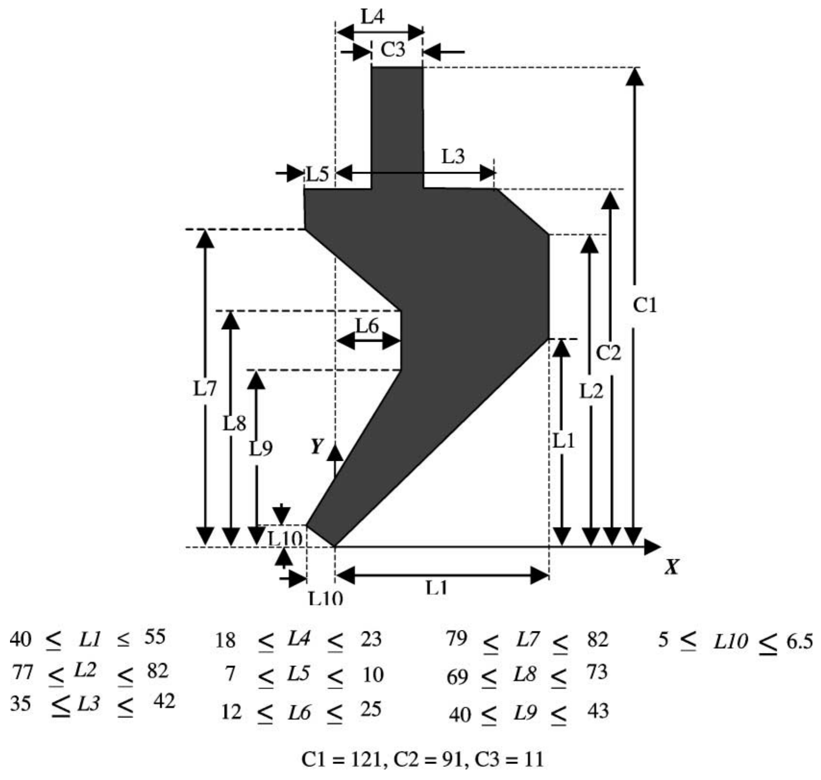


Fig. 3. Parametric representation of the gooseneck punch.

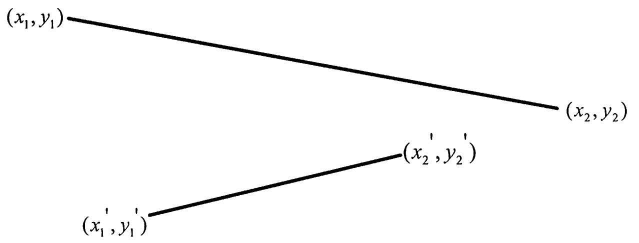


Fig. 4. Line–line intersection.

parameters by performing interference checks between the parametric punch shape and various intermediate workpiece shapes resulting during the bending process. The approach for generating intersection constraints is described in detail in Section 5.

2. *Find a punch shape that does not interfere with any intermediate workpiece shape and has the maximum strength*. As a second step, the constraints on punch parameters are used to find a punch shape that satisfies all intersection constraints that are generated while trying to maximize the punch strength at the same time. A formulation that combines mixed-integer programming and enumeration technique is used to carry out this step. The approach for this step is described in detail in Section 6.
3. *Verify that the designed punch can withstand stresses resulting from the bending forces*. It is important that the punch that is designed is able to withstand the bending forces. Finite element methods are used to determine if the resulting punch will be able to withstand the bending forces or not. The approach for this step is described in Section 7.

5. Punch parameter constraint generation

5.1. Generating constraints to eliminate interference

In order to eliminate the possibility of interference between the part shape and punch shape, none of the line segments defining the punch profile should intersect with line segments defining the workpiece profile. Therefore, to avoid interference between punch and part, we need to identify conditions that eliminate the possibility of intersection between the two line segments. Fig. 4 shows two different line segments that are defined by their end points. Let A be the line segment defined by (x_1, y_1) and (x_2, y_2). Similarly, let B be the line segment defined by (x_1', y_1') and (x_2', y_2').

Any point on the line segment A is given by

$$x = (x_2 - x_1)u + x_1,$$

$$y = (y_2 - y_1)u + y_1,$$

where, u is a parameter. For a point to be on A, $0 \leqslant u \leqslant 1$.

Similarly, any point on the line segment B is given by

$$x' = (x'_2 - x'_1)u' + x'_1,$$

$$y' = (y'_2 - y'_1)u' + y'_1,$$

where u' is a parameter. For a point to be on B, $0 \leqslant u' \leq 1$.

For intersection to occur between A and B, the intersection point (x_i, y_i) must lie on both A and B, i.e., $x = x'$ and $y = y'$. Solving for u' and u gives

$$u = \left[\frac{x'_2y'_1 - y'_2x'_1 + x_1(y'_2 - y'_1) - y_1(x'_2 - x'_1)}{(x'_2 - x'_1)(y_2 - y_1) - (y'_2 - y'_1)(x_2 - x_1)}\right],$$

$$u' = \left[\frac{x_1y_2 - y_1x_2 - x'_1(y_2 - y_1) + y'_1(x_2 - x_1)}{(x'_2 - x'_1)(y_2 - y_1) - (y'_2 - y'_1)(x_2 - x_1)}\right].$$

If the two line segments A and B are not parallel, then the two line segments will not intersect if and only if

$$u < 0, \quad \text{or}\, u > 1, \quad \text{or}\, u' < 0 \quad \text{or}\, u' > 1.$$

Let $D = (x'_2 - x'_1)(y_2 - y_1) - (y'_2 - y'_1)(x_2 - x_1)$.

This can be interpreted as given below:

1. Constraint corresponding to $u < 0$:
   If $D > 0$, then
   $x'_2y'_1 - x'_1y'_2 + x_1y'_2 - x_1y'_1 - y_1x'_2 + y_1x'_1 < 0$.
   If $D < 0$, then
   $-x'_2y'_1 + x'_1y'_2 - x_1y'_2 + x_1y'_1 + y_1x'_2 - y_1x'_1 < 0$.
2. Constraint corresponding to $u > 1$ (i.e. $-u + 1 < 0$):
   If $D > 0$, then
   $x'_2y_2 - x'_2y'_1 + x'_1y'_2 - x'_1y_2 - y'_2x_2 + y'_1x_2 < 0$.
   If $D < 0$, then
   $-x'_2y_2 + x'_2y'_1 - x'_1y'_2 + x'_1y_2 + y'_2x_2 - y'_1x_2 < 0$.
3. Constraint corresponding to $u' < 0$:
   If $D > 0$, then
   $x_1y_2 - y_1x_2 - x'_1y_2 + x'_1y_1 + y'_1x_2 - y'_1x_1 < 0$.
   If $D < 0$, then
   $-x_1y_2 + y_1x_2 + x'_1y_2 - x'_1y_1 - y'_1x_2 + y'_1x_1 < 0$.
4. Constraint corresponding to $u' > 1$ (i.e., $-u' + 1 < 0$):
   If $D > 0$, then
   $x_2y_1 - x_1y_2 + x'_2y_2 - x'_2y_1 - y'_2x_2 + y'_2x_1 < 0$.
   If $D < 0$, then
   $-x_2y_1 + x_1y_2 - x'_2y_2 + x'_2y_1 + y'_2x_2 - y'_2x_1 < 0$.

Fig. 5 shows punch profile-coordinates in terms of punch parameters. For every pair of lines on punch profile and part profile, we can write a set of constraints that eliminates the possibility of intersection between these two lines using the above described conditions. Therefore, if we consider all pairs of lines on punch profiles and part profiles, then we can generate a comprehensive set of constraints that eliminate the possibility of intersection between punch and workpiece shapes.

For the case shown in Fig. 6, taking into consideration the part line having end points (− 22.62, 22.62) and (6.36, 38.89) and the punch line having end points ($L4$, $L8$) and ($L4$, $L9$), the value of D is less than zero with the range of given punch parameters, which gives the following set of *disjunctive* constraints:

$$28.98L8 - 16.27L6 - 1023.55 < 0,$$

$$16.27L6 - 28.98L9 + 1022.87 < 0,$$

$$22.62(L8 - L9) + L6L8 - L6L9 < 0,$$

$$6.36(L8 - L9) + L6L9 - L6L8 < 0.$$

Since these constraints are disjunctive in nature, in order to eliminate the possibility of intersection, atleast one of these constraints should be satisfied.

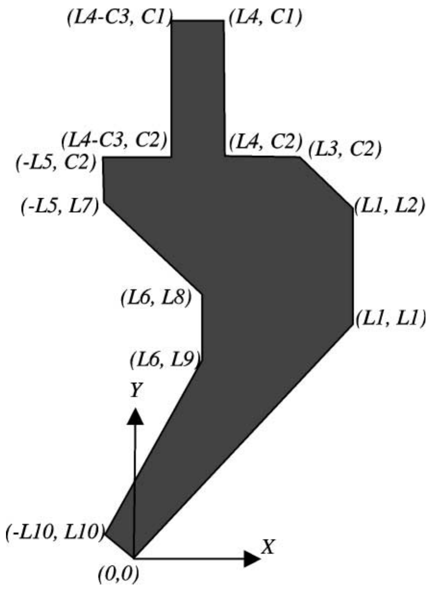


Fig. 5. Punch coordinates in terms of punch parameters.

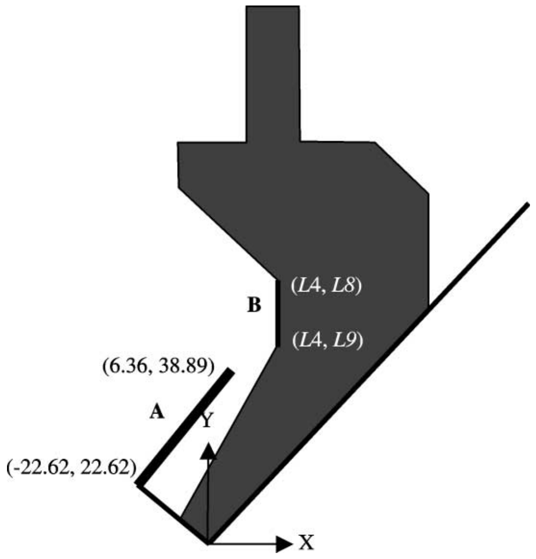


Fig. 6. An example of constraint generation.

5.2. Heuristics to eliminate unnecessary constraints

The punch parameter constraint generation method discussed in Section 5.1 involves automatically generating constraints for eliminating intersection between every line of the part with every line of the punch. This process may result in a very large number of constraints. This results in a time-consuming optimization process. In order to reduce the number of constraints, two heuristics can be applied to remove those constraints that are redundant.

- *Profile partitioning heuristics*. The first heuristic that eliminates redundant constraints involves dividing the punch and part profiles into two segments. The extreme points on a punch divide the punch profile into two profile segments as shown in Fig. 7. Similarly the part profile can also be divided into two profile segments at the bend. The division of the profiles of the punch and part into segments helps in reducing the number of constraints in the following manner. The part-profile segments on side A need to be checked for interference with only the punch-profile segments on side A as can be seen in Fig. 7. Similarly, the part-profile segments on side B need to be checked for interference with only the punch-profile segments on side B. This heuristic is based on the following observation. Any part-profile line segment on side A will intersect with the punch-profile line segment on side A before intersecting with the punch-profile line segment on side B. Therefore, if there is an intersection between the part-profile segment and punch-profile segment on side A, then intersection check between the part-profile segment on side A and the punch-profile segment on side B need not be carried out. This method reduces the total number of constraints that need to be included in the optimization process. If the punch and the part profiles are divided equally into two profile segments, then the reduction in the number of constraints is 50%.
- *Heuristics for identifying redundant constraints*. The second heuristic is to eliminate constraints that correspond to those pairs of line segments that will never intersect. This is done to ensure that all constraints that play no role in determining the punch parameters are eliminated. From the conditions described in Section 5.1, two line segments will intersect if the values of u and u' lie within the range of $(0, 1)$. Similar to the non-intersection conditions described in the Section 5.1, intersection conditions can also be written, that must be satisfied by the two line segments in order for them to intersect. Such intersection conditions are then used to eliminate redundant constraints. If a line segment on the workpiece and a line segment on a punch does not satisfy intersection condition for all values in the given punch parameters' range, then it can be safely assumed that these two line segments will never intersect in that range. Hence, all constraints that would otherwise have been generated can now be safely eliminated. The parameters that define the punch only have a certain range of values based on manufacturer specifications. For example, consider two line segments as shown in Fig. 8. The first line segment is defined by end points (20, 30) and (30, 30). Similarly, let the second line segment be defined by end points (x, y_1) and (x, y_2). If the parameter range of x is $40 \leqslant x \leqslant 50$, then it can be concluded that the two line segments will never intersect. This heuristic when used in tandem with the profile partitioning heuristic helps eliminate a very large percentage of the redundant disjunctive constraints that were originally generated. From the implementation of this heuristic, it was observed that there was approximately 75% reduction in the number of constraints by using these two heuristics. This reduction in constraints helps in making the punch-design process more tractable and easier to solve.

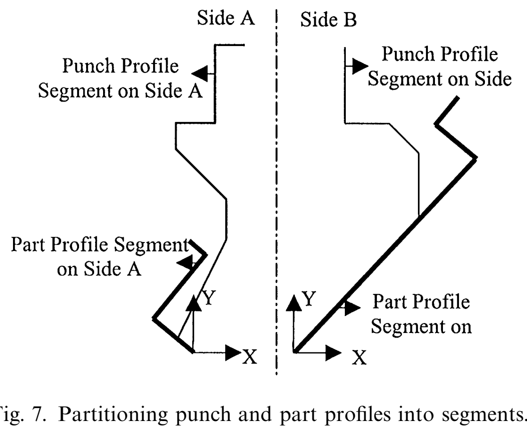

Side A
Side B
Punch Profile Segment on Side A
Punch Profile Segment on Side
Part Profile Segment on Side A
Part Profile Segment on
Y
X
Y
X

Fig. 7. Partitioning punch and part profiles into segments.

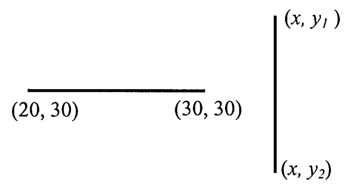


Fig. 8. Heuristic for identifying redundant constraints.

6. Synthesizing tool shape using tool parameter constraints

Section 5.1 describes the mathematical condition that ensures that two line segments do not intersect. These conditions result in constraints containing the parameters that define the two line segments. From the constraint equations, it is clear that for any two line segments to satisfy the non-intersection criteria, atleast one of the four constraints has to be satisfied. Intersection constraints are, therefore, (1) conditional as the constraint to be chosen depends on the sign of the denominator, D and (2) disjunctive in nature. Standard linear formulation based on conjunctive constraints does not work in this case, where the constraints are disjunctive [25]. Moreover, no commercial optimization packages are available that solve disjunctive constraints. Therefore, it is necessary to convert these OR-type of conditional disjunctive constraints to AND-type of conjunctive constraints. For converting the disjunctive constraints to a form that can be used in optimization formulation, we use additional constraint control variables be used to selectively include or exclude constraints.

6.1. Constraint control variables for handling disjunctive constraints

This section describes the method used to convert disjunctive constraints to conjunctive constraints. In order to handle disjunctive constraints, integer constraint control variables are introduced. The following example shows how the disjunctive constraints generated for the intermediate parts are handled using constraint control variables. Constraint control variables I_1, I_2, I_3, I_4 are introduced. These variables are integers and must satisfy the following conditions:

- $I_1 + I_2 + I_3 + I_4 = 3$,
- $0 \leqslant I_1 \leqslant 1$,
- $0 \leqslant I_2 \leqslant 1$,
- $0 \leqslant I_3 \leqslant 1$,
- $0 \leqslant I_4 \leqslant 1$.

Since I_1, I_2, I_3, I_4 are integers and are in the range of $(0,1)$, the above five conditions ensure that only one of the constraint control variable takes the value 0. All the other constraint control variables should take the value 1. Consider the example shown in Fig. 6. The constraint control variables are added to these disjunctive constraints in the following manner:

$$28.98L8 - 16.27L6 - 1023.55 - \gamma I_1 < 0,$$
$$16.27L6 - 28.98L9 + 1022.87 - \gamma I_2 < 0,$$
$$22.62(L8 - L9) + L6L8 - L6L9 - \gamma I_3 < 0,$$
$$6.36(L8 - L9) + L6L9 - L6L8 - \gamma I_4 < 0,$$

where γ is a very large number, significantly greater than the possible numerical value of the left sides of the above described constraints (i.e., of the order of 10^6). The disjunctive constraints are converted to conjunctive constraints in the following manner. When $I_1 = 0$ and $I_2 = I_3 = I_4 = 1$,

$$28.98L8 - 16.27L6 - 1023.55 < 0,$$
$$16.27L6 - 28.98L9 + 1022.87 - \gamma < 0,$$
$$22.62(L8 - L9) + L6L8 - L6L9 - \gamma < 0,$$
$$6.36(L8 - L9) + L6L9 - L6L8 - \gamma < 0.$$

The above equations show that one of the four initial conditions is retained while the rest of the conditions become trivially true because of the very large value of γ. If this condition is satisfied then the two line segments will not intersect. However, if the condition is not satisfied, the iteration continues where the constraint control variables I_2, I_3, I_4 will successively take the value of zero. Whenever a constraint control variable is set to 1, the constraint to which it has been applied becomes trivially true due to a very large negative number being added to its value. Whenever, a constraint control variable is set to 0, it reverts back to its original form. These constraint control variables allow these disjunctive (OR type) constraints to be handled as conjunctive (AND type) constraints. The integer programming engine used automatically tries various appropriate combinations of constraint control variables to select the right constraint.

6.2. Constraint control variable for handling conditional constraints based on the sign of D

As described in Section 5.1, some of the constraints generated depend upon the sign of D. Many times it is not possible to know the sign of D, upfront. It is important that the sign of D be determined correctly to ensure that the right constraints are generated. Whenever, the sign of D can be established, it is not necessary to use the constraint control variables that are being defined here. However, there are cases in which sign of D cannot be established and it can either become positive or negative depending on the value of parameters. In such cases, it becomes imperative that constraint control variables are used to handle these conditional constraints.

From Section 5.1, we know that

$$D = (x'_2 - x'_1)(y_2 - y_1) - (y'_2 - y'_1)(x_2 - x_1).$$

When the two line segments are parallel (i.e., $D = 0$), it is important that no constraints are added. To ensure this, constraint control variables T_1, T_2 and T_3 are introduced. These constraint control variables are integers and must satisfy the following conditions:

- $T_1 + T_2 + T_3 = 2$,
- $0 \leqslant T_1 \leqslant 1$,
- $0 \leqslant T_2 \leqslant 1$,
- $0 \leqslant T_3 \leqslant 1$.

These conditions ensure that only one of the constraint control variable can take the value of 0. All other constraint control variables should take the value of 1. Now, these constraint control variables are added to the conditional constraints in the following manner. Let us assume that the equations are of the format C/D. Then the conditional constraints can be seen to clearly satisfy the following rules:

- if $D > 0$ then $C < 0$,
- if $D < 0$ then $-C < 0$,
- if $D = 0$ then no constraint.

Now the constraint control variables are introduced in the following manner:

$$\begin{aligned} &C - \gamma T_1 < 0,\\ &-C - \gamma T_2 < 0,\\ &-D - \gamma T_1 < 0,\\ &D - \gamma T_2 < 0,\\ &D - \gamma T_3 \leqslant 0,\\ &-D - \gamma T_3 \leqslant 0. \end{aligned}$$

where γ is a very large number significantly greater in magnitude than C and D (e.g., of the order of 10^6).

From the above formulation, it is clear that

- When $T1 = 0$, only $C < 0$ and $-D < 0$ apply. Others are trivially true.
- When $T2 = 0$, only $-C < 0$ and $D < 0$ apply. Others are trivially true.
- When $T3 = 0$, only $D \leqslant 0$ and $-D \leqslant 0$ apply. Others are trivially true.

Therefore, the formulation only allows the right combination of constraints. These constraints are those that determine the shape of the punch.

6.3. *Combined mixed-integer/enumeration formulation*

Techniques described in Sections 6.1 and 6.2 convert the conditional and disjunctive constraint formulations to non-linear mixed-integer formulations using integer constraint control variables. The convenient method to solve this optimization formulation would be to use mixed-integer optimization solvers. However, the currently available commercial solvers do not have the capability to solve optimization formulation that includes integer constraint control variables and non-linear constraints. However, there are solvers that can solve mixed-integer formulation with linear constraints. These solvers take only a few seconds to solve even large mixed-integer formulations. In order to use these solvers, the constraints need to be converted into a linear format.

Consider the parametric shape of the gooseneck punch as shown in Fig. 5. The intermediate shape of the sheet metal part is already defined in terms of co-ordinate points, which are numerical in nature. From the parametric form of the punch, it can be perceived that giving numerical values to the parameter set ($L1$, $L3$, $L4$, $L5$, $L6$, $L10$) or the parameter set ($L2$, $L7$, $L8$, $L9$) will ensure that the constraints generated will be linear in nature. In this particular problem, since the number of parameters in the set ($L2$, $L7$, $L8$, $L9$) are lesser in number, they shall be given numerical values. The parameters are given a numerical value by an iteration method, where these parameters are introduced in a loop, starting from the lowest value and ending with the highest value of their respective parameter range. This results in all the constraints being converted from a non-linear format to a linear format.

Consider the example as shown in Fig. 6. The constraints that are generated are

$$\begin{aligned} &28.98L8 - 16.27L6 - 1023.55 < 0,\\ &16.27L6 - 28.98L9 + 1022.87 < 0,\\ &22.62(L8 - L9) + L6L8 - L6L9 < 0,\\ &6.36(L8 - L9) + L6L9 - L6L8 < 0. \end{aligned}$$

The parameters $L8$ and $L9$ are in the range of (70, 73) and (43, 46), respectively. Iterating the above equations over the parameter range of $L8$ and $L9$ reduces the above constraints into a linear form. During the first iteration, the value of $L8 = 70$ and $L9 = 43$. This changes the constraints as given below:

$$\begin{aligned} &-16.27L6 + 1005.05 < 0,\\ &16.27L6 - 223.27 < 0,\\ &610.74 + 70L6 - 43L6 < 0,\\ &171.72 + 43L6 - 70L6 < 0. \end{aligned}$$

The solution methodology presented here utilizes a combination of mixed-integer optimization techniques with enumeration techniques. Fig. 9 shows the combined mixed integer and enumeration formulation used to design a punch that will be able to bend multiple parts. The steps for the combined formulation are as follows:

Step 1: During this step the disjunctive nature of the constraints is removed by using constraint control variables. Section 6.1 gives a complete description of this process.

Step 2: The conjunctive constraints resulting from step 1 are still conditional in nature. To remove this conditional nature, integer constraint control variables are introduced. Section 6.2 gives a complete description of this process.

Step 3: Mixed-integer non-linear constraints result from step 2. Parameters are identified that will convert the constraints into a linear format. These parameters are then iterated over their respective parameter ranges. This results in linear mixed-integer constraints.

Step 4: The linear mixed-integer constraint formulation is solved using the CPLEX solver of AMPL. The result of this optimization gives the values of the punch parameters that will ensure that there is no interference between the various sheet metal parts and the resulting punch. The punch parameter value of each cycle is checked with the solution of the previous iteration. If the present solution results in a more optimal solution, then the present parameter solution set replaces the previously

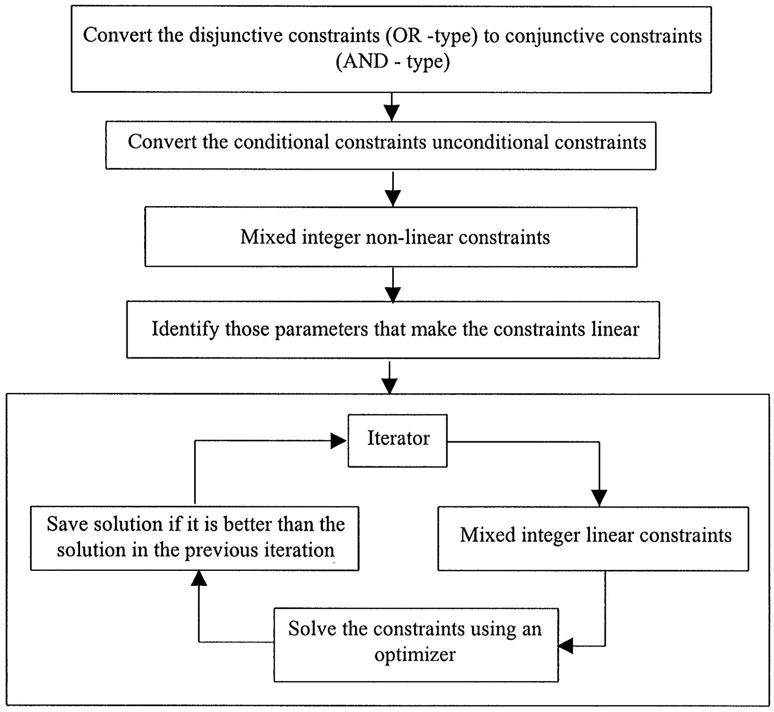

Convert the disjunctive constraints (OR -type) to conjunctive constraints (AND - type)
Convert the conditional constraints unconditional constraints
Mixed integer non-linear constraints
Identify those parameters that make the constraints linear
Iterator
Save solution if it is better than the solution in the previous iteration
Mixed integer linear constraints
Solve the constraints using an optimizer

Fig. 9. Combined mixed-integer/enumeration formulation.

obtained values. At the end of the iterative cycle, the best solution of punch parameter values determines a punch shape that will not interfere with any of the sheet metal parts.

The advantage of using such a combination method is the time saved in using fast mixed-integer solvers. Consider the parametric punch profile shown in Fig. 5. In this case, the punch shape is defined by 10 different parameters. If purely enumeration techniques were used to determine a punch shape that will not interfere with the sheet metal parts, the time taken to solve such a problem would have been very large. By using the mixed-integer solver, the enumeration is reduced to just four parameters. An optimization process that takes a few seconds now replaces the large amount of time required to enumerate the six parameters over their respective parameter range. This exponential reduction in time saved makes this combination of mixed-integer/enumeration formulation very beneficial.

The strength of the punch depends on many factors like the material of the punch and the cross section of the punch. For the mixed-integer formulation discussed in this chapter, the cross section of the punch is used as the objective function for strength. Therefore, we use $L1$–$L6$ as an approximation for the punch strength. The value $L1$–$L6$ increases the width of the main punch body and, therefore, increases the punch strength. This objective function is only a surrogate objective function. The actual strength of the punch is checked using FEA methods.

7. Implementation details

This section discusses the implementation of the algorithm explained in Section 5. Section 7.1 explains the system architecture of the implementation of these algorithms. Fig. 10 shows the various modules of the implementation system. Section 7.2 presents an example run of the implementation system.

7.1. System architecture

The implementation system architecture consists of four main components. These components are:

- *Geometric reasoning component:* The geometric reasoning is carried out using C + + and ACIS. ACIS is a 3D geometric modeler developed by spatial technologies. ACIS is delivered as a library of classes written in C + +.
- *Optimization component*: The optimization is carried out using AMPL. AMPL is a modeling language for linear, non-linear, and integer programming problems. AMPL has been chosen because of its ability to solve large mixed-integer optimization problems. Additionally, AMPL can be called using C + + and the results of AMPL can be retrieved into other file formats.
- *Graphical user interface* (*GUI*): The graphical user interface is developed using Java. The rendering of the parts, punch and the interaction between these two is

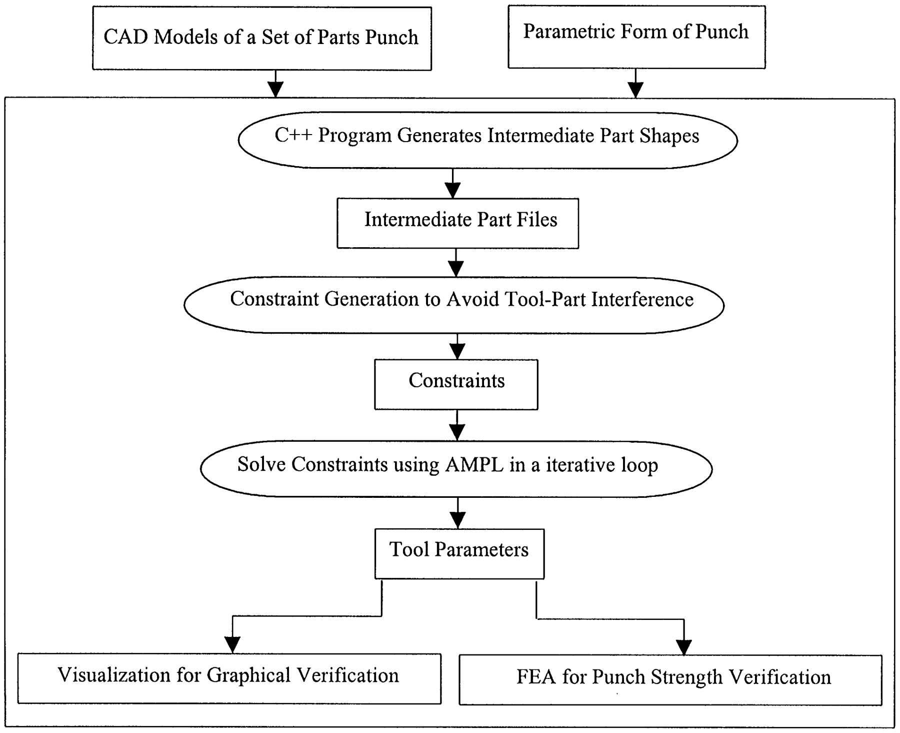

CAD Models of a Set of Parts Punch
Parametric Form of Punch
C++ Program Generates Intermediate Part Shapes
Intermediate Part Files
Constraint Generation to Avoid Tool-Part Interference
Constraints
Solve Constraints using AMPL in a iterative loop
Tool Parameters
Visualization for Graphical Verification
FEA for Punch Strength Verification

Fig. 10. Implementation procedure.

carried out using Java 3D, a package in Java 2. Java 3D provides the ability to create and maneuver solid models.

- *Finite element analysis component*: ANSYS is a finite element modeling and analysis package for numerically solving a wide variety of mechanical problems. In the present system architecture, ANSYS is being used to carry out structural analysis. ANSYS is used to carry out stress calculations to analyze the effects of the bending forces on the punch strength.

The implementation procedure involves the following steps:

1. Given the dimensions of the part and its final shape, intermediate part shapes are generated using C + + programs. The co-ordinates of the end points of the intermediate part shape are stored to a file.
2. Using the co-ordinates of the intermediate part and the co-ordinates of the punch and substituting them in the equations shown in Section 5.1, the constraints are generated for each pair of lines, one of the punch and the other of the part. The heuristics described in Section 5.2 are applied to eliminate the redundant constraints. The constraints that remain are then saved to another file.
3. We have then implemented our constraint solver using AMPL, which takes the above constraint file as the input and optimizes to give the values of the parameters that satisfy the constraints.

7.2. Example run of the implementation system

This section gives an example run of the system. Consider the 10 parts shown in Fig. 11. The material chosen for the parts was low-carbon steel having a thickness of 1.5 mm. The punch material is tool grade steel having a HRC of 43–48. The main frame of the GUI has four buttons:

1. Part selection button.
2. Tool specifications button.
3. Optimization button.
4. Result verification button.

7.2.1. Part selection

When the part selection button is clicked, the frame shown in Fig. 12 is displayed. The main features of this frame are:

1. *Workspace selection*: The user can create a new workspace or open an existing workspace. This workspace denotes the directory that the user will be using during the process of designing the punch.
2. *Part selection*: By clicking on the "BROWSE" button, the user chooses the file denoting the dimensions of a part. This file having the ".part" extension contains the dimensions as well as the bend angles for the part chosen. The part that is chosen is then displayed on the screen.
3. *Bend sequence*: When the "enter the bend sequence" button (Fig. 12) is clicked, the frame shown in Fig. 13

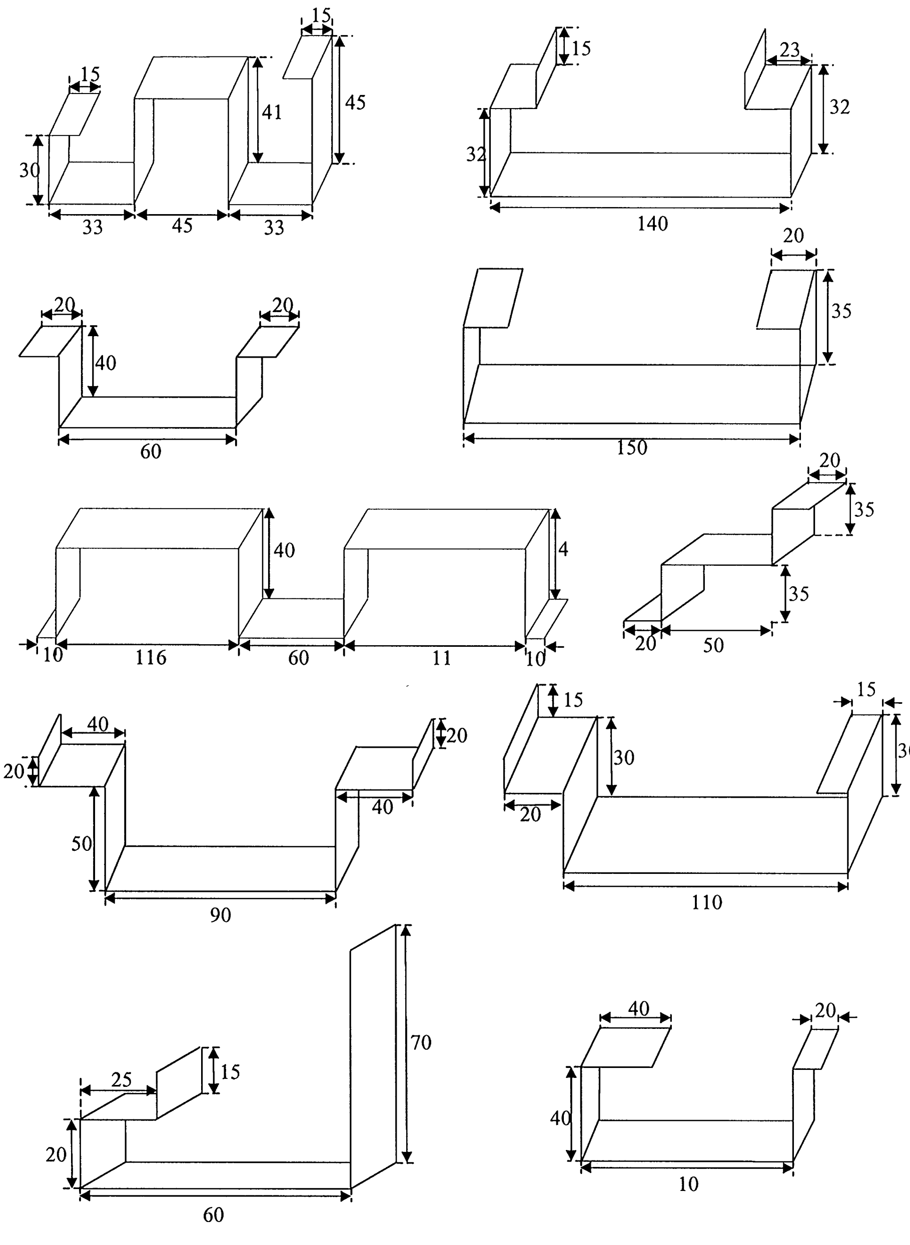


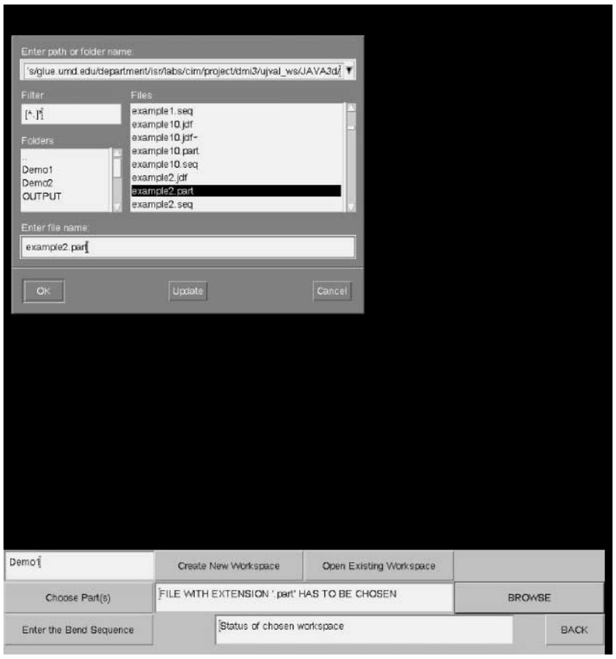
Fig. 11. Ten example sheet-metal parts.

appears on the screen. In this frame, the user can choose the bend sequence for the part that was chosen. The user clicks on the faces of the part displayed on the screen to choose the edge that needs to be bent and the orientation of the part during this process of bending. Thus, the user clicks on two adjacent faces of the part to denote the edge that needs to be bent. The second face clicked denotes that face of the part that needs to be outside the punch press. The status box helps the user determine if the bend edge chosen has already been chosen in the bend sequence. It also tells the user when all the bend edges have been chosen. When all the bend edges of the part are chosen, the "SUBMIT DATA" button is clicked. This causes the creation of the ".seq" file for the part.

The user can go back to the *part selection* frame and choose multiple parts. For the example considered, the user chooses the ten parts and determines the sequence of bending operations for each of the parts.

7.2.2. Tool specifications

When the "tool specifications" button is clicked, the frame shown in Fig. 14 is displayed on screen. This frame helps to determine the range for the parameters that define the gooseneck punch. When the user clicks on the "tool figure" button, a parametric figure of the punch



's/glue.umd.edu/department/isr/labs/cim/project/dmi3/ujval_ws/JAVA3d/

[^.]*

example1.seq
example10.jdf
example10.jdf~
example10.part
example10.seq
example2.jdf
example2.part
example2.seq

Demo1
Demo2
OUTPUT

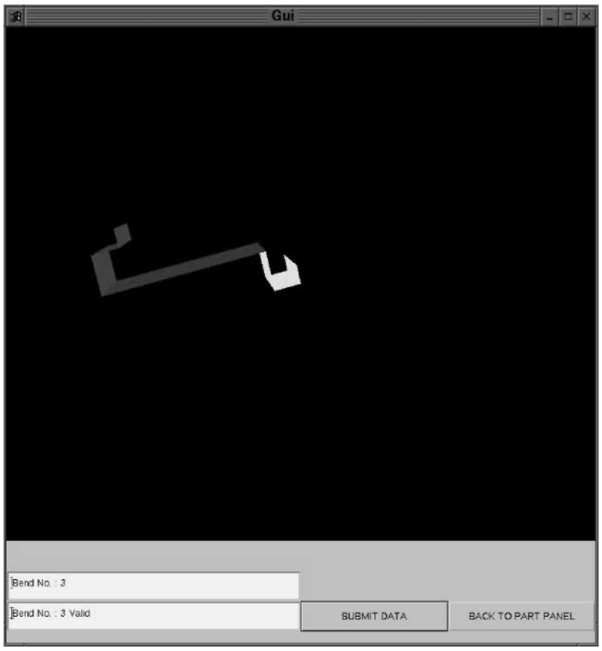
example2.part

Demo1

FILE WITH EXTENSION '.part' HAS TO BE CHOSEN

Status of chosen workspace

Fig. 12. Workspace and input file(s) selection.

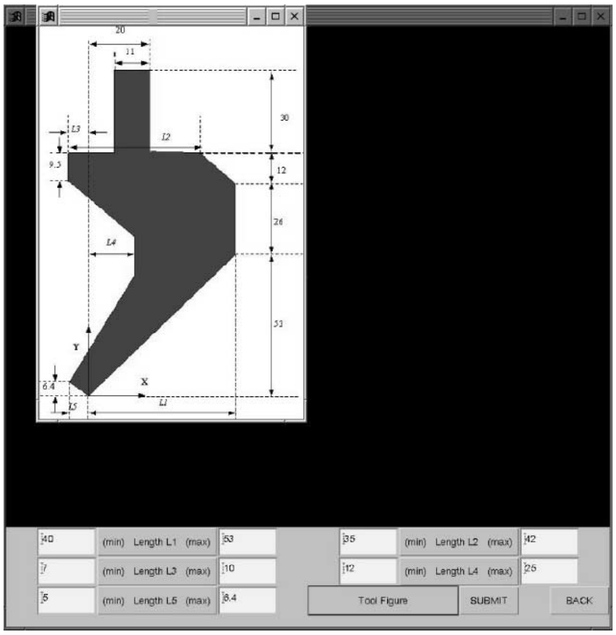


Bend No. : 3

Bend No. : 3 Valid

Fig. 13. Bend sequence generation.

that is being designed is displayed on screen as shown in Fig. 14. The user is given the flexibility to determine the range for the parameters of the punch. The user enters the upper and lower values for each of the parameters. Clicking the "submit" button will create a new child process, which runs a C + + program that automatically creates the *constraints file*. This C + + program takes the ".part" and ".seq" files for all the parts chosen and the parametric form of the punch, as input. It then automati-



40	(min) Length L1 (max)	53	35	(min) Length L2 (max)	42
7	(min) Length L3 (max)	10	12	(min) Length L4 (max)	25
5	(min) Length L5 (max)	8.4			

Fig. 14. Tool specifications.

cally creates the constraints using the algorithm and heuristics explained in Sections 5.1 and 5.2. The user then returns to the main frame of the GUI. For the example considered, the parameters for the punch have lower and upper values as given below:

$$40 \leqslant L1 \leqslant 55, \quad 35 \leqslant L3 \leqslant 42, \quad 7 \leqslant L5 \leqslant 10,$$
$$79 \leqslant L7 \leqslant 82, \quad 40 \leqslant L9 \leqslant 43,$$
$$77 \leqslant L2 \leqslant 82, \quad 18 \leqslant L4 \leqslant 23, \quad 12 \leqslant L6 \leqslant 25,$$
$$69 \leqslant L8 \leqslant 73, \quad 5 \leqslant L10 \leqslant 6.5.$$

Other restrictions on the size of the punch were determined as follows:

$$L1 - L4 \geqslant 30,$$
$$5 \leqslant L1 - L2 \leqslant 15,$$
$$45 \leqslant L2 + L3 \leqslant 50.$$

Apart from these constraints on the punch parameters, the other constraints are the ones that are generated as a result of the check for interference between the punch and the part.

7.2.3. *Optimization*

When the user clicks on the "optimization" button in the main frame, the optimization process is carried out using the AMPL software package. Clicking the "optimize" button causes a child AMPL process that involves the following steps:

1. The *constraints file* is taken as input by the optimizer.
2. AMPL uses its mixed-integer optimizer to carry out the optimization.

3. The output of the optimization process contains the values of the parameters of the punch. These values are saved to a ".sol" file.
4. The status box in the frame changes the status from "Optimization is being carried out … Please wait" to "Optimization has been completed … return to main menu".

7.2.4. Result verification

When the user clicks on the "result verification" button on the main frame, a frame containing the following features appears on the screen:

1. *Parameter values*: Clicking the "get results" button on this frame displays the results of the optimization process. The ".sol" file contains the values of the punch parameters and these are displayed in this frame.
2. *Graphical verification*: Clicking the "graphical verification" button on this frame opens a display window as shown in Fig. 15. This window displays the interaction of the punch with every intermediate part shape. The "previous" and "next" buttons help the user to see the interaction between the punch and the intermediate part shape, as determined by the bend sequence. The display on the screen can be rotated and enlarged by the user, thus helping the user verify that the designed punch does not interfere with any of the intermediate part shapes.
3. *FEA*: Clicking on the "get results" button automatically creates the ".iges" file. This IGES file contains all the information regarding keypoints and line segments that define the shape of the resulting punch the material properties of the punch and the forces acting on the punch. A separate process is used to set the ANSYS environment. When the IGES file is imported in ANSYS, it automatically generates a finely meshed representation of the punch. This finite element representation of the punch can then be used to analyze the various stresses acting on the punch during the process of bending. This FEA procedure helps to analyze the strength of the punch. If the strength of the punch is able to withstand the forces of bending, then it can be assumed that the designed punch is safe to be used to bend all the parts.

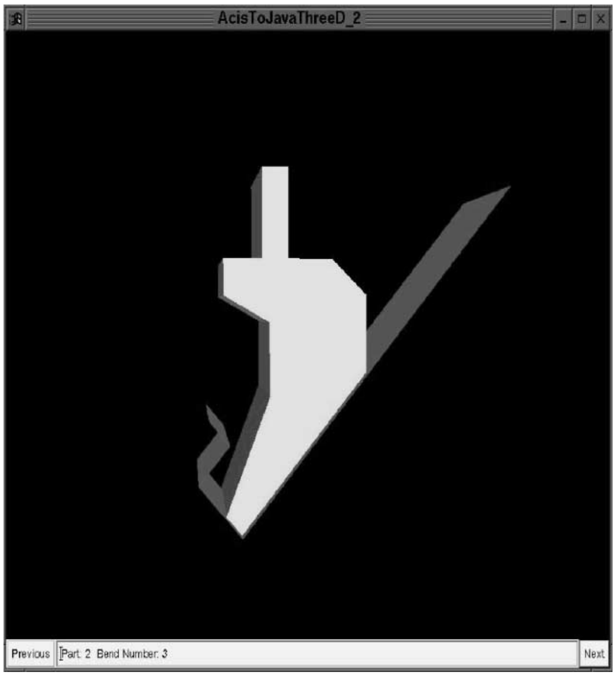


Fig. 15. Graphical verification of part–tool interaction.

The satisfaction of the complete constraint file results in a punch having the following set of parameters:

- $L1 = 53$, $L2 = 79$, $L3 = 38$, $L4 = 20$, $L5 = 7.0$, $L6 = 12$, $L7 = 81.5$, $L8 = 70$, $L9 = 45.5$, $L10 = 6.5$.

This punch is then visually verified to ensure that there is no interference between the part and the tool. FEA is carried out to check for punch strength. The punch satisfies all strength considerations. The system took 1 h and 48 min using a Sun Solaris Ultra 10 machine to determine a single punch that will bend all the 10 parts. This is compared to years of computing if purely enumerative methods were used to determine a single punch for bending these 10 parts. This punch shape passes the strength test using FEA software ANSYS.

8. Conclusions

This paper describes an algorithm for designing punch shapes that do not intersect with a given set of parts and at the same time have enough strength to withstand the forces of bending. This paper makes contributions in the following areas:

1. *Algorithm to design a single tool for bending multiple parts*. A new algorithm was developed for designing a punching tool for multiple sheet metal parts. Traditionally, tool engineers used to determine a single punch for each type of part. The algorithm detailed in this paper helps a tool engineer synthesize a single punch for multiple different kinds of parts. The process of designing the punch involved creating a parametric form of the tool, checking this parametric shape for interference with the various intermediate part shapes and then optimizing the punch for strength while constrained by the interference formulations developed in this paper.
2. *Ability to handle disjunctive and conditional constraints*. This paper developed an approach to convert conditional and disjunctive type of constraints

to continuous "AND" type of constraints using constraint control variables that take only values of zero and one. This approach helps to convert a set of disjunctive constraints to a set of easily solvable constraints.

3. *A prototype implementation*. Based on the approach described in this paper, we have developed a prototype system that helps to synthesize a tool for multiple sheet-metal parts. Details on this system have been provided in Section 6. This system automatically synthesizes the shape of a tool that does not interfere with multiple sheet metal parts and then analyzes the tool for strength using FEA methods.

8.1. Benefits

The benefits in designing a single punch for multiple sheet metal parts are:

1. *Setup time reduction*. This approach will help process planners in selecting a single punch shape that will work for multiple types of parts. This will help in reducing the setup times and setup costs for the small batch manufacturing environment. For the example of 10 parts, a single-part planning approach would have resulted in 10 setup changes while multi-part planning reduces the number of setups to one.
2. *Tool cost savings*. A single tool is used to bend multiple parts. This leads to a reduction in the cost of tools. For the example of 10 parts, a single tool for each part would have resulted in 10 different tools while this approach results in one tool for all the 10 parts.
3. *Automated process planning*. The system proposed in this paper automatically generates the shape of punch that can bend multiple parts. As a result, this system can be integrated with computer-aided process planning systems. This will help process planners to develop process plans quickly.

8.2. Directions for future work

The approach described in this paper has its limitations. These limitations can be future research issues in the field of tool design for multi-part tool selection. Some of the current limitations and areas for future research are:

1. *Failure to design a punch*. The present algorithm returns a failure when there is no single punch that can be used to bend all the parts. In such a case of failure to design a punch, it will prove useful if the family of parts can be sub-divided into smaller part families so that a single punch can be designed to bend parts in each sub-family of parts. This will require backtracking capabilities along with much superior optimization techniques.
2. *Incorporating alternative operation sequences in generation of constraints*. Currently only a given operation sequence is considered for each part in the part family. Alternative operation sequences need to be considered. Therefore, it becomes necessary to integrate the operations sequence generation techniques with the techniques developed as a part of this research.
3. *Three dimensional part shapes*. The current work is applicable only to 2.5D parts. Future research will have to extend it to 3D part shapes. The interaction of the part and the tool also needs to be extended to the 3D domain.

Acknowledgements

This research has been supported by the NSF Grant DMI9896255. Opinions expressed in this paper are those of authors and do not necessarily reflect opinion of the National Science Foundation.

References

[1] Amada Sheet Metal Working Research Association. Bending technique, 1st ed. Machinist Publishing Company Limited, 1981.
[2] Benson SD. Press Brake technology: a guide to precision sheet metal bending. Society of Manufacturing Engineers, 1997.
[3] Pollak HW, Tool design, 2nd ed. Prentice Hall, Englewood Cliffs, NJ, 1988.
[4] Wilson FW, editor-in-chief. Fundamentals of tool design. Society of Manufacturing Engineers, Academic Press, New York, 1962.
[5] Eary DF, Reed EA. Techniques of pressworking sheet metal – an engineering approach to die design, 1974.
[6] Bourne DA. Intelligent manufacturing workstations. Knowledge-based automation of processes. ASME Winter Annual Meeting, Anaheim, CA, 1992, pp. 77–84.
[7] Gupta SK, Bourne DA, Kim K, Krishanan SS. Automated process planning for sheet metal bending operations. J Manuf Systems 1998;17(5):338–60.
[8] Gupta SK, Bourne DA. Sheet metal bending: generating shared setups. ASME J Manuf Sci Eng 1999;121:689–94.
[9] deVin LJ, deVries J, Streppel AH, Kals HJJ. A CAPP system for small batch manufacturing of sheet metal components. In: Proceedings of the 24th CIRP International Seminar on Manufacturing Systems, Copenhagen, 1992. p. 171–82.
[10] deVin LJ, deVries J, Streppel AH, Klaassen EJW, Kals HJJ. The generation of bending sequences in a CAPP system for sheet metal components. J Mater Process Technol 1994;41:331–9.
[11] Radin B, Shpitalni M, Hartman I. Two stage algorithm for rapid determination of the bending sequence in sheet metal products. ASME Design Automation Conference, Irvine, CA 1996.
[12] Nnaji BO, Kang TS, Yeh SC, Chen JP. Feature reasoning for sheet metal components. Int J Prod Res 1991;29(9):1867–78.
[13] Yut G, Chang TC. A study of automated process planning for sheet metal products. NSF Design and Manufacturing Systems Conference, January 1993.
[14] Uzsoy R. An experimental expert system for process planning of sheet metal parts. Comput Ind Eng 1991;20(1):59.
[15] Braibant V, Fleury C. Shape optimal design using B-Splines. Comput Methods Appl Mech Eng 1984;44:247–67.

[16] Rajan SD, Budhiman J. Gani L, Topologically-based adaptive mesh generator for shape optimal design. 32nd Structures, Structural and Dynamics and Material Conference, 1991. p. 653–63.
[17] Rossen DW, Grosse IR. A feature based shape optimization technique for the configuration and parametric design of flat plates. Eng Comput 1992;8:81.
[18] Schramm U, Pilkey WD. The coupling of geometric dimensions, finite elements using NURBS – A study in shape optimization. Finite Elements Anal Des 1993;15:11–34.
[19] Lindby T, Santos JLT. Shape design sensitivity analysis and optimization with an existing associative CAD system. Symposium on Multidisciplinary Analysis and Optimization, Panama City, FL, 1994.
[20] Noel F, Leon JC, Trompette P. Shape optimization of three-dimensional parts based on a closed loop between structural analysis and geometric modeling. Eng Comput 1995;11: 114–21.
[21] Salagame R, Belegundu AD. Shape optimization with p-adaptivity. Symposium on Multidisciplinary Analysis and Optimization, Panama City, Fl.
[22] Zhang WH, Beckers P, Fleury C. A unified parametric design approach to structural shape optimization. Int J Numer Methods 1995;38:283–91.
[23] Salagame R. Automated shape optimization using parametric solid models and p-adaptive finite element analysis. ASME Design Engineering Technical Conference, Sacramento, CA, 1997.
[24] Vajna IS, Schabacker mM, Schmidt IR. General procedure for parameterisation with 3-D CAD systems. ASME Design Engineering Technical Conferences, Las Vegas, September 12–15, 1999.
[25] Greenberg H. Integer Programming, 1971.