|| Volume 4 || Issue 2 || February 2019 || ISSN (Online) 2456-0774

INTERNATIONAL JOURNAL OF ADVANCE SCIENTIFIC RESEARCH AND ENGINEERING TRENDS

INTERPOLATION AND REDUCTION ALGORITHMS FOR DISCRETE MODELING OF POCKET MILLING

Munish Kumar, Dr. Pankaj Khatak*

Mechanical Engineering Department, Guru Jambheshwar University of Science & Technology, Hisar 125001, India

**Corresponding Author Email: pankajkhatak@gmail.com*

Abstract: Pocket milling is one of the fundamental machining processes in the manufacturing industry. A number of researchers has presented methods and techniques for optimal milling of a pocket. Most of the methods are based on work area approximation schemes using discrete elements such as squares. This paper presents interpolation algorithms for modeling of work area of a pocket. For decreasing the discrete squares, reduction algorithms are also presented. The algorithms are implemented in an optimization problem of pocket milling toolpath. Results indicate that while interpolation algorithms effectively model the design element, reduction algorithms greatly decrease the search space of the optimization methods leading to a decrease in convergence time.

Keywords: *Discretization, interpolation, reduction, optimization.*

I INTRODUCTION

In a typical machining process, a cutting tool is required to follow a sequence to machine given part. The part may consist of a number of design features such as pockets, contours, or holes[1]. A tool-path is generated which covers all these design features at least once. This generation of toolpath is accomplished either manually or using a CAM software such as MasterCAM, ESPIRIT, etc. In both the case, a set of points is marked onto the work piece which serve as an entry/exit for cutting tool. The design feature itself consist of a large number of points in general. This can be explained with the fact that when a CNC controller receives a command for linear interpolation through G01, a number of points between the end points are calculated and fed to the actuators in sequence. The drives of axes are actuated in pulses of motion. Tool seems to be moving in a straight line but at micro level it moves in a zig-zag motion since feed for x and y-axis are fed one by one insteps[2].

The accuracy of a toolpath depends largely on these interpolation points. Therefore, a large number of points are calculated to obtain a perfect straight-line motion. It implies that these interpolation points define a particular design feature and hence a part. For this, interpolation of such movements is performed in this researchalso[3].SimilartoaCNCcontroller,the number of points obtained here is also very large. A toolpath is considered as a sequence of these points.

Such a large number of points makes it difficult to find an optimal sequence or a toolpath. This is because of possibilities of a vast search space. To explore such large spaces is a tough task for any optimization problem and is not feasible in most of the cases. Computational complexity increases while processing the points leading to higher costs and time disadvantages. Task can be simplified if the search space is reduced so as to make optimization techniques able to handleit[4].

The process of discretization provides a good method of approximation of a workpiece. Using a discretized set of points or squares, a part can be easily approximated. Hence a small number of points is necessary to find an optimal sequence rather than using a large set of points[5]. This reduces the search space for an optimization method along with a decrease in computational complexity. Due to these reasons a discretization framework is proposed in this research based on the concept of design elements and squares grid. Such discretization techniques are discussed by [6], [7] and[8].

A prepared drawing of part marks various dimensions and features of that part such as pockets, holes, fillets, contours, etc. These design features are needed to be formed on a workpiece through machining. Efforts are made to ensure that proposed design features are cut with efficiency and quality. It is the assignment of a planner or programmer to efficiently machine those features onto the workpiece using adequate cutters[9]. A process plan is prepared for the same consisting of several cutter locations along with machining parameters associated with them. After selecting an appropriate cutter for current operation, cutter location data (CL data) is used to create toolpaths. It connects a number of cutter locations on workpiece in sequence which cutting tool has to follow to machine a particular design feature [10]. A number of optimal size cutters and relative toolpaths are needed for creation of an efficient process plan. Machining parameters, on one hand, influence the selection of cutters and toolpaths, but design features play a dominant role in efficient preparation of toolpaths and processplans[11].

S. Omirou points out that set of arbitrarily oriented primitive shapes such as rectangular blocks, circular cylinders, spheres, cones and tori is sufficient for modelling 90% of machined part [12]. Typical design features include rectangular pockets, circular pockets, holes, slots, fillets, chamfers, contours etc. These design features can be further segmented into design elements. The design elements are considered here as basic units which shape a design feature and hence a part. Typical elements include lines and arcs. A set of one or more design elements connected together constitute a design feature. For instance, a 3D part having 3 design features is shown in Figure1.

The design elements of a part lead to separation of areas that are to be machined, known as machinable area, from other areas of workpiece. Machinable area is the area of actual concern where a cutter is placed/moved to cut a design feature. Different parts of machinable area are visited by a cutter in sequence for successful machining of part/component. Therefore, a cutting strategy or toolpath can be defined as the sequence of cutter locations of an element inside machinable area. Usually, main objective of a typical toolpath optimization is to obtain such a cutting strategy or toolpath which allows complete cutting of all the elements in least amount of time. Furthermore, time taken to cut a part is characterized by the distance travelled by a cutting tool during machining. Short distances or tool travels lead to shorter time periods. It is thus obvious that distance travelled by a tool dependsuponthecutterlocationsorsimplypointson

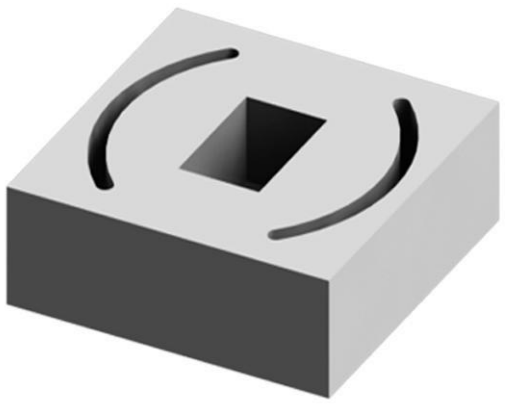
Figure 2. 3D part having three design features.

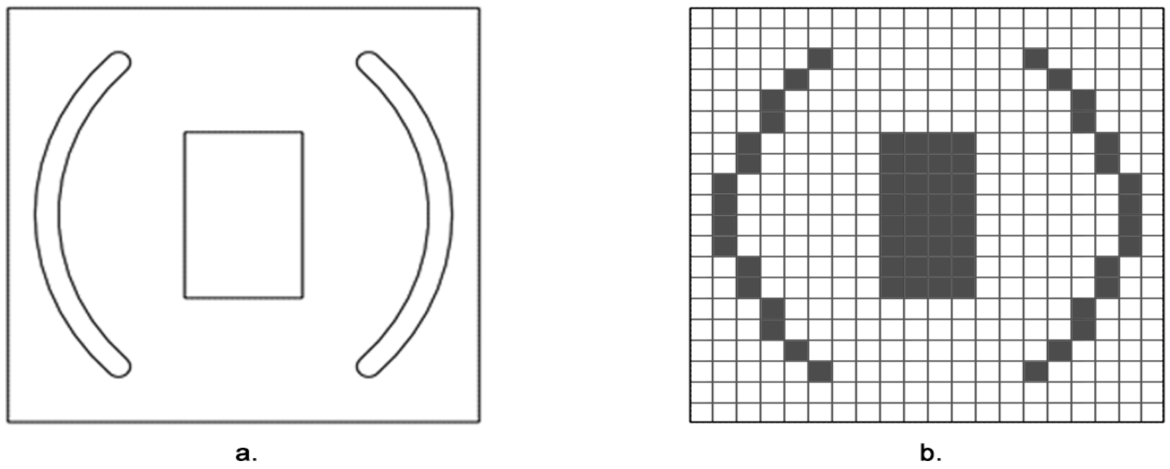

Figure 1. Discretization framework showing, a. Features of part, b. Approximation of given part by square grid.

workpiece visited by it. Figure 2b shows the discretized model of the 3D part shown in Figure 1. The design features are represented with the help of discrete squares. A toolpath is required to visit each discrete square to machine the givenpart.

In this paper, algorithms are presented for implementation in discrete modelling of pocket milling work area. Section 2 discusses the interpolation algorithms for linear, circular and pocket elements. Section 3 presents reduction algorithms for linear, circular and rectangular pocket. In Section 4, tests are conducted to validate the performance of algorithms and results arepresented.

II INTERPOLATION OF LINEAR, CIRCULAR AND POCKET ELEMENTS

Theinterpolationalgorithmoran‘interpolator‟isrespon sible for calculation of data points for a design element. These calculations, in turn, are based on standard equations of geometric entities such as straight line, circle, etc. The main purpose of an interpolation algorithm is to calculate intermediate data points between points defined by user. This includes a number of calculations to obtain valid and correct interpolation data points. A simple and easy to implement method is to use standard equations of geometrical entities using which x and y-coordinate values of a number of data points at regular interval can be calculated. The so-obtained x-y values are used to activate corresponding elements ofworkMat.

The interpolation algorithms are developed for each design element individually. Interpolator of linear elements is based on standard equation of straight line while the interpolator for circular elements is based on standard equation of a circle, discussed in further sections. Modeling of rectangular pockets is not based on this approach, rather a more direct technique is implemented. This approach provided a simple method of modeling rectangular pockets and avoid complex calculations. The data obtained from user for a particular design element is transferred to interpolator of that element. Generally, construction data such as end points, radius, length, width, etc. are required by an interpolator. It is programmed to obtain such data and calculate intermediate carry out furtherprocessing.

2.1 LinearInterpolation

The interpolation algorithm for a linear element is shown in Figure 4. It is based on standard equation of a straight line, $y = mx + c$. This equation canbe modified as shown below,

$$\frac{y - y1}{y2 - y1} = \frac{x - x1}{x2 - x1}$$

The above equation is used to calculate x and y coordinates of a linear element. By putting the value of x, corresponding value of y can be easily calculated, given the two end points (x1, y1) and (x2, y2). In this context, first slope of the line is calculated using below formula,

$$m = \frac{y2 - y1}{x2 - x1}$$

Based on the value of slope, selection of the coordinate to be supplied to the equation of line is performer. Figure 3 shows slope values of a straight line. If the value of slope is greater than 1, values of x-coordinate are generated at regular interval within the range of [x1, x2]. Now, for each value of x-coordinate, a value of y-coordinate is calculated by putting the variables in the equation. And if the value of slope is less than one, y-coordinates are generated at regular intervals and corresponding values of x-coordinate are calculated through the equation. The number of points generated between two end points of any line is thrice the distance between the twopoints.

After all the data points have been calculated, x and y-coordinate values are stored in respective arrays and transferred for square activation. It has been discussed earlier that the x and y-coordinate values are rounded off to their nearest integer values. Then, squares present atthe

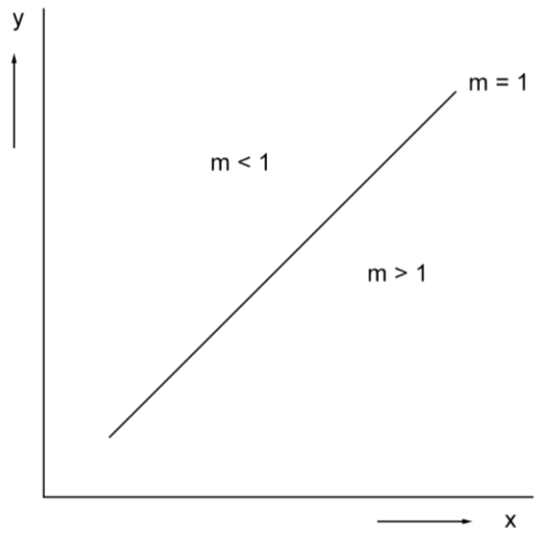

Figure 3. Values of slope of a straight line.

location of these points are activated by assigning ‘1’tocorresponding element of *workMat*. In this way, a linear element is added to the workpiece. All the data is stored in arrays and transferred to the decision maker (DM) and visualized on graphs. DM stores all points combined with their data in *pData* array.

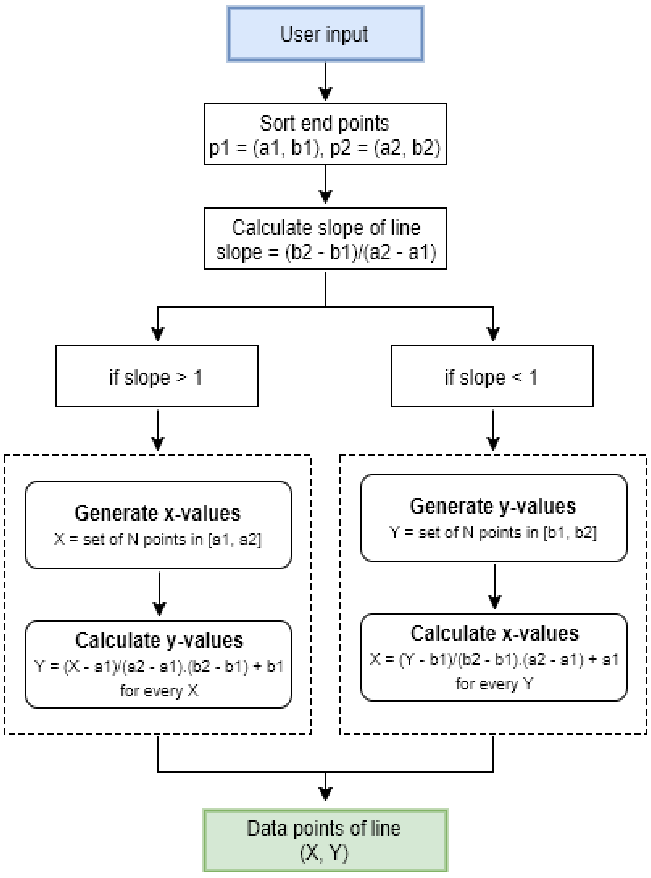
User input
Sort end points
p1 = (a1, b1), p2 = (a2, b2)
Calculate slope of line
slope = (b2 - b1)/(a2 - a1)
if slope > 1
if slope < 1
Generate x-values
X = set of N points in [a1, a2]
Calculate y-values
Y = (X - a1)/(a2 - a1).(b2 - b1) + b1
for every X
Generate y-values
Y = set of N points in [b1, b2]
Calculate x-values
X = (Y - b1)/(b2 - b1).(a2 - a1) + a1
for every Y
Data points of line
(X, Y)

Figure 4. Linear interpolator.

2.2 CircularInterpolation

The principle of a circular interpolator is similar to a linear interpolator. It is based on standard equation of circle, which is,

$$x^2 + y^2 = r^2$$

This is the equation of a standard circle with radius 'r'and centreat origin.This equation is used to calculate x and y coordinate values of data points. The interpolation of a circular arc is processed in three steps (see Figure 7):-

1. Generation of circle atorigin.
2. Shift circle to actual centerpoint.
3. Trim circle to create desiredarc.

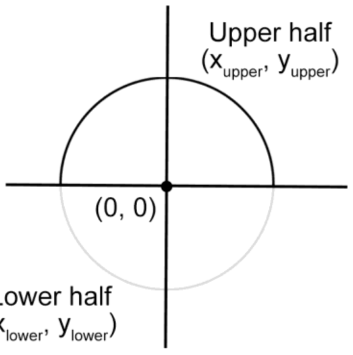

Figure 5. Full circle at origin.

From user input, the values of end points, and radius are retrieved at the first step. Then a circle with same radius is generated at origin. This is accomplished by calculating the data points of the circle, i.e. (X0, Y0), using above equation. For this purpose, values of x-coordinate at regular interval are generated. To ensure smooth approximation of the circle, the distance between points is taken as 0.001 mm (1 micron). For these x-coordinates, value of y-coordinates is calculated. Values of only upper half of the circle are obtained in this way (Xupper, Yupper) as shown in Figure 5. Therefore, lower half of the circle (Xlower, Ylower) is obtained from upper circle. This can be done by reversing Xupper and taking negative of Yupper. The full circle of at origin would have coordinates X0 = [Xupper; Xlower], Y0 = [Yupper; Ylower].

Based on the end points and radius of the arc, the

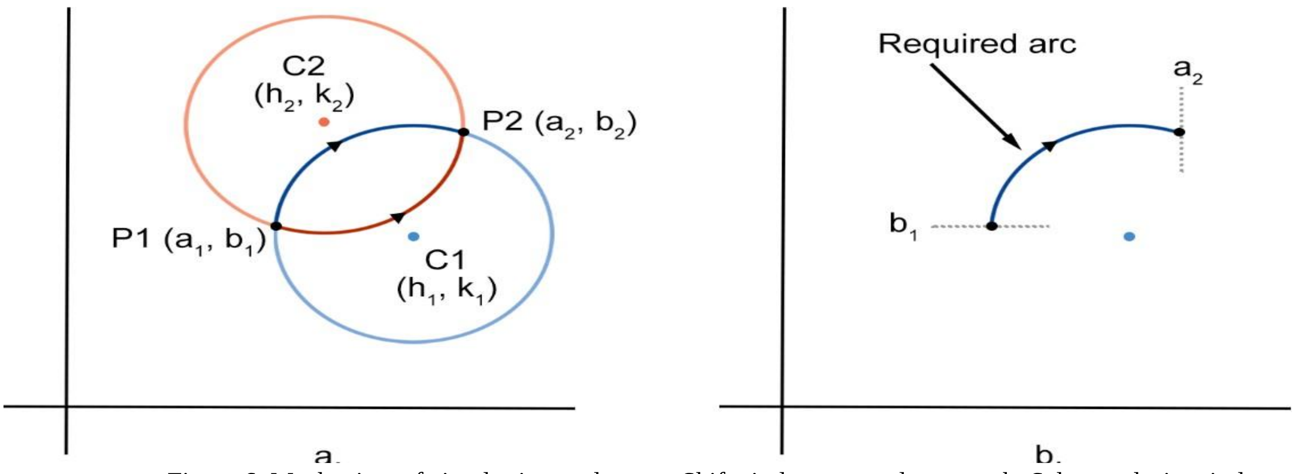

Figure 6. Mechanism of circular interpolator, a. Shift circles to actual centers, b. Select and trim circle to obtain arc.

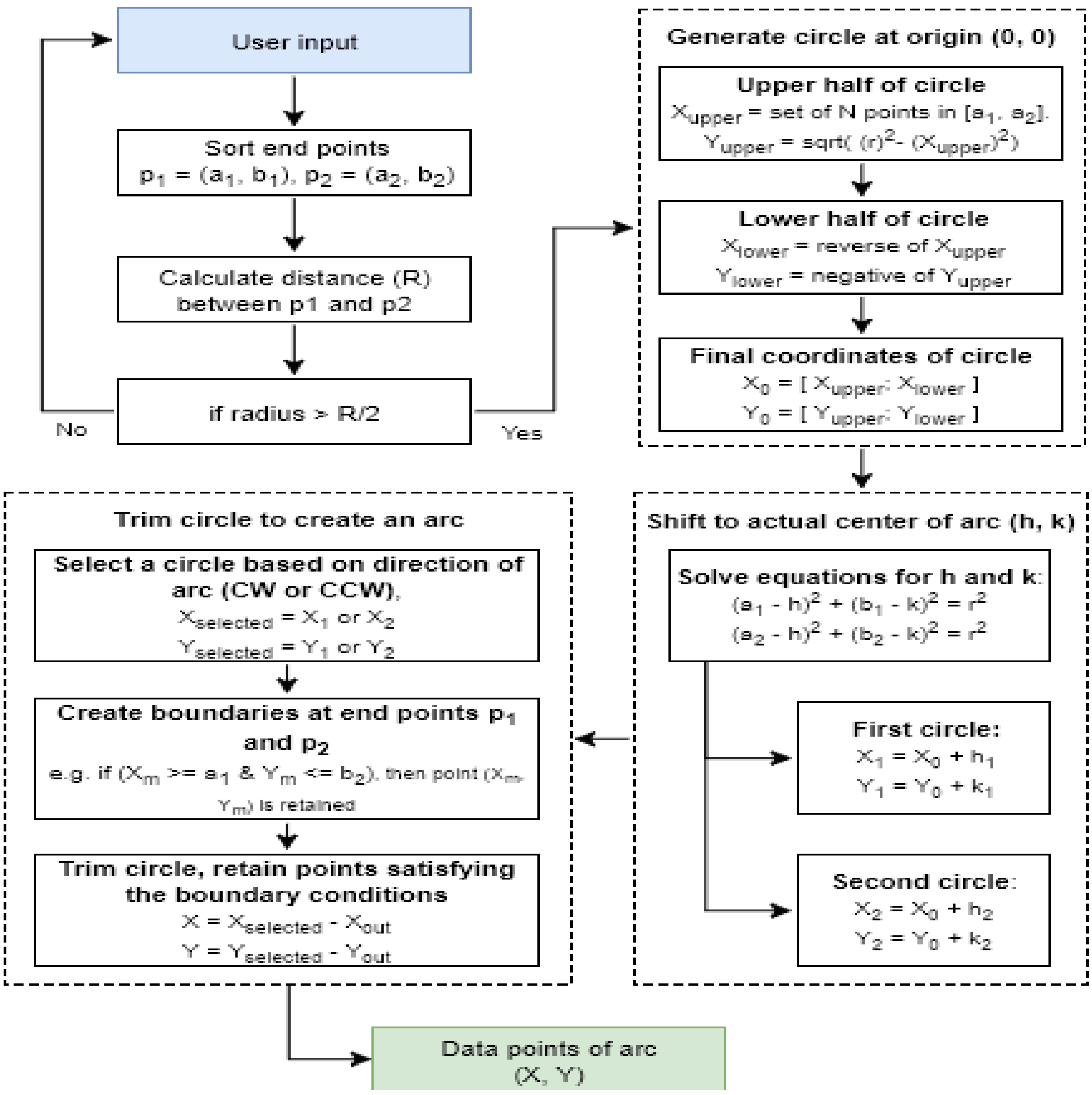
User input
Sort end points
$p_1 = (a_1, b_1)$, $p_2 = (a_2, b_2)$
Calculate distance (R) between p1 and p2
if radius > R/2
No
Yes
Generate circle at origin (0, 0)
Upper half of circle
X_{upper} = set of N points in $[a_1, a_2]$.
$Y_{upper} = \text{sqrt}((r)^2 - (X_{upper})^2)$
Lower half of circle
X_{lower} = reverse of X_{upper}
Y_{lower} = negative of Y_{upper}
Final coordinates of circle
$X_0 = [X_{upper}; X_{lower}]$
$Y_0 = [Y_{upper}; Y_{lower}]$
Shift to actual center of arc (h, k)
Solve equations for h and k:
$(a_1 - h)^2 + (b_1 - k)^2 = r^2$
$(a_2 - h)^2 + (b_2 - k)^2 = r^2$
First circle:
$X_1 = X_0 + h_1$
$Y_1 = Y_0 + k_1$
Second circle:
$X_2 = X_0 + h_2$
$Y_2 = Y_0 + k_2$
Trim circle to create an arc
Select a circle based on direction of arc (CW or CCW),
$X_{selected} = X_1$ or X_2
$Y_{selected} = Y_1$ or Y_2
Create boundaries at end points p_1 and p_2
e.g. if ($X_m >= a_1$ & $Y_m <= b_2$), then point (X_m, Y_m) is retained
Trim circle, retain points satisfying the boundary conditions
$X = X_{selected} - X_{out}$
$Y = Y_{selected} - Y_{out}$
Data points of arc (X, Y)

Figure 7. Circular interpolator.

center of arc is calculated in the next step. This can be done using above equation of circle. By putting the values two equations are obtained as: -

$$(a_1 - h)^2 + (b_1 - k)^2 = r^2$$
$$(a_2 - h)^2 + (b_2 - k)^2 = r^2$$

These two equations can be solved for h and k after specifying the values of (a1, b1), (a2, b2) and r. The solution to the equations yields two center points, viz. (h1, k1) and (h2, k2). This implies the fact that there can be two circles of similar radius passing through any two given points. The full circle (X0, Y0) can now be shifted to these center points as shown in Figure 6(a) in red and blue. This is performed by adding value of h to each of the x-coordinates and k toy-coordinates.

These two circles give two possible arcs between points P1 and P2, one in clock-wise direction and the other in counter-clockwise direction. To obtain required arc, one of the circles needs to be selected and trimmed. This selection is made on the basis of direction of the arc as specified by user. So, based on the direction of arc and orientation of points (P1 and P2) with respect to each other, one circle is selected and the other gets deleted. In the next step the circle is trimmed to obtain arc. For this, two boundary conditions are applied to both the points, each along x and y-axis. These conditions are also based on the orientation of the points P1 and P2. For example, in Figure 6(b), the blue circle is selected and boundary conditions are applied. For a point (Xm, Ym) on the periphery of the circle to lie within the boundary, the conditions are as follows,

$$X_m < a2, Y_m > b1$$

All points of blue circle satisfying the boundary conditions are retained while deleting others. This trimming process of the circle gives an arc in clockwise direction between the points P1 and P2. Similarly, to obtain an arc in counter-clockwise direction, red circle is selected. In that case, the boundary conditions would be,

$$X_m > a1, Y_m < b2$$

The orientation of points plays a vital role in selection of circles and the boundary conditions. After trimming the circle, so obtained x and y-coordinates of the arc (X, Y) are then transferred for square activation. The process of square activation of an arc is similar to that of a line discussed in previous section where elements of workMat are assigned‘1’foractivation.Theindexofactivesquaresarestoredand transferred to the decision maker (DM) which combines them with corresponding construction and machining attributes.

2.3 Rectangular Pocket Interpolation

The interpolation of a rectangular pocket is not actually an interpolation process. But it is a more direct approach than interpolation and does not involve the calculation of data points. As discussed earlier, after defining the dimensions of a pocket, user is required to specify location of the pocket in terms of a base point. It can be any of the four corners of the pocket. The primary base point is the lower left corner of pocket which serves as a starting point for interpolation. Any base point can be chosen by the user, and by some arithmetic means it is transferred to the lower left corner in background.

A loop is initialized from this point to process the points. In every iteration of the loop points are considered row-wise as shown in Figure 8. The activation of squares takes place within the loop while processing each point. The points are constrained to length and width of the pocket. Therefore, no point lying outside the dimensions of pocket is considered. The iterations are continued until the last point indicated in the figure. So, loop is terminated after this point is processed, signifying interpolation of all the points of a pocket. The index values of all these points is stored in an array during the iterations. After completion of the process, index values obtained from the loop are transferred to the decision maker (DM) in the form of X-Y values of points.

Hence, a more direct approach for rectangular pockets is used in which no complex calculations are needed. Also, the process of activation takes place during iterations. Therefore, no separate process is required for square activation as in case of linear and circular interpolators where both processes are performed in two separate steps.

III REDUCTION OF DISCRETE POINTS OF DESIGN ELEMENTS

Reduction is a process of reducing the number of activated squares for an element. This process is carried out after interpolation algorithms have successfully calculated data points for all the elements defined by user. Before executing reduction algorithms, data arrays need to be verified and validated against any discrepancies. The main data arrays required by reduction algorithms is pData and toolData. Similar to interpolation algorithms, separate reduction algorithms are developed for linear/circular element and for rectangular pocket. The main algorithm for reduction is shown in Figure9.

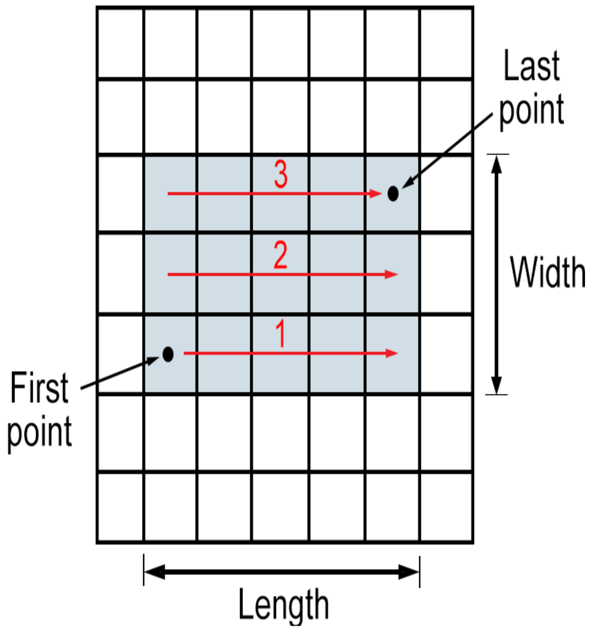

Figure 8. Rectangular pocket interpolator.

Validation of data is performed by pressing Activate button on the control panel. When this button is pressed, background algorithms acquire all data arrays from the decision maker. These data arrays are then passed through various pre-defined set of conditions. A data array is considered as validated if it satisfies all the conditions. If any data array is does not qualify any condition, an error is issued to the user asking for data correction. Validated data arrays are then coupled together in a structure data type, named vars. This structure facilitates easy availability of complete data to algorithms at a single place.

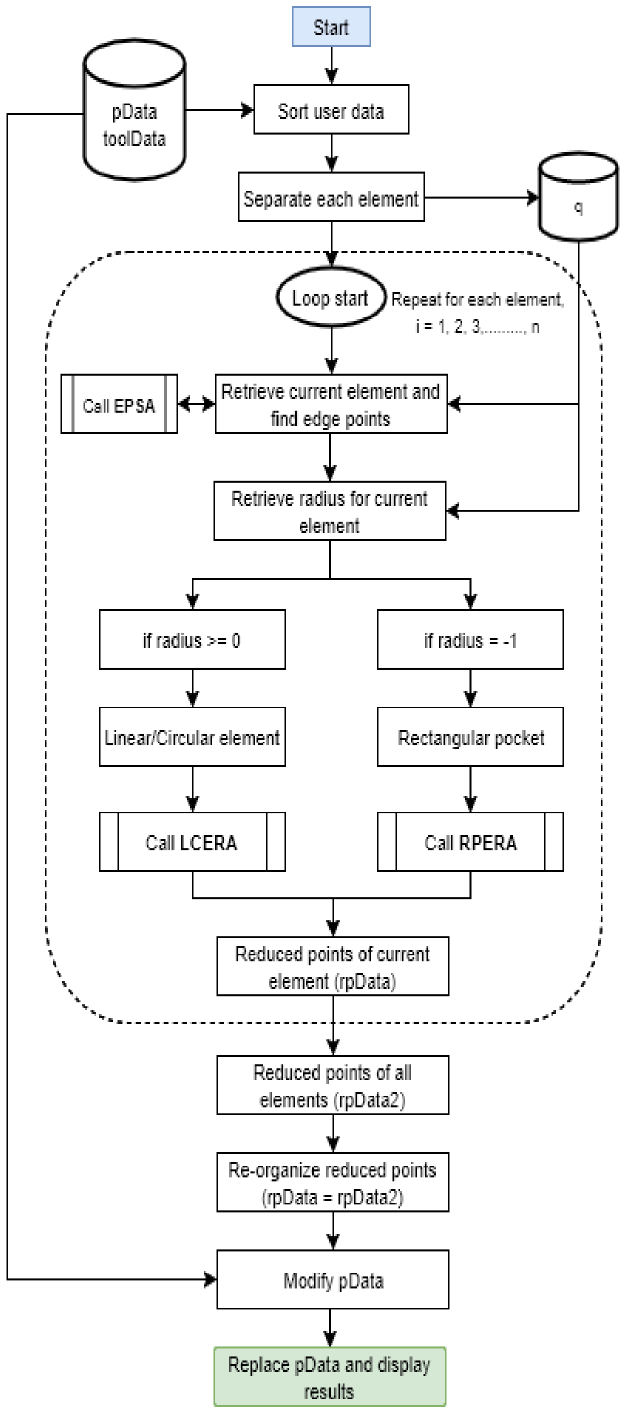
Start
pData
toolData
Sort user data
Separate each element
q
Loop start
Repeat for each element,
i = 1, 2, 3,........., n
Call EPSA
Retrieve current element and find edge points
Retrieve radius for current element
if radius >= 0
if radius = -1
Linear/Circular element
Rectangular pocket
Call LCERA
Call RPERA
Reduced points of current element (rpData)
Reduced points of all elements (rpData2)
Re-organize reduced points (rpData = rpData2)
Modify pData
Replace pData and display results

Figure 9. Flow-chart of main reduction algorithm.

Decision maker (DM) transfers complete vars structure and required data is extracted by reduction algorithm through simple commands. From this structure, pData and toolData are taken out and stored in local storage spaces. In the next step, each element is separated from pData along with complete construction and machining attributes. This separation is performed easily as the element number (eCounter) of an element is associated with every point of that element. The separated elements are stored in a cell named „q" where points bearing same eCounterare storedcollectively.

The separation of elements is necessary for execution of reduction algorithm. This is because it deals with each element separately and calls corresponding reduction function or algorithm. After separating the elements, they are transferred to main loop of the algorithm. This loop is executed till every element is reduced.

At the start of the main loop, one element is retrieved from q cell. It also supplies the radius value for the current element which distinguish the element as linear, circular or pocket. For this element, the points lying on the edges or corners are obtained through an algorithm named 'EPSA'. It stands for 'Edge PointSearch Algorithm', and decides which point are present on the corners (in lines and arcs) or on the edges (in rectangular pocket). It is worth mentioning that a reduction process is accelerated from opposite ends at the sametime.

Based on the radius of current element, decision to call a reduction algorithm for the element is made. For all linear elements, the radius is always kept 0, whereas for a rectangular pocket, it is 1. However, a circular arc always have a radius greater than zero. The reduction process of a line and arc is performed by asingle function named 'LCERA', 'Linear &CircularElementReductionAlgorithm'.Forapocket,it isperformed by 'RPERA', 'Rectangular Pocket ElementReduction Algorithm'.

These reduction algorithms perform reduction on the element and deliver reduced points. These reduced points represent the index values of R-active squares on the grid, and stored in an array rpData. Every element defined by the user passes through the loop by turn and reduced points are stored thereby. When all elements have been processed, the loop terminates. At the termination, the reduced points of every element stored in rp Data are transferred collectively to a new array rpData2 while previous array gets deleted. The rpData2 serves as a temporary storage only and after organizing the data points in correct way, they are transferred again to newly created rpData array.

At the end of this reduction process when all reduced points are retrieved from the reduction loop, it can now be used to modify previous data of all elements. It can be noted that rpData contains only X-Y coordinates of the points and no other data. The pData, which contains all the data in combined form, is modified according to this array. As a safety precaution, pData is not modified directly, but first a copy of this array is prepared. This new array is then modified as reduced points and the data concerned is kept and other points as well as their data is deleted. After completion of modification process, which is performed in a loop, provisions are made to check data validity and other discrepancies. At last, results are shown on screen and the modified array replaces old pData, thus reducing the number of points or say, number of active squares.

3.1 Edge Points Search Algorithm(EPSA)

The primary purpose of this search algorithm is to determine the points lying on the corners or at edges. We have discussed above how reduction process takes place. The process starts simultaneously from opposite ends, i.e. from both end points in case of line and arcs. In case of rectangular pocket, the reduction process is performed in two rounds, i.e. first from both upper and lower edges, and then from both left and right edges. Therefore, determination of these corners and edge points becomes a necessity.

The data related to the element under reduction is transferred to EPSA in the main reduction loop. After receiving the data, radius is checked to determine the type of current element. If it is a line or arc, then the two end points are placed in an array (Edges) and transferred back to the main loop. It is a simple process and does not take much computational efforts. But to process a pocket, a loop is initialized and all points of the pocket element are passed through theloop.

It is important to understand default sequencing of the points of an element first. When the grid is mapped as a matrix array and active squares are assigned a value of 1, the location or index of these active squares are stored in pData array. The points are arranged row-wise in the array along with their associated data, i.e. all points in first row, then second row, and so on. It can be observed that all points in a row are collinear as well as adjacent to each other. This property of point arrangement is useful for determining the points lying on the edges. While in loop, three consecutive points are selected simultaneously at every iteration. These points are then passed through two conditions related to adjacency and collinearity. It is evident that condition of adjacency and collinearity breaks between last points of one and first point of the second row. For instance, points in first row are all collinear as well as adjacent from 1 to 5. The last pair of three point satisfying this condition is of points 3, 4 and 5. The next pair would be 4, 5 and 6, in which first two points are adjacent (4 and 5) and three points are not collinear. This is the first condition. For every such pair of three points where none is collinear and first two are adjacent, the middle point belongs to the right edge and hence stored in C2 array. Similarly, if none are collinear and last two points are adjacent, then the middle point belongs to the left edge which is stored in C1 array. This procedure is repeated until every point the element is processed. Thepointsshadedinred(6,11,16,21)arestored in C1 and those in blue (5, 10, 15, 20) are stored in C2. In this process the first and the last points, viz. point 1 and 25 cannot be checked through adjacency and collinearity conditions as they are no points to pair with. Therefore, the first point is added at top of C1 and the last point is added to the bottom of C2 to obtain complete left and rightedges.

The above loop derives only left and right edges from given points of the element. No complex procedures are needed to obtain the upper and lower edges. The points in first row of the element form the lower edge, excluding the end points. Similarly, the points in the last row between two end points form the upper edge. It is worth mentioning here that a point included in left or right edge need not be included in upper or lower edge. Thus, points 2, 3, and 4 form the lower edge, while 22, 23 and 24 form the upper edge.

All the three edges obtained in this way are stored in different arrays named edge1, edge2, edge3, edge4. These arrays are then combined in a single array named Edges and transferred to the main reduction algorithm for further processing.

3.2 Linear and Circular Element Reduction Algorithm(LCERA)

The reduction process is described in previous sections. Reduction of a linear element as well as circular element are carried out by a single algorithm function. This is mainly due to the fact that both the entities have two end points and interpolated data points in between. Reduction is initialized from both end points at the same time. This reduction procedure of the elements is called 'Linear and Circular Element Reduction Algorithm (LCERA)'. This can be regarded as a sub-function of main reduction algorithm. This sub-function is completely controlled by main algorithm and called uponrequirement.

LCERA is complex algorithm constituting a number of calculations and processes through loops and conditions. The working of LCERA is shown in Figure 13. It is called by the main algorithm when it encounters a linear or circular element and transfers data of elements in an array Element. The radius of tool drives the process forward eliminating the points lying inside the tool vicinity. For reduction of a line or arc, the tool radius must be larger than the distance between the end points of line or arc. If it is greater than the distance, all the points would lie inside its vicinity and hence every point would be eliminated. To avoid this problem, a condition is applied at the beginning of main loop of LCERA to check the radius value of tool against the distance between endpoints.

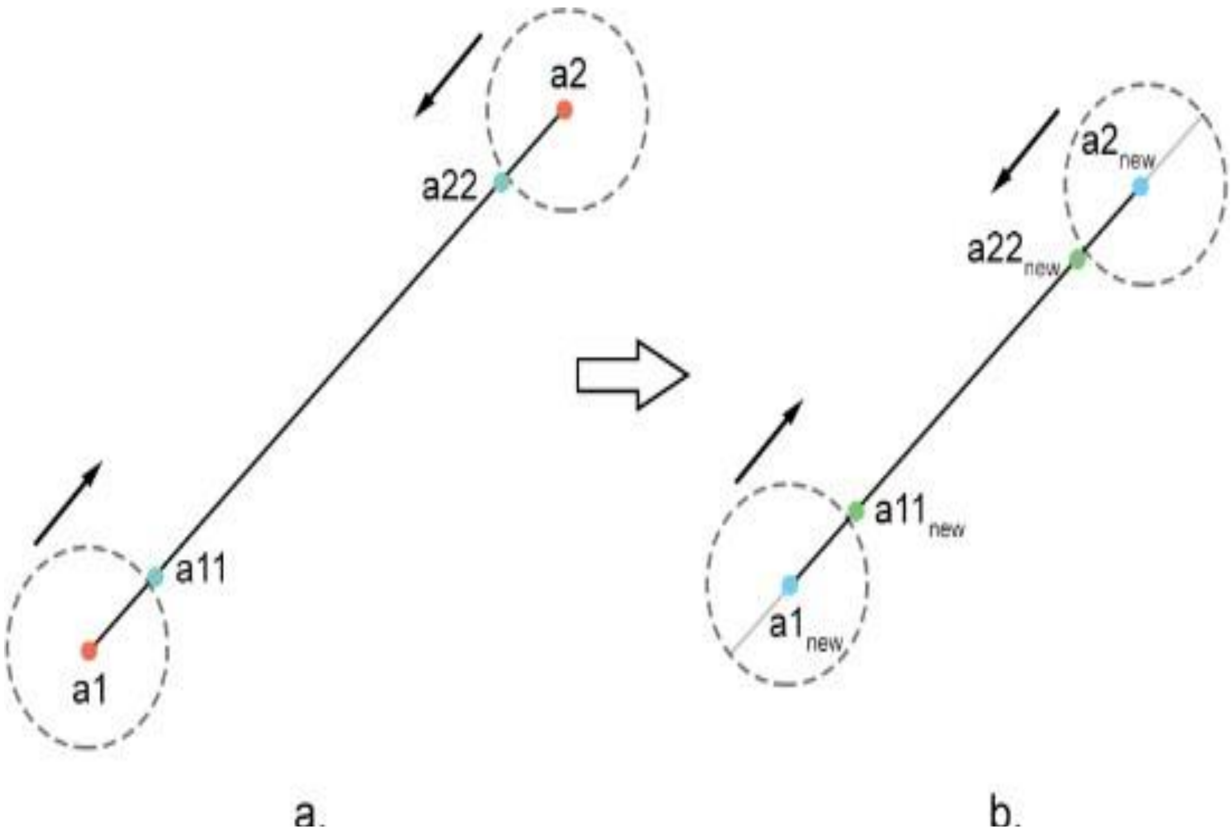

Figure 12. Mechanism of LCERA.

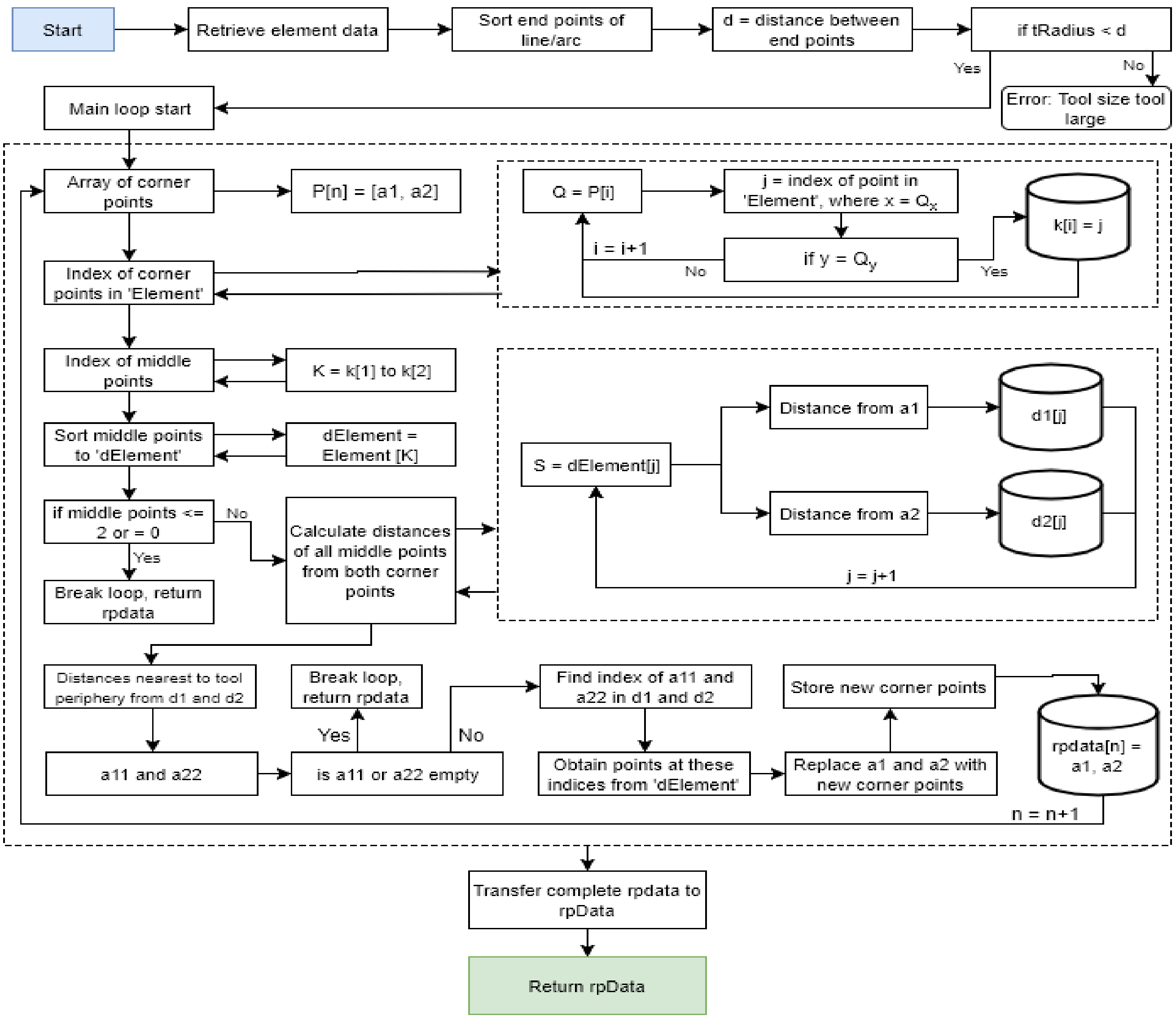
Start
Retrieve element data
Sort end points of line/arc
d = distance between end points
if tRadius < d
Yes
No
Error: Tool size tool large
Main loop start
Array of corner points
P[n] = [a1, a2]
Q = P[i]
j = index of point in 'Element', where x = Q_x
k[i] = j
i = i+1
No
if y = Q_y
Yes
Index of corner points in 'Element'
Index of middle points
K = k[1] to k[2]
Sort middle points to 'dElement'
dElement = Element [K]
Distance from a1
d1[j]
S = dElement[j]
Distance from a2
d2[j]
if middle points <= 2 or = 0
No
Yes
Calculate distances of all middle points from both corner points
Break loop, return rpdata
j = j+1
Distances nearest to tool periphery from d1 and d2
Break loop, return rpdata
Find index of a11 and a22 in d1 and d2
Store new corner points
rpdata[n] = a1, a2
Yes
No
a11 and a22
is a11 or a22 empty
Obtain points at these indices from 'dElement'
Replace a1 and a2 with new corner points
n = n+1
Transfer complete rpdata to rpData
Return rpData

Figure 13. Flow-chart and working of LCERA.

In this process, reduction is initialized from both the end points simultaneously. The end points are considered as ‘cornerpoints’whileremainingpointsbetweenthemare termedas‘middlepoints’.Forinstance,inFigure12(a),a1and a2 (in red) are the corner points while others are middle points. These points are sorted in the beginning by algorithm. After selecting the corner points for current iteration, the search for next corner points is triggered. The search is carried out from the middle points only, ignoring any other points which lie out of bounds of both end points. To get the indices of middle points, indices of corner points are determined first. The indices between these two values denote the middle points. All these middle points are stored in an array named dElement, copied from Element. This is a temporary array, and the search of next corner points is performed involving only the points stored here. To do so, the distances of all points in dElement from both corner points is calculated. This is done in a loop and the values are stored in arrays d1 and d2, where d1 contains value of distances of all dElement points from first corner point, and d2 from second corner point. From all these distances, the value of distance nearest to the tool periphery are obtained from both d1 and d2, and stored in a11 and a22. It can be noted that the distance stored in a11 is closest to a1 while that in a22 is closest to a2. Before progressing further, a termination condition is set up at this point. The condition implies that a11 and a22 must not be empty. If a11 and a22 are empty it means that no point is closest to a1 and a2, respectively. This condition is used to terminate the loop and return values obtained so far. The condition checks the values of thus obtained closest distances (a11 and a22). There might be some cases where no such point exists in dElement which have the distance as per above specified condition. This happens whenever all points of dElement lie within the vicinity of the tool, or the two corner points are so close to each other that there is no middle point in existence (in dElement). In these two cases, it is concluded that all points of the element have been processed and no further reduction can take place. The arrays a11 and a22 in termination cases are found to beempty.

In the next step after termination criterion in overlooked, the points located at these closest distances are obtained by first finding the location/index of these distances in d1 and d2 arrays and then getting the points from dElement lying at those locations. These new obtained points are then stored in a11 and a22, replacing previously stored distance values. Thus, the points closest to tool periphery from both end points are a11 and a22, shown in Figure 12(a) (in blue). These two points are stored in a local array namedrpdata.

Both corner points and middle points keep changing at every iteration. The new points obtained, a11 and a22, replace previous corner points a1 and a2. The next iteration continues considering new corner points. In this iteration middle points would limit to new corner points only. Further corner points obtained in this iteration would again be stored in rpdata and the process continues. In this way, till the last iteration of the loop, a set of corner points is obtained. These are termed as the reduced points of the element. The linear element shown in Figure 12 would be reduced after three iterations. The points obtained at successive iterations i1, i2 and i3 are shown in Figure 14. In this case, the loop terminates after iteration i3 is completed. After i3, all the remaining points would lie in the vicinity of the tool or there may be no further points. Thus, the termination criterion is satisfied and points obtained till i3 iteration in rpdata are returned by thealgorithm.

The array rpdata contains all points obtained through iterations. This array is then copied to a global array named rpData which is then returned back to the main reduction algorithm. This loop is able to reduce a linear and circular element of any size and dimensions. The algorithm is called bythe main reduction algorithm and is not available explicitly to other algorithms and functions.

3.3 Rectangular Pocket Element ReductionAlgorithm (RPERA)

The reduction algorithm developed for reducing a rectangular pocket element is ‘Rectangular Pocket Element ReductionAlgorithm(RPERA)’.Thisalgorithmissimilarin function to LCERA. The key difference between the two algorithms is that while LCERA deals with one corner point from both ends, RPERA deals with a set of corner points first at left-right edges and then upper-lower edges. This set of corner points contains the points lying at an edge of the pocket. These edge points are supplied by Edge Points Search Algorithm (EPSA). All points lying on one edge are processed simultaneously by the algorithm. Therefore, this algorithm comprises of a number of loops, more than that in LCERA. Also there is an increase in computational complexities due to presence of loops, data storage arrays and a large number of calculations. As a pocket contains vast number of active squares, this algorithm has to deal with a rich space. Due to these reasons, it became obligatory to make this algorithm as compact and effective as possible to avoid any discrepancies. It can be said that RPERA is one of the most complex algorithm of the wholeframework.

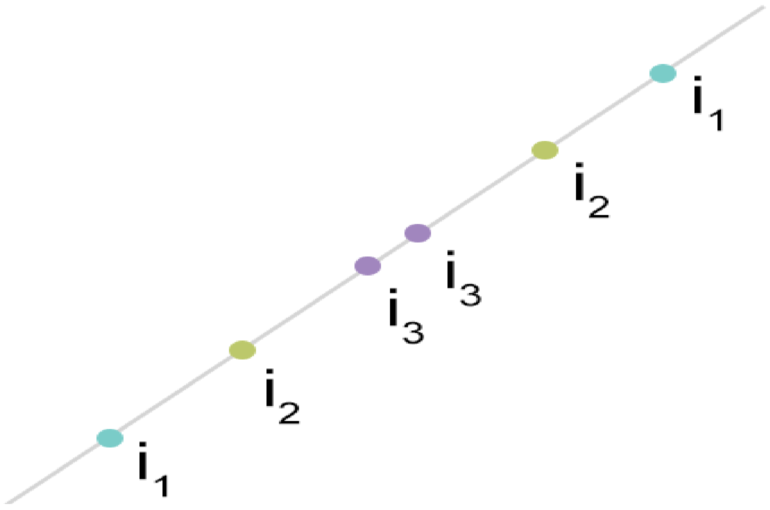

Figure 14. Reduced points obtained at successive iterations.

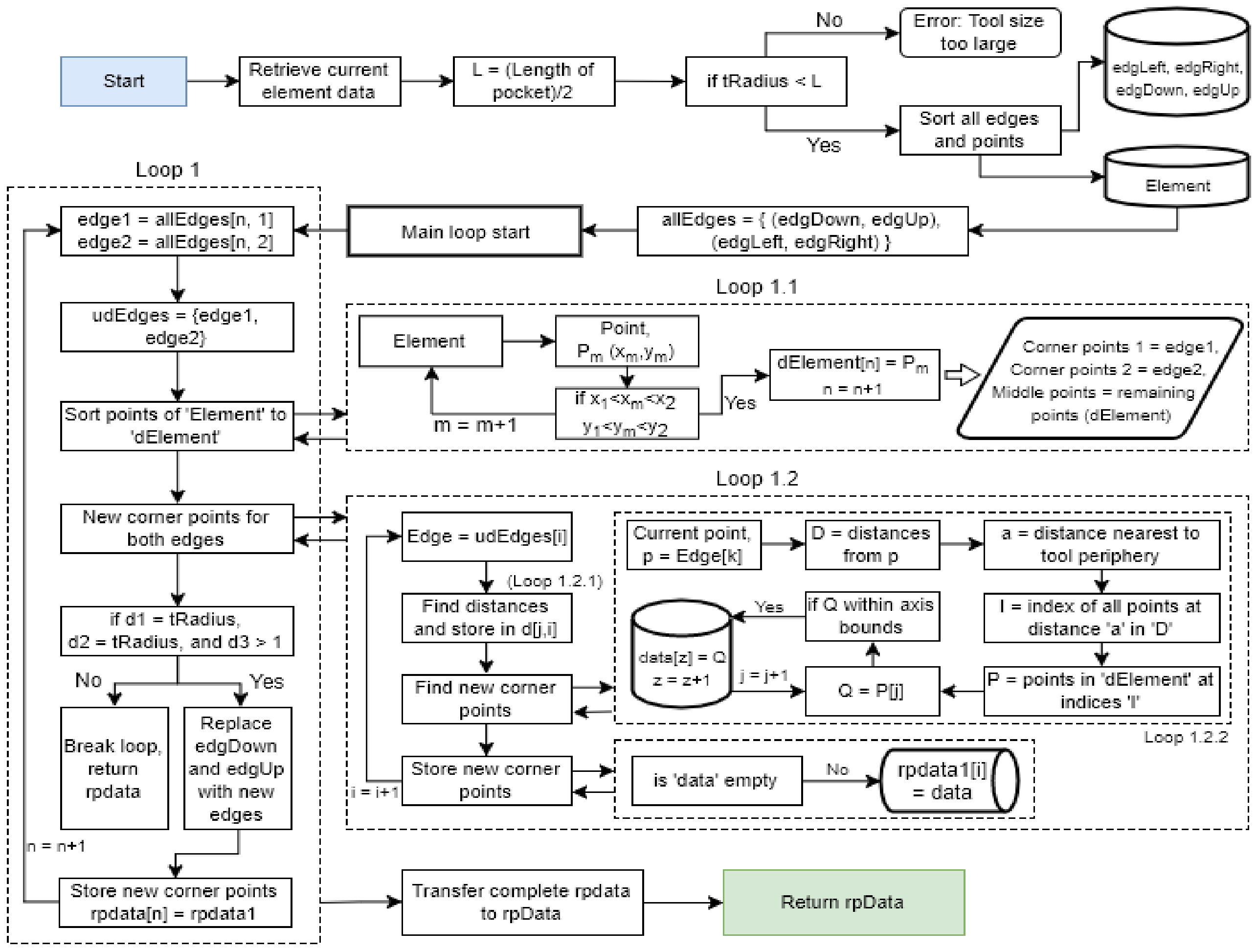
Start
Retrieve current element data
L = (Length of pocket)/2
if tRadius < L
No
Error: Tool size too large
Yes
Sort all edges and points
edgLeft, edgRight, edgDown, edgUp
Element
allEdges = { (edgDown, edgUp), (edgLeft, edgRight) }
Main loop start
Loop 1
edge1 = allEdges[n, 1]
edge2 = allEdges[n, 2]
udEdges = {edge1, edge2}
Sort points of 'Element' to 'dElement'
New corner points for both edges
if d1 = tRadius, d2 = tRadius, and d3 > 1
No
Yes
Break loop, return rpdata
Replace edgDown and edgUp with new edges
n = n+1
Store new corner points rpdata[n] = rpdata1
Loop 1.1
Element
Point, P_m (x_m, y_m)
if x_1<x_m<x_2 y_1<y_m<y_2
m = m+1
Yes
dElement[n] = P_m
n = n+1
Corner points 1 = edge1, Corner points 2 = edge2, Middle points = remaining points (dElement)
Loop 1.2
Edge = udEdges[i]
(Loop 1.2.1)
Find distances and store in d[j,i]
Find new corner points
Store new corner points
i = i+1
Current point, p = Edge[k]
D = distances from p
a = distance nearest to tool periphery
I = index of all points at distance 'a' in 'D'
P = points in 'dElement' at indices 'I'
Q = P[j]
if Q within axis bounds
Yes
data[z] = Q
z = z+1
j = j+1
Loop 1.2.2
is 'data' empty
No
rpdata1[i] = data
Transfer complete rpdata to rpData
Return rpData

Figure 15. Flow-chart and working of RPERA.

The working of RPERA is shown inFigure 15. This algorithm, similar to LCERA, is completely controlled by the main reduction algorithm. The main algorithm supplies necessary information related to the element and tool in the form of an array Element. Here also, a termination condition at the beginning of the main loop is present. This condition is similar to the first termination condition of LCERA. In this case, the size of the tool is compared with the length or width of the pocket (whichever is the shortest). For a successful reduction, the radius of tool must be less than half the length or width of the pocket, L. Further execution takes place if this condition is not satisfied, otherwise an error is issued to the user informing about tool size. In the next step, all edges in Element are sorted and re-organized. The edges retrievedfrom EPSA,i.e. edge1, edge2, edge3, and edge4, are sorted as edgLeft, edgRight, edgDown and edgUp, respectively and stored againinElement.The edge is also created to assist the loop in creating opposite edges simultaneously.

As indicated in Figure 15, a number of loops are present i the algorithm. Of them, Loop 1 is the main loop that executes all other sub- loops such as Loop 1.1 and Loop 1.2. There are also two sub-loopofLoop1.2,indicatedasLoop1.2.1andLoop 1.2.2. The main loop is executed twice, once for upper-lower edges and again for left-right edges. Each time, two oppositeedgesareselected,startingfromupperandlower

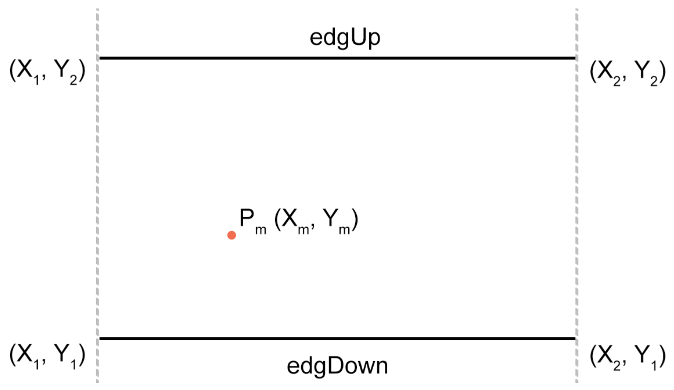

Figure 16. Axis bounds (upper and lower edges).

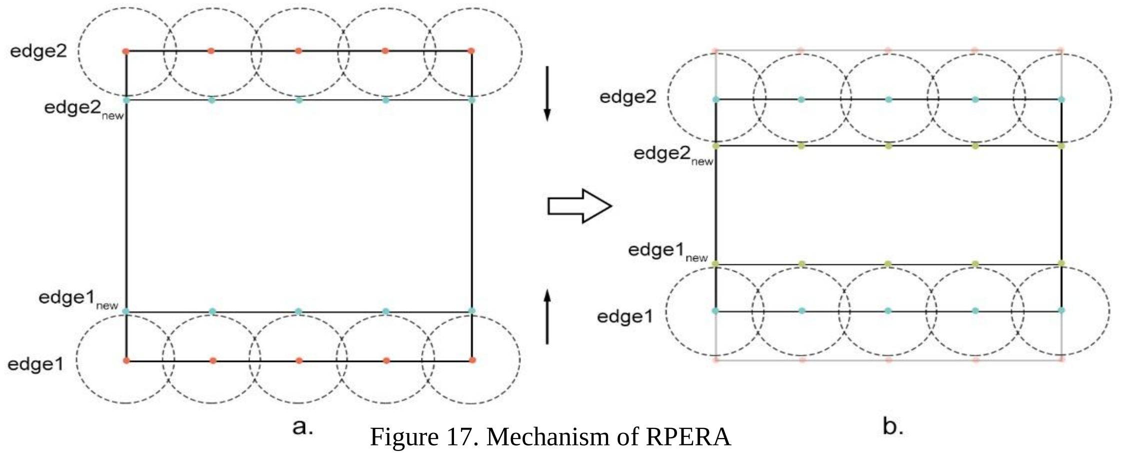

Figure 17. Mechanism of RPERA

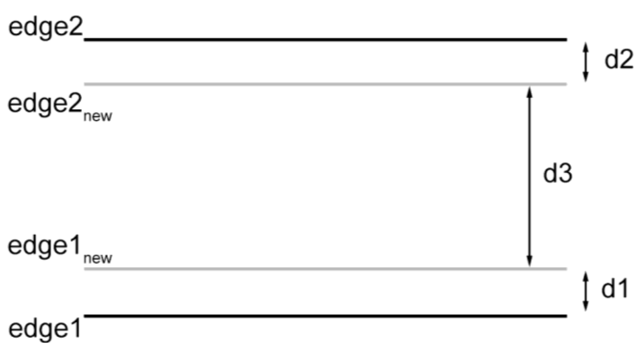

Figure 18. Various distances between new and old edges.

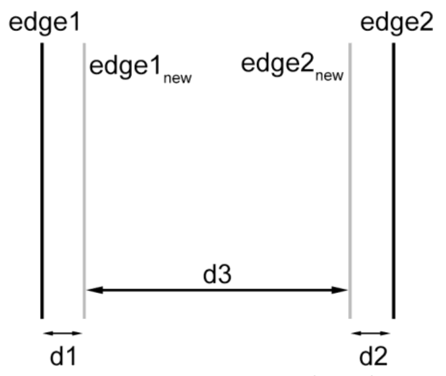

Figure 19. Various distances for left-right edges.

Edges. This set of edges is retrieved from allEdges array as shown in Figure 16, first edge (edgDown) is copied in edge1 and second (edgeUp) in edge2. Both these edges are then stored in a new array named udEdges. The points lying on both these edges are considered as the corner points. Similar to LCERA, the middle points between upper and lower edges are copied to dElement from Element. The next corners are to be searched from dElement. For sorting of elements to dElement, a sub loop of the main loop (Loop 1.1) is used. In this loop, at every iteration a points from Element is selected, Pm with coordinates (Xm, Ym). The condition for entry of a point in dElement is applied to this point. According to this condition, the x-coordinate of the point, Xm, must lie between minimum and maximum value of x-coordinates of an edge (i.e. X1 and X2). Similarly, the y-coordinate of the point, Ym, must lie between minimum and maximum values of y-coordinate of both edges (i.e. Y1 and Y2), as shown in Figure 16. All points satisfying this condition are copied from Element to dElement.

In the next step new corner points are searched from dElement, Loop 1.2 is employed for this purpose. This loop also consists of two iterations, one for each edge. In the first iteration, first edge stored in udEdges is selected, i.e. edge1. Now, from all the corner points which constitute this edge distance of all middle points is calculated. This requires a lot of computational time as the number of points under processing is very large. Loop 1.2.1 performs these calculations;foreachcornerpoint‘I’,thedistanceofamiddlepoint s„j"isstoredinanarraydj,i.Asamatteroffact, for first corner point the distances of all middle points are kept in the first column of d, and for second corner point, in second column, and so on. After every point has been processed, number of columns of d would be equal to the number of corner points, and number of rows would be equal to total number of middle points in dElement. As the distance of every middle point from corner points has been calculated, new corner points can now be obtained using these distances. This process is similar to that in LCERA where points closest to the tool periphery are selected for a corner point on both sides. Here in this case, all corner points of edge1 are processed simultaneously as compared to LCERA. In other words, points closest to tool periphery are obtained for every corner point one by one. This is accomplished through a loop (Loop1.2.2).

Foracornerpoint‘p’,correspondingcolumnofdistances is copied from multi-column array d and stored in a temporary single-column array D. This array changes at every iteration as values of distances for current point is copied to it. From the values of distances the closest distance (a) to tool periphery is searched and its index inthe column is saved in array I. There can be multiple values of a because of more than one point satisfying the above condition of closeness. In that case the number of indices would be more than one, that is, I would contain more than one index. The points present at these indices in dElement would be the next possible corner points. A possible corner point means that it has to pass through a condition to be selected as a corner point. Note that one condition of axis bounds has already been applied while copying points from Element to dElement as discussed previously. The next condition implies that the new corner points must not lie on the same axis as old corner point (y-axis in case of upper-lower edges and x-axis for left-right edges). The points not satisfying this condition are eliminated while other are retained and stored in array data. Loop 1.2.2 processes all corner points of edge1 and new corner points are stored in data array. In the next step, these corner points are transferred to array rpdata1 while array data is erased to be used for next iteration. The second iteration of Loop 1.2 takes place considering edge2 now. The same process is repeated for all corner points of edge2 in Loop 1.2.1 and Loop 1.2.2. After processing of all corner points the values are stored in data array. The new corner points for edge2 are then transferred to rpdata1 array.

These new corner points for both the edges constitute new edges as shown in Figure 17. At this point in Loop 1, a termination condition is applied. This is the only termination condition in entire algorithm. To check the condition, three types of distances are calculated between old edges and new edges. These distances are shownin

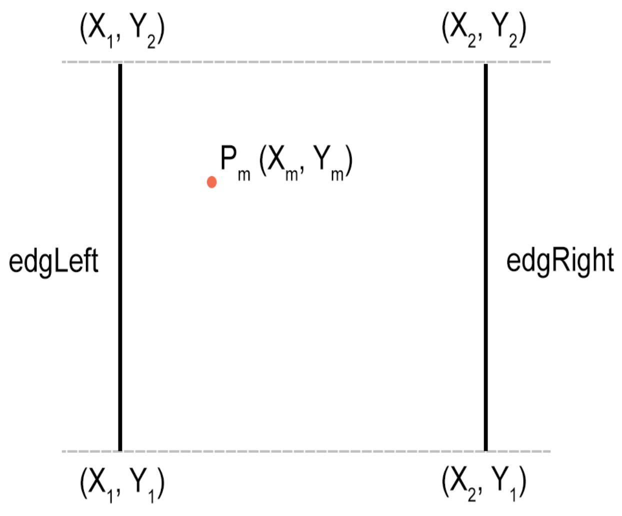

Figure 20. Axis bounds for left-right edges.

Figure 18, namely d1, d2 and d3. The distance d1 is calculated between y-coordinate of edge1new and edge1, d2 is calculated between y-coordinate of edge2 and edge2new, and the distance d3 is calculated between both the new edges, i.e. edge2new and edge1new. These distances are used to check the validity of newly obtained edge. For a new edge or set of two edges to be valid, d1 and d2 must be equal to the tool radius, and d3 must be greater than 1 not zero or negative. If the set does not qualify this validity test, termination condition forces the loop to break and halt further execution. In case when the edges are found to be valid, they replace both old edges; after which they are stored in new array named rpdata (not to be confused with rpdata1 array, see note on next page). This array is the main storage array of Loop 1 where all newly obtained edges (corner points) are stored.

The new points obtained in this iteration are considered as corner points for the next iteration. Taking these into consideration, above procedure is repeated to get another set of

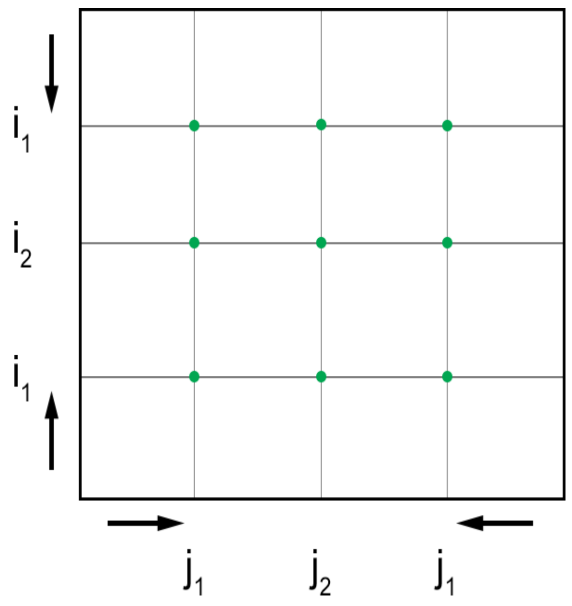

Figure 21. Reduced points obtained after completion of algorithm.

corner points. This process is repeated a number of times to obtain sets of corner points. Only when the edges formed by corner points are close enough that no further reduction is possible, termination criterion halts the processing. All the corner points obtained through iterations are stored in rpdata. Note that these pointsarethereducedpointsforupper-loweredges.Inthe next step, second iteration of Loop 1 is initialized. This time the edges to be considered are left-right edges instead of upper-lower. It has discussed previously that reduction for upper-lower edges takes place considering all points of Element. From these points, reduced points are obtained. Now, for left-right edges not all points are considered but reduced points of upper-lower edges are taken into account. In this way, points reduced in vertical direction would now be reduced horizontally. Hence, left and right edges are retrieved from allEdges and stored in edge1 and edge2, respectively. Furthermore, edge1 and edge2 are transferred to udEdges as in previous iteration.

Next step follows copying points to dElement for further processing. As stated above, the points are copied from rpdata which contains previously reduced points. From corner points of left-right edges, new corner points are obtained through Loop 1.2. Same procedure is repeated in this case as for upper-lower edges. Loops 1.2.1 and 1.2.2 are executed similarly to find corner points. The conditions of axis bounds remain the same in this case too apart from a few modifications. These condition are shown in Figure 19. The only visible change is the orientation of the edges, other bounds remain similar to previously discussed bounds for upper-lower edges. Based on these conditions, points from rpdata are copied to dElement which is then followed by calculation of distances. The closest distances are determined and corresponding points from dElement are obtained which constitute new edges. But these edges have to be valid to be used for future iterations. Therefore, a validation test similar to previous case is used, shown in Figure 19. To qualify as valid edges, values of d1 and d2 must be equal to the tool radius, and d3 must be greater than 1. Edges qualifying these conditions then replace the old edges (corner points) and serve as edge1 and edge2 for next iteration. These edges or points are stored in rpdata1 and after completion of all iterations, complete set of points is stored in rpdata array. In this way the points obtained after reduction of upper-lower edges are again reduced with respect to left-right edges.

After all iterations have been completed and all corner points are obtained, they are then transferred to array rpData for output. We have seen that reduction of linear or circular elements (LCERA) yields a set of points at some distance from each other. In case of rectangular pocket (RPERA), a grid pattern of reduced points is obtained, which means the points lie in two directions. Figure 21 shows the reduced points of the rectangular pocket. It can be seen that all points of the rectangular pocket have been reduced to the points shown (in green). These points are obtained after reduction in context of upper-lower as well asleft-right edges. In both the case, two iterations proved to be enough for reduction. The iterations for upper-lower edges are i1 and i2 while that for left-right edges are j1 andj2.

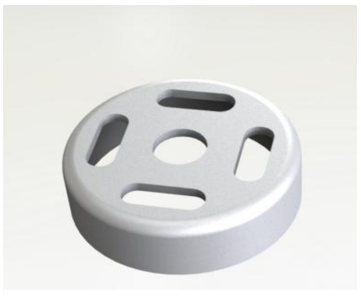
Figure 22. A clutch bell (3D Model).

It is evident from Figure 21 that a small number of points are obtained as compared to actual number of points in a rectangular pocket. RPERA is able to reduce a rectangular pocket of any size efficiently. The algorithm is more sophisticated than LCERA as the former deals with lesser number of points. The chances of error have been reduced by incorporating various conditions and checks inthe

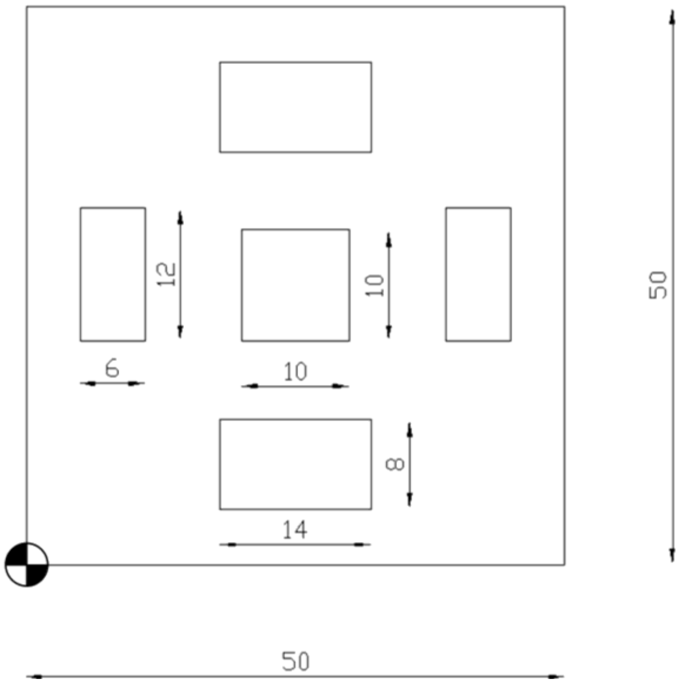

Figure 23. Part for experimentation and testing.

algorithms. This makes the algorithm well suited for reduction purposes. Apart from the main reduction algorithm, we have discussed three algorithms, namely EPSA, LCERA and RPERA. These are the backbone of reduction process. Every algorithm does its pre-defined work based on a set of rule and procedures. These algorithms together make reduction process a powerful tool and one of the most complex procedure in the entire framework.

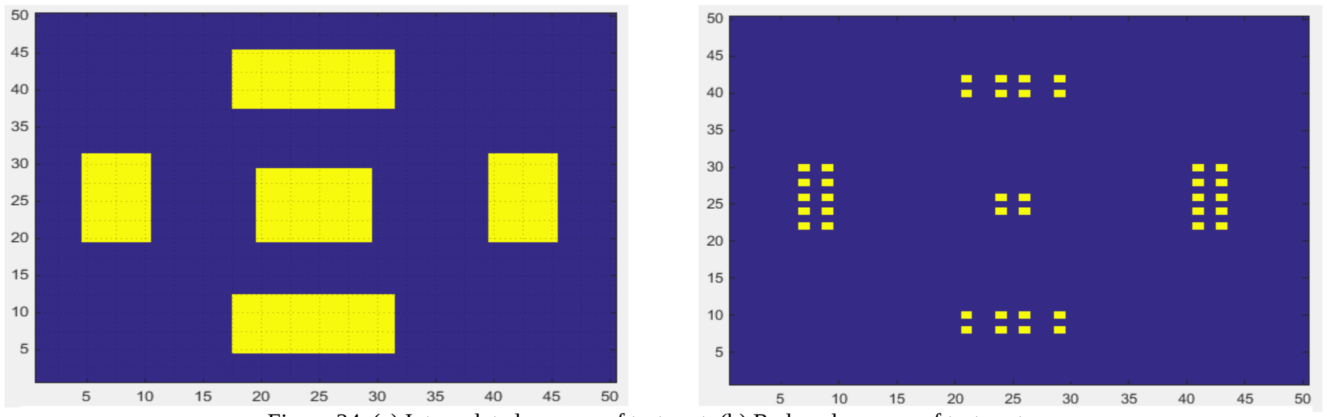

Figure 24. (a) Interpolated squares of test part, (b) Reduced squares of test part.

IV RESULTS AND DISCUSSION

To test the validity and performance of the above algorithms, a genetic algorithm (GA) based optimization of pocket milling is considered. The workpiece inspired by a clutch bell (Figure 22) of size 50x50 mm is defined and five rectangular pocket elements are added as shown in the Figure 23.

After defining the dimensions and other parameters, interpolation algorithms were used to model the test part as shown in Figure 24a. The reduction algorithms were then implemented to decrease the number of discrete squares as shown in Figure 24b. It is evident from Figure 24 that, the reduction algorithm significantly decreased the discrete squares required to model the testpart.

The reduction algorithm is a key component of the proposed framework. It reduces the number of squares and hence the search space by considering tool offsets. The reduction of an element is performed on the basis of tool size defined by user for that element. A number of experiments have been conducted test the levels of reduction and its effects of computational time and convergence of the solution ofGA.

A large number of squares increase the length of chromosomes in a GA population. This large-sized population thus takes a long time to achieve convergence as slow improvement takes place. Convergence is largely affected by the length of chromosome, size of population, and other factors such as crossover and mutation rates. The results of experiments are shown below in Table 1

Table 1. Experiment results showing number of squares and convergence time.

Sr. No.	Element Size (mm)	No. of squares (Interpolation)	No. of squares (Reduction)	Time1	Time2
1.	10 x 10	121	16	>3 hrs.	85 sec
2.	12 x 12	169	25	>5 hrs.	160 sec
3.	20 x 20	441	36	>10 hrs.	7.5 min
4.	20 x 20	441	16	>10 hrs.	85 sec
5.	30 x 30	961	81	>1 day	20 min
6.	30 x 30	961	64	>1 day.	12 min

R
E
F
E

It is evident from the table that a large number of squares are generated by interpolation. Even a small pocket of 1cm size is defined by 121 squares. So, in this case, the length of one chromosome would be 121. To process such as large size chromosomes, calculate their fitness values and reproducing offspring becomes very difficult and time consuming. As there are a number of such chromosomes present in the population, it is no less than a nightmare for any optimization technique. GA being a probabilistic approach would keep on testing and recombining chromosomes in search for at least satisfactory fitness levels. In fact, search space becomes very large and chances of finding global optimum or even converging to a solution arereduced.

The reduction algorithm is shown to have a positive effect on both the search space size and convergencetime.Areductionofsearchspaceby85to90 % is observed depending upon the number of elements and size of tools. For example, in the first experiment, a pocket having 121 squares is reduced to only 16 squares and convergence time is reduced from more than 3 hours to only 85 seconds. A drastic reduction in convergence time is thus observed from the table. Time is reduced by ~ 98% in all the experiments meaning that convergence is achieved in very less time after performing reduction. This allows to optimize toolpaths for parts containing large design elements in comparatively less time. A task which would otherwise require hours of computational times even on high-end CPUs.

V CONCLUSION

In this paper, algorithms for interpolation and reduction of discrete elements were presented. These algorithms are utilized in approximation and modeling of work area in pocket milling. The algorithms are developed for linear element, circular element and rectangular pocket element. It was observed that interpolation results in large number of discrete squares which in turn increases the search space and hence convergence time of optimization method. The reduction process allows decreasing of discrete squares by considering tool offset values. The results indicate a huge reduction in the search space as well as convergence time of optimizationmethods.

REFERENCES

1.R. Saravanan, P. Asokan, and M. Sachithanandam, "Comparative analysis of conventional and non-conventional optimisation techniques for CNC turning process," Int. J. Adv. Manuf. Technol., vol. 17, no. 7, pp. 471–476, 2001.

2.M. Mendes, M. D. Mikhailov, and R. Y. Qassim, "A mixed-integer linear programming model for part mix, tool allocation, and process plan selection in

CNC machining centres," Int. J. Mach. Tools Manuf., vol. 43, no. 11, pp. 1179–1184, 2003.

3.W. Fan, X. S. Gao, C. H. Lee, K. Zhang, and Q. Zhang, "Time-optimal interpolation for five-axis CNC machining along parametric tool path based on linear programming," Int. J. Adv. Manuf. Technol., vol. 69, no. 5–8, pp. 1373–1388,2013.

4.Munish Kumar and Pankaj Khatak, "An Investigation of Conventional and Non-Conventional Optimization Techniques in CNCMachining"

5.A. I. Sonomez, A. Baykasoglu, T. Dereli, and I. H. Filiz, "Dynamic optimization of multipass milling operations via geometric programming," Int. J. Mach. Tools Manuf., vol. 39, pp. 297–320, 1999.

6.P. H. Wu, Y. W. Li, and C. H. Chu, "Optimized tool path generation based on dynamic programming for five-axis flank milling of rule surface," Int. J. Mach. Tools Manuf., vol. 48, no. 11, pp. 1224–1233, 2008.

7.M. Kovacic, M. Brezocnik, I. Pahole, J. Balic, and B. Kecelj, "Evolutionary programming of CNC machines," J. Mater. Process. Technol., vol. 165, pp. 1379–1387, 2005.

8.J. Barclay, V. Dhokia, and A. Nassehi, "Generating Milling Tool Paths for Prismatic Parts Using Genetic Programming," Procedia CIRP, vol. 33, pp. 491–496, 2015.

9.A. Nassehi, W. Essink, and J. Barclay, "Evolutionary algorithms for generation and optimization of tool paths," CIRP Ann. - Manuf. Technol., vol. 64, no. 1, pp. 455–458, 2015.

10.S. Moshat, S. Datta, A. Bandyopadhyay, and P. Pal, "Optimization of CNC end milling process parameters using PCA-based Taguchi method," Int. J. Eng. Sci. Technol., vol. 2, no. 1, pp. 95–102, 2010.

11.I. Asiltürk and S. Neşeli, "Multi response optimisation of CNC turning parameters via Taguchi method-based response surface analysis," Meas. J. Int. Meas. Confed., vol. 45, no. 4, pp. 785–794, 2012.

12.S. L. Omirou, "Space curve interpolation for CNC machines," J. Mater. Process. Technol., vol. 141, no. 3, pp. 343–350, 2003.