An improved method for calculating the no-fit polygon

Hamish T. Dean*, Yiliu Tu, John F. Raffensperger

200 Armagh Street, P.O. Box 13-761, Christchurch, New Zealand

Abstract

The no-fit polygon (NFP) is the set of feasible locations that one polygon may take with respect to another polygon, such that the polygons do not overlap. Feasible locations are required for most of the solutions to two-dimensional packing problems, and also for other problems such as robot motion planning.

Efficient methods to calculate the NFP of two convex polygons, or one convex and one non-convex polygon have been developed by other researchers. However, when both polygons are non-convex, the current methods of calculation are inefficient or difficult to implement. This paper presents an extension of Ghosh’s (CVGIP: Image Understanding 54 (1991) 119) NFP algorithm, and uses manipulation of sorted lists of polygon edges to find the NFP efficiently.

Keywords: No-fit polygon; Two-dimensional packing; Minkowski sum

1. Introduction

An issue in two-dimensional packing is determining the set of feasible locations that one polygon may take with respect to another polygon, such that the polygons do not overlap. This set of locations is known as a no-fit polygon (NFP). The terms Minkowski sum, Φ-function, hodograph, dilation, envelope and configuration space obstacle have also been used by other researchers.

Let each polygon be represented by an ordered list of edges. The location of each polygon i in the two-dimensional plane is represented by a reference point, \( r_i \). The reference point is located at point (0, 0) of a polygon’s local coordinate system (see Fig. 1).
Cunninghame-Green [2] showed that for the case when polygons \( A \) and \( B \) are convex, \( \text{NFP}[A, B] \) can be created by ordering the edges of \( A \) and \(-B\) in increasing slope order. \( \text{NFP}[A, B] \)’s edges correspond exactly to this slope order (see Fig. 4).

When one or more of the polygons are non-convex, an obvious way of calculating the relevant NFP is to decompose each polygon \( i \) into a set of \( N_i \) convex sub-polygons \( (\text{CSP}[i])_1 \to (\text{CSP}[i])_{N_i} \). Overlap will occur between the two polygons if any sub-polygon of \( A \) overlaps any sub-polygon of \( B \). \( \text{NFP}[A, B] \) is the union of \( \text{NFP}[\text{CSP}[A]_i \cup \text{CSP}[B]_j] \), where \( i = 1 \ldots N_A \) and \( j = 1 \ldots N_B \).

There are two drawbacks to the polygon subdivision approach. Firstly, efficient algorithms are required for polygon decomposition and polygon composition. Secondly, it is possible that a non-convex polygon that has \( N \) edges in cavities (see Fig. 5) can be decomposed into no less than \( N \) CSPs. The NFP of two of these polygons would require the composition of \( N^2 \) sub-NFPs. Polygons used in industries such as garment manufacturing often have large numbers of edges in their curve-like cavities, and the sub-division
**Definition 2.2.** An edge $i$ of a polygon is a *turning point* if the sign of $\alpha(i)$ is opposite to the sign of $\alpha(i + 1)$.

Bennell et al. [7] states that a polygon is *convex* if and only if it does not contain any turning points. Otherwise it is *non-convex*.

The initial stage of Ghosh’s approach is to sort all the edges of polygon $A$ and polygon $B$ by slope into one list which we will call $\text{MergeList}$. If both polygon $A$ and $B$ are convex, then $\text{MergeList}$ gives the edge order for $\text{NFP}[A, B]$, and the method is equivalent to that of Cunninghame-Green [2].

Assuming polygon $A$ is non-convex, and polygon $B$ is convex, the method proceeds as follows:

Starting in $\text{MergeList}$ at the first edge of polygon $A$, visit the edges of $A$ in order, and add them to the list of edges ($\text{NFPList}$) which make up $\text{NFP}[A, B]$. If edge $A$ is a turning point, then the direction of travel along $\text{MergeList}$ is reversed. Any edges of $B$ which are passed are added to $\text{NFPList}$. $B$ edges are positive if the direction of travel forward, and negative if the direction is backward. This continues until the first edge of polygon $A$ has been returned to. The resulting $\text{NFPList}$ we will call $\text{GhoshList}$. The above algorithm is given in Pseudocode 2.1.

```plaintext
p = Position in MergeList which corresponds to A0
i = 0
Dir = 1

Loop{
  If MergeList[p].PolygonType = A Then
    If MergeList[p].PolygonIndex = i Then
      GhoshList = GhoshList + MergeList[p]
      If MergeList[p].IsTurningPoint = True Then Dir = Dir* - 1
      i = i + 1 (If i > A.Size Then i = 0)
    End If
  Else
    GhoshList = GhoshList + MergeList[p]*Dir
  End If
}

p = p + Dir
|While(i ≠ 0)

Pseudocode 2.1: Algorithm to find GhoshList
```

The process of finding GhoshList is seen easily with what Ghosh calls a slope diagram (see Fig. 6). The points on the diagram are at the slope of the edges of polygons $A$ and $B$.

Following around the slope diagram, starting and finishing at $A0$, mimics the process of traversing over $\text{MergeList}$. The outer envelope of $\text{NFPList}$ gives $\text{NFP}[A, B]$ (see Fig. 7).

Ghosh’s method works for all simple polygons (no holes) when polygon $A$ is non-convex and polygon $B$ is convex. The method also works when both polygons are non-convex, as long as no two cavities from either polygon interfere which each other. This occurs when an interval of $\text{MergeList}$ has wrongly ordered edges from both polygons. When this does occur, this interval must be traversed in two or more parallel paths. Although the theory of traversal by parallel paths holds true for complex non-convex cases, there are considerable implementation problems in sorting out the paths. These difficulties led Bennell et al. [7] to seek a different approach.
The calculation difficulties of Bennell’s method have motivated development of a more robust and efficient method of calculating NFPs. Like Bennell’s method, it exploits the fact that the NFP of a non-convex polygon and a convex polygon can be easily and efficiently found by Ghosh’s method. However, the new method does not use dummy edges to replace cavities of \( B \).

3. A new method

Intuitively, it would seem a good idea to modify Bennell’s method to start “looking” for the next \( B \) edge of a cavity once an occurrence of the current \( B \) edge has been found, instead of continuing to look for the furthest occurrence of that \( B \) edge. However, if there is more than one occurrence of a \( B \) edge in any given segment then this approach will run into difficulties.

A solution to this is to make sure that each traversal segment contains only positive or negative occurrences of each \( B \) edge of a particular cavity. Replacing a \( B \) cavity with a dummy edge \( D \) will not guarantee this (see Fig. 8).

Fig. 8 shows a dummy edge \( D \), whose cavity is composed of edges \( B1 \) and \( B2 \). In this example, Bennell’s method would require only one segment which would contain a single occurrence of \( D \). However, this segment contains both positive and negative occurrences of \( B1 \).

To guarantee that there is only positive or negative occurrences of a given cavity \( B \) edge, we split the traversal of GhoshList using the algorithm given in Pseudo-code 3.1:

\[
p = \text{The position in GhoshList which corresponds to } A0.
\]
\[
TravelDir = -1
\]
\[
CurrentSign = +1
\]
\[
TravelSign = +1
\]
\[
i = 1
\]

Loop1{
    \[
p = p + TravelDir
    \]
    If GhoshList[\( p \)].\( PolygonType \) = \( A \) Then
        If GhoshList[\( p \)].\( PolygonIndex \) = 0 And TravelDir = \( -1 \) Then
            \( \text{Seg}[i].End} = p \)
            Exit Algorithm
        Else If GhoshList[\( p \)].\( IsTurningPoint \) = True Then
            TravelSign = TravelSign* - 1
            \( \text{Seg}[i].Start} = p \)
        End If
    Else If GhoshList[\( p \)].\( IsInCavity \) = True And \( CurrentSign \neq TravelSign \) Then
        TravelDir = TravelDir* - 1
        \( p = \text{Seg}[i].Start \)
        TravelSign = TravelSign* - 1
        Exit Loop1
    End If
}

Fig. 9 shows the slope diagram for polygons A and B. It will be split into four segments, $A9 \rightarrow A1$, $A1 \rightarrow A3$, $A3 \rightarrow A6$, and $A6 \rightarrow A9$, using the method described earlier. For each of the four segments, we count the cavities of $B$ whose span intersects it. Cavity 1 intersects segments $A9 \rightarrow A1$, $A3 \rightarrow A6$, and $A6 \rightarrow A9$. Cavity 2 intersects $A9 \rightarrow A1$, $A1 \rightarrow A3$, and $A3 \rightarrow A6$. Each segment must “process” the cavities which intersect it.

To create NFPList with correct numerical ordering of both A and B edges, GhoshList and MergeList must be re-traversed together.
3.4. Traversal moves onto MergeList

If either the starting or ending limit, \( L \), of the current segment is reached, and all relevant cavities and non-cavity \( B \) edges have not been processed, then the traversal shall continue on MergeList rather than on GhoshList. The traversal starts on MergeList at edge \( L \) and continues searching for the current \( B \) edge in the direction that the traversal on GhoshList was taking. Any \( A \) edges encountered on MergeList are ignored. The traversal will return back to GhoshList when \( L \) is passed in the opposite direction of which MergeList was entered. \( L \) is added to NFPLList.

A degenerate case can occur which causes the traversal of MergeList to continue without ever returning to \( L \). The happens when, directly after the last edge of the final cavity of the current segment has been added to NFPLList, the traversal does not head back to \( L \) in the opposite direction to which MergeList was entered. Because this is the final cavity of the current segment, after processing this cavity, we want to move onto the next segment. And because the final cavity of the current segment is also the first cavity of the next segment, we can make the transition from the current segment to the next segment within MergeList. After the last edge of the final cavity is added to NFPLList, we add \( L \) to NFPLList (opposite sign to previously added \( L \)). The next segment becomes the current segment, and we traverse back over MergeList starting at the last edge of the final cavity (now the first edge) and add \( B \) edges to NFPLList in order until \( L \) is reached in the opposite direction of which MergeList was entered. \( L \) is added to NFPLList.

3.5. Cavity before \( B_0 \)

As stated earlier, we start our traversals of GhoshList and MergeList at an arbitrary occurrence of \( +B_0 \). However, often the first cavity we come across is not actually the “first” cavity of the starting segment. That is, that cavity is not the first cavity which should be processed after processing all cavities in the previous section. \( +B_0 \) is used because it is either not part of a \( B \) cavity or is the first edge of a \( B \) cavity. Instead of moving to the next segment when all the cavities and non-cavity \( B \) edges have been processed, for the first segment we move to the next segment when we have reached the “final” cavity. That is, the cavity which should be processed directly prior to moving onto the next segment.

To determine the final cavity of the first segment, we process the cavities as per usual. If, during the traversal of MergeList, we start and finish processing a cavity which intersects the current segment without leaving MergeList, then the final cavity is the cavity which was processed directly prior to this. An example is shown in Fig. 11.

Here the \( B \) cavity involving \( B_2 \) and \( B_3 \) intersects the starting segment \( A_3 \rightarrow A_2 \) twice. However, \( +B_0 \) is located between these two intersections. Starting at \( +B_0 \), we pass \( A_6, B_1, A_7, A_0, B_2, A_1, A_2, B_3 \). At this point we have processed one intersection (CavA) of the \( B \) cavity. We continue our traversal past \( -A_2, B_4, \) and \( A_2 \). We have now reached the end of our segment, but CavB has not yet been processed, so we move onto MergeList. We pass \( B_0, B_1, B_2, \) and \( B_3 \), without leaving MergeList. Now we have processed CavB, but it was done entirely within MergeList, so our final cavity of the starting segment is CavA.

So the initial partial traversal of segment \( A_3 \rightarrow A_2 \) adds edges \( (B_0, A_6, B_1, A_7, A_0, B_2, A_1, A_2, B_3, -A_2, B_4, A_2) \) to NFPLList, processing CavA. The traversal of segment \( A_2 \rightarrow A_3 \) adds edges \( (-B_4, -A_2, -B_3, A_2, -B_2, A_3) \) to NFPLList. The final partial traversal of segment \( A_3 \rightarrow A_2 \) processes CavB, and adds edges \( (B_2, A_4, B_3, B_4, A_5) \) to NFPLList.
End If

\[
p = p + \text{Dir}
\]

If GhoshList[p] = B0 And NextB = 0 And CurrSeg = OrigSeg And Seg[CurrSeg].CavitiesLeft = 0 Then Exit Algorithm

\}

\textbf{TraverseMergeList}\{

\textit{OrigPos} = \text{Pos} = \text{Position of } p \text{ in MergeList}
\textit{OrigDir} = \text{TotalDir} = \text{MergeDir} = \text{BMult}\ast \text{Dir}

\textbf{Loop2}\{

\textit{Pos} = \text{Pos} + \text{MergeDir}

If MergeList[\textit{Pos}].\text{PolygonType} = B Then

\hspace{1em} \text{If MergeList}[\textit{Pos}].\text{PolygonIndex} = \text{NextB} \text{ Then}

\hspace{2em} \text{NFPList} = \text{NFPList} + \text{MergeList}[\textit{Pos}]\ast \text{BMult}

\hspace{2em} \text{NextB} = \text{NextB} + \text{BMult}

\hspace{2em} \text{If MergeList}[\textit{Pos}].\text{IsTurningPoint} = \text{True} \text{ Then}

\hspace{3em} \text{MergeDir} = \text{MergeDir}\ast - 1

\hspace{2em} \text{End If}

\hspace{2em} \text{If CurrSeg = OrigSeg And Condition1() = True Then}

\hspace{3em} \text{CavityBeforeB0} = \text{True}

\hspace{3em} \text{Exit Loop2}

\hspace{2em} \text{Else If MergeDir = OrigDir And}

\hspace{3em} \text{Condition2() = True Then}

\hspace{4em} \text{Exit Loop2}

\hspace{2em} \text{End If}

\hspace{2em} \text{End If}

\hspace{2em} \text{Else If Pos = OrigPos Then}

\hspace{3em} \text{TotalDir} = \text{TotalDir} + \text{MergeDir}

\hspace{3em} \text{If TotalDir} = 0 \text{ Then}

\hspace{4em} \text{Dir} = \text{Dir}\ast - 1

\hspace{4em} \text{NFPList} = \text{NFPList} + \text{MergeList}[\textit{Pos}]\ast \text{Dir}

\hspace{4em} \text{Return To Loop1}

\hspace{2em} \text{End If}

\hspace{2em} \text{End If}

\}

\textbf{End If}

\textit{CavityBeforeB0} = \text{True}

\text{Remove from NFPList all B edges just added in TraverseMergeList up to the penultimate cavity added}

\}

If CavityBeforeB0 = True Then

\text{NFPList} = \text{NFPList} + \text{MergeList}[(\text{OrigPos})\ast - 1\ast \text{OrigDir}]

\text{BMult} = \text{BMult}\ast - 1

\text{Return To Loop1}

\}
**Theorem 4.3.** The outer envelope of the polygon described by NFPList[A, −B] can be constructed without negative edges of NFPList[A, −B].

**Proof.** The outer envelope of the Minkowski sum of A and −B is equivalent to the outer envelope of polygon described by NFPList[A, −B]. Eq. (1) states that the Minkowski sum of A and −B is the result of vector additions of all combinations of points from A and points −B. Consequently, the outer envelope of NFPList[A, −B] can then be constructed using only positive edges of A and −B. Therefore, the outer envelope of the polygon described by NFPList[A, −B] can be constructed without negative edges of NFPList[A, −B]. □

Using Theorems 4.1 and 4.2 we can reduce the number of calculations required to find the outer envelope because we know that edges that lie on the convex hull of their respective polygons occur at least once on the outer envelope. However, because construction of NFPList can give rise to multiple positive and negative copies of these edges, we need to establish some rules to determine which of these occurrences are actually members of the outer envelope.

An edge E is a member of its respective polygons convex hull, conv(P), then it will occur on NFPList[A, −B] if it is sliding along a convex vertex of the other polygon Q. For this to occur, the following conditions must hold:

**Condition 4.1.** E is non-negative.

**Condition 4.2.** E is a member of conv(P).

**Condition 4.3.** If the Q edge that precedes E goes from point s to point t, then t must be a point on conv(Q).

**Condition 4.4.** If the Q edge following E goes from point u to point v, then u must be a point on conv(Q).

**Condition 4.5.** The angle of E must be between the angles of the Q edges that precede and follow E. If those Q edges are not a member of conv(Q), then we use the convex edge that would replace them.

We can now divide the problem of finding the outer envelope of NFPList into sub-problems of finding the outer envelope of each set of edges between the edges already identified to lie on the outer envelope. We can further reduce the number of candidate edges for the outer envelope by removing negative edges using Theorem 4.3.

An example of the reduction in calculation is shown below in Fig. 12.

NFPList[A, B] is the equivalent of OE[B0, B1, A4, A5, B2, B3, A0, A1, A2, B4, −A2, B5, A2, A3], where OE[x] represents the outer envelope of x. However, using Theorems 4.1 and 4.2 we can identify that edges A4, A5, A2, A2, and A3 are on outer envelope. Using Theorem 4.3 we can discard edge −A2. So the calculation of the edge list of NFPList[A, B] can be simplified to (OE[B0, B1], A4, A5, OE[B2, B3, A0, A1], A2, OE[B4, B5], A2, A3).

5. Adjusting for parallel edges

The method given in this paper creates NFPList[A, B], such that when the reference point of B is touching the perimeter of NFPList[A, B], polygons A and B touch. In two-dimensional packing problems, often it is
Fig. 13. Data set.

Table 1
Polygon properties

<table>
<thead>
<tr>
<th>Polygon</th>
<th>Edges</th>
<th>Cavities</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>65</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>43</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>21</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>58</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>128</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>64</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>141</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 2
NFP calculation results

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Edges</th>
<th>Non-negative edges</th>
<th>Time 1 (s)</th>
<th>Time 2 (s)</th>
<th>% Impr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>138</td>
<td>134</td>
<td>0.016</td>
<td>0.007</td>
<td>57.50</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>132</td>
<td>120</td>
<td>0.017</td>
<td>0.007</td>
<td>60.47</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>162</td>
<td>124</td>
<td>0.022</td>
<td>0.010</td>
<td>55.56</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>265</td>
<td>189</td>
<td>0.032</td>
<td>0.012</td>
<td>61.25</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>525</td>
<td>359</td>
<td>0.068</td>
<td>0.033</td>
<td>51.18</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>421</td>
<td>275</td>
<td>0.036</td>
<td>0.018</td>
<td>49.45</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>201</td>
<td>165</td>
<td>0.025</td>
<td>0.014</td>
<td>41.94</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>506</td>
<td>356</td>
<td>0.070</td>
<td>0.032</td>
<td>53.71</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>260</td>
<td>173</td>
<td>0.020</td>
<td>0.008</td>
<td>61.22</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>104</td>
<td>84</td>
<td>0.006</td>
<td>0.003</td>
<td>57.14</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>143</td>
<td>122</td>
<td>0.014</td>
<td>0.005</td>
<td>66.67</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>249</td>
<td>210</td>
<td>0.037</td>
<td>0.014</td>
<td>61.29</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>253</td>
<td>180</td>
<td>0.020</td>
<td>0.010</td>
<td>50.00</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>319</td>
<td>213</td>
<td>0.024</td>
<td>0.012</td>
<td>51.67</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>212</td>
<td>198</td>
<td>0.043</td>
<td>0.014</td>
<td>66.67</td>
</tr>
</tbody>
</table>
References