COLLOQUY

Cooking Up the Optimal Baking Algorithm

Tony Burand, Michael Tetzlaff, and Jacob Smith

Appendix: Source Code

Heating Model

```
(* Function to generate a table of surface areas for the data points, \
where an edge has a value of 1, a corner has a value greater than or \
equal to one, and an interior data point or a point outside the \
simulation area has a value of 0.  More or less brute force. *)
CalculateBorder[n_, subdiv_] :=
  Module[{\[CapitalTheta]1, \[CapitalTheta]2, X1, X2, Y1, Y2, Slope,
    HalfSubdiv, Result, iMin, iMax, jMin, jMax, temp, i, j},
   Result = Table[Table[0, {j, 1, subdiv}], {i, 1, subdiv}];
   For[k = 1, k <= n, k = k + 1,
    \[CapitalTheta]1 = 2 \[Pi]/n*(k - 1);
    \[CapitalTheta]2 = 2 \[Pi]/n*k;
    X1 = Cos[\[CapitalTheta]1];
    Y1 = Sin[\[CapitalTheta]1];
    X2 = Cos[\[CapitalTheta]2];
    Y2 = Sin[\[CapitalTheta]2];
    HalfSubdiv = Floor[subdiv/2];
    iMin = HalfSubdiv + Round[HalfSubdiv*X1] + 1;
    iMax = HalfSubdiv + Round[HalfSubdiv*X2] + 1;
    jMin = HalfSubdiv + Round[HalfSubdiv*Y1] + 1;
    jMax = HalfSubdiv + Round[HalfSubdiv*Y2] + 1;

    Result[[iMin]][[jMin]] = N[2 - Sin[((n - 2)/n) \[Pi] - \[Pi]/2]];

    If[X2 == X1,
     If[jMin > jMax,
      For[j = jMax + 1, j < jMin, j = j + 1,
       Result[[iMin]][[j]] = 1;
       ],
      (* else *)
      For[j = jMin + 1, j < jMax, j = j + 1,
       Result[[iMin]][[j]] = 1;
       ]
      ],
     (* else *)
     Slope = (Y2 - Y1)/(X2 - X1);
     If[Abs[Slope] > 1,
      If[jMin > jMax,
       For[j = jMax + 1, j < jMin, j = j + 1,
        i =
         HalfSubdiv +
          Round[HalfSubdiv*((X1 - X2)*(j - jMax)/(jMin - jMax) +
              X2)] + 1;
        Result[[i]][[j]] = 1;
        ],
       (* else *)
       For[j = jMin + 1, j < jMax, j = j + 1,
        i =
         HalfSubdiv +
          Round[HalfSubdiv*((X2 - X1)*(j - jMin)/(jMax - jMin) +
              X1)] + 1;
        Result[[i]][[j]] = 1;
        ]
       ],
      (* else *)
      If[iMin > iMax,
       For[i = iMax + 1, i < iMin, i = i + 1,
        j =
         HalfSubdiv +
          Round[HalfSubdiv*((Y1 - Y2)*(i - iMax)/(iMin - iMax) +
              Y2)] + 1;
        Result[[i]][[j]] = 1;
        ],
       (* else *)
       For[i = iMin + 1, i < iMax, i = i + 1,
        j =
         HalfSubdiv +
          Round[HalfSubdiv*((Y2 - Y1)*(i - iMin)/(iMax - iMin) +
              Y1)] + 1;
        Result[[i]][[j]] = 1;
        ]
       ]
      ]
     ]
    ];
   Result
```

```
   ];

(* Function to generate a mask table where the corner data points \
have an value of 1 and all other data point have a a value of 0 *)
CalculateCornerMask[n_, subdiv_] :=
  Module[{\[CapitalTheta]1, \[CapitalTheta]2, X1, X2, Y1, Y2, Slope,
    HalfSubdiv, Result, iMin, iMax, jMin, jMax, temp, i, j},
   Result = Table[Table[0, {j, 1, subdiv}], {i, 1, subdiv}];
   For[k = 1, k <= n, k = k + 1,
    \[CapitalTheta]1 = 2 \[Pi]/n*(k - 1);
    \[CapitalTheta]2 = 2 \[Pi]/n*k;
    X1 = Cos[\[CapitalTheta]1];
    Y1 = Sin[\[CapitalTheta]1];
    X2 = Cos[\[CapitalTheta]2];
    Y2 = Sin[\[CapitalTheta]2];
    HalfSubdiv = Floor[subdiv/2];
    iMin = HalfSubdiv + Round[HalfSubdiv*X1] + 1;
    iMax = HalfSubdiv + Round[HalfSubdiv*X2] + 1;
    jMin = HalfSubdiv + Round[HalfSubdiv*Y1] + 1;
    jMax = HalfSubdiv + Round[HalfSubdiv*Y2] + 1;

    Result[[iMin]][[jMin]] = 1;
    ];
   Result
   ];

(* Function to generate a mask table where the non-corner edge data \
points have an value of 1 and all other data point have a a value of \
0 *)
CalculateEdgeMask[n_, subdiv_] :=
  Module[{\[CapitalTheta]1, \[CapitalTheta]2, X1, X2, Y1, Y2, Slope,
    HalfSubdiv, Result, iMin, iMax, jMin, jMax, temp, i, j},
   Result = Table[Table[0, {j, 1, subdiv}], {i, 1, subdiv}];
   For[k = 1, k <= n, k = k + 1,
    \[CapitalTheta]1 = 2 \[Pi]/n*(k - 1);
    \[CapitalTheta]2 = 2 \[Pi]/n*k;
    X1 = Cos[\[CapitalTheta]1];
    Y1 = Sin[\[CapitalTheta]1];
    X2 = Cos[\[CapitalTheta]2];
    Y2 = Sin[\[CapitalTheta]2];
    HalfSubdiv = Floor[subdiv/2];
    iMin = HalfSubdiv + Round[HalfSubdiv*X1] + 1;
    iMax = HalfSubdiv + Round[HalfSubdiv*X2] + 1;
    jMin = HalfSubdiv + Round[HalfSubdiv*Y1] + 1;
    jMax = HalfSubdiv + Round[HalfSubdiv*Y2] + 1;

    If[X2 == X1,
       If[jMin > jMax,
        For[j = jMax + 1, j < jMin, j = j + 1,
         Result[[iMin]][[j]] = 1;
         ],
        (* else *)
        For[j = jMin + 1, j < jMax, j = j + 1,
         Result[[iMin]][[j]] = 1;
         ]
        ],
       (* else *)
       Slope = (Y2 - Y1)/(X2 - X1);
       If[Abs[Slope] > 1,
        If[jMin > jMax,
         For[j = jMax + 1, j < jMin, j = j + 1,

          i = HalfSubdiv +
            Round[HalfSubdiv*((X1 - X2)*(j - jMax)/(jMin - jMax) +
                X2)] + 1;
          Result[[i]][[j]] = 1;
          ],
         (* else *)
         For[j = jMin + 1, j < jMax, j = j + 1,

          i = HalfSubdiv +
            Round[HalfSubdiv*((X2 - X1)*(j - jMin)/(jMax - jMin) +
                X1)] + 1;
          Result[[i]][[j]] = 1;
          ]
         ],
        (* else *)
        If[iMin > iMax,
         For[i = iMax + 1, i < iMin, i = i + 1,

          j = HalfSubdiv +
```

```
            Round[HalfSubdiv*((Y1 - Y2)*(i - iMax)/(iMin - iMax) +
                Y2)] + 1;
           Result[[i]][[j]] = 1;
           ],
          (* else *)
          For[i = iMin + 1, i < iMax, i = i + 1,

           j = HalfSubdiv +
            Round[HalfSubdiv*((Y2 - Y1)*(i - iMin)/(iMax - iMin) +
                Y1)] + 1;
           Result[[i]][[j]] = 1;
           ]
          ]
         ]
        ]
       Result[[iMin]][[jMin]] = 0;
     ];
    Result
    ];

(* Function to generate a mask table where the corner data points \
have an value of 1 and all other data point have a a value of 0 *)
CalculateStencil[n_, subdiv_] :=
  Module[{\[CapitalTheta]1, \[CapitalTheta]2, X1, X2, Y1, Y2, Slope,
    HalfSubdiv, Result, iMin, iMax, jMin, jMax, temp, i, j},
   Result = Table[Table[0, {j, 1, subdiv}], {i, 1, subdiv}];
   For[k = 1, k <= n, k = k + 1,
    \[CapitalTheta]1 = 2 \[Pi]/n*(k - 1);
    \[CapitalTheta]2 = 2 \[Pi]/n*k;
    X1 = Cos[\[CapitalTheta]1];
    Y1 = Sin[\[CapitalTheta]1];
    X2 = Cos[\[CapitalTheta]2];
    Y2 = Sin[\[CapitalTheta]2];
    HalfSubdiv = Floor[subdiv/2];
    iMin = HalfSubdiv + Round[HalfSubdiv*X1] + 1;
    iMax = HalfSubdiv + Round[HalfSubdiv*X2] + 1;
    jMin = HalfSubdiv + Round[HalfSubdiv*Y1] + 1;
    jMax = HalfSubdiv + Round[HalfSubdiv*Y2] + 1;

    If[X2 == X1,
     If[jMin > jMax,
      For[j = jMax, j <= jMin, j = j + 1,
       If[iMin > HalfSubdiv,
        For[i = iMin, i >= HalfSubdiv, i = i - 1,
         Result[[i]][[j]] = 1;
         ],
        (* else iMin <= HalfSubdiv *)
        For[i = iMin, i <= HalfSubdiv, i = i + 1,
         Result[[i]][[j]] = 1;
         ]
        ]
       ]
      (* else jMin <= jMax *)
      For[j = jMin, j <= jMax, j = j + 1,
       If[iMin > HalfSubdiv,
        For[i = iMin, i >= HalfSubdiv, i = i - 1,
         Result[[i]][[j]] = 1;
         ],
        (* else iMin <= HalfSubdiv *)
        For[i = iMin, i <= HalfSubdiv, i = i + 1,
         Result[[i]][[j]] = 1;
         ]
        ]
       ],
     (* else X2 != X1 *)
     Slope = (Y2 - Y1)/(X2 - X1);
     If[Abs[Slope] > 1,
      If[jMin > jMax,
       For[j = jMax, j <= jMin, j = j + 1,
        i =
         HalfSubdiv +
          Round[HalfSubdiv*((X1 - X2)*(j - jMax)/(jMin - jMax) +
              X2)] + 1;
        If[i > HalfSubdiv,
         For[i = i, i >= HalfSubdiv, i = i - 1,
          Result[[i]][[j]] = 1;
          ],
         (* else i <= HalfSubdiv *)
         For[i = i, i <= HalfSubdiv, i = i + 1,
          Result[[i]][[j]] = 1;
```

```
          ]
         ]
        ],
       (* else jMin <= jMax *)
       For[j = jMin, j <= jMax, j = j + 1,
        i =
         HalfSubdiv +
          Round[HalfSubdiv*((X2 - X1)*(j - jMin)/(jMax - jMin) +
              X1)] + 1;
        If[i > HalfSubdiv,
         For[i = i, i >= HalfSubdiv, i = i - 1,
          Result[[i]][[j]] = 1;
          ],
         (* else i <= HalfSubdiv *)
         For[i = i, i <= HalfSubdiv, i = i + 1,
          Result[[i]][[j]] = 1;
          ]
         ]
        ]
       ],
      (* else Abs[Slope]<=1 *)
      If[iMin > iMax,
       For[i = iMax, i <= iMin, i = i + 1,
        j =
         HalfSubdiv +
          Round[HalfSubdiv*((Y1 - Y2)*(i - iMax)/(iMin - iMax) +
              Y2)] + 1;
        If[j > HalfSubdiv,
         For[j = j, j >= HalfSubdiv, j = j - 1,
          Result[[i]][[j]] = 1;
          ],
         (* else j <= HalfSubdiv *)
         For[j = j, j <= HalfSubdiv, j = j + 1,
          Result[[i]][[j]] = 1;
          ]
         ]
        ],
       (* else iMin <= iMax *)
       For[i = iMin, i <= iMax, i = i + 1,
        j =
         HalfSubdiv +
          Round[HalfSubdiv*((Y2 - Y1)*(i - iMin)/(iMax - iMin) +
              Y1)] + 1;
        If[j > HalfSubdiv,
         For[j = j, j >= HalfSubdiv, j = j - 1,
          Result[[i]][[j]] = 1;
          ],
         (* else j <= HalfSubdiv *)
         For[j = j, j <= HalfSubdiv, j = j + 1,
          Result[[i]][[j]] = 1;
          ]
         ]
        ]
       ]
      ]
     ];
   Result
   ];

(* Function to calculate the change in temperature using Newton's Law \
of Heating and Cooling *)
HeatTransfer[\[Beta]_, T0_, dx_, dy_, dz_, AreaGrid_,
   Temperatures_] := \[Beta]/(dx dy dz)*
     AreaGrid*(T0 - Temperatures) /. Indeterminate -> 0.0 /.
   ComplexInfinity -> 0.0;

(* Function to calculate the change in temperature due to diffusion, \
using a FTCS method *)
Diffusion[\[Alpha]_, dx_, dy_, dz_, Stencil_, Temperatures_] :=
  Table[Table[Table[
          \[Alpha]*(
       If[
          i > 1 && i < Dimensions[Temperatures][[2]] &&
           Stencil[[i + 1]][[j]] == 1 && Stencil[[i - 1]][[j]] == 1,

          Temperatures[[k]][[i + 1]][[j]] -
           2 Temperatures[[k]][[i]][[j]] +
           Temperatures[[k]][[i - 1]][[j]], 0]/dx^2
        +
        If[j > 1 && j < Dimensions[Temperatures][[3]] &&
```

```
            Stencil[[i]][[j + 1]] == 1 && Stencil[[i]][[j - 1]] == 1,

           Temperatures[[k]][[i]][[j + 1]] -
            2 Temperatures[[k]][[i]][[j]] +
            Temperatures[[k]][[i]][[j - 1]], 0]/dy^2
         + If[k > 1 && k < Dimensions[Temperatures][[1]],

           Temperatures[[k + 1]][[i]][[j]] -
            2 Temperatures[[k]][[i]][[j]] +
            Temperatures[[k - 1]][[i]][[j]], 0]/dz^2
        ),
      {j, 1, Dimensions[Temperatures][[3]]}],
     {i, 1, Dimensions[Temperatures][[2]]}],
    {k, 1, Dimensions[Temperatures][[1]]}];

(* The simulation method *)
(* n: number of sides *)
(* \[Alpha]: Diffusion parameter *)
(* \[Beta]: Pan heating parameter *)
(* \[Gamma]: Air heating parameter *)
(* w: Pan width *)
(* h: Brownie height *)
(* T: Time of simulation *)
(* nx: Number of subdivisions along each horizontal axis *)
(* nz: Number of subdivisions along the vertical axis *)
(* nt: Number of time steps *)
(* Temp: Oven/pan temperature *)
Simulation[n_, \[Alpha]_, \[Beta]_, \[Gamma]_, w_, h_, T_, nx_, nz_,
    nt_, Temp_] := Module[
    {A, P, dw, dh, dt, Stencil, Border, AreaGrid3D, AreaGrid3DTop,
     TempSums, Temperatures, i},
    (* Calculate size of spatial and time steps *)
    dw = w/nx;
    dh = h/nz;
    dt = T/nt;

    (* Calculate the correct area and perimeter *)
    A = 1/4 n w^2 Sin[(1 - (n - 2)/n) \[Pi]/
        2] Cos[(1 - (n - 2)/n) \[Pi]/2];
    P = n w Sin[(1 - (n - 2)/n) \[Pi]/2];

    (* Generate stencil and border tables *)
    Stencil = CalculateStencil[n, nx];
    Border = CalculateBorder[n, nx];

    (* Use 2D stencil and border to Calculate 3D surface area tables \
for the pan and the air *)
    AreaGrid3DTop =
     Join[Table[
       Table[Table[0.0, {y, 1, nx}], {x, 1, nx}], {z, 1,
        nz - 1}], {Stencil}];
    AreaGrid3DTop = AreaGrid3DTop*A/Total[Total[Stencil]];
    AreaGrid3D = Table[Border, {i, 1, nz}];
    AreaGrid3D = AreaGrid3D*P*h/(Total[Total[Border]]*nz);
    AreaGrid3D[[1]] = AreaGrid3D[[1]] + AreaGrid3DTop[[nz]];

    (* Setup temperature table *)
    Temperatures =
     Table[Table[
       Table[22.4*Stencil[[i]][[j]], {j, 1, nx}], {i, 1, nx}], {k, 1,
       nz}];
    TempSums =
     Table[Table[
       Table[22.4*Stencil[[i]][[j]], {j, 1, nx}], {i, 1, nx}], {k, 1,
       nz}];

    (* Advance the simulation through all the time steps *)
    For[i = 1, i <= nt, i = i + 1,
     Temperatures = Temperatures +
       (HeatTransfer[\[Beta], Temp, dw, dw, dh, AreaGrid3D,
           Temperatures] +
         HeatTransfer[\[Gamma], Temp, dw, dw, dh, AreaGrid3DTop,
          Temperatures] +
         Diffusion[\[Alpha], dw, dw, dh, Stencil, Temperatures])*dt;
     TempSums = TempSums + Temperatures;
     ];

    (* Return the final temperatures and the average temerature for \
every data point *)
    {Temperatures, TempSums/(nt + 1)}
    ];
```

Space Optimization, Fitted Heat Equation, and Weighted Evaluation

```
(*)
L = 30; W = 60; a = 81; percent = 1.08;(* constants used in pan area \
models *)
side[n_, A_] := Sqrt[(
 4 A Tan[\[Pi]/n])/n]; \[Theta][n_] := (n - 2)/n \[Pi];
apothem[n_, A_] := Sqrt[A/(n Tan[\[Pi]/n])];
radius[n_, A_] := Sqrt[(2 A)/(n Sin[(2 \[Pi])/n])];
perimeter[n_, A_] := n*side[n, A]; height[n_, A_] := \!\(\*
TagBox[GridBox[{
{"\[Piecewise]", GridBox[{
{
RowBox[{"2",
RowBox[{"apothem", "[",
RowBox[{"n", ",", "A"}], "]"}]}],
RowBox[{
RowBox[{"Mod", "[",
RowBox[{"n", ",", "2"}], "]"}], "==", "0"}]},
{
RowBox[{
RowBox[{"apothem", "[",
RowBox[{"n", ",", "A"}], "]"}], "+",
RowBox[{"radius", "[",
RowBox[{"n", ",", "A"}], "]"}]}],
RowBox[{
RowBox[{"Mod", "[",
RowBox[{"n", ",", "2"}], "]"}], "==", "1"}]}
},
AllowedDimensions->{2, Automatic},
Editable->True,
GridBoxAlignment->{
         "Columns" -> {{Left}}, "ColumnsIndexed" -> {},
          "Rows" -> {{Baseline}}, "RowsIndexed" -> {}},
GridBoxItemSize->{
         "Columns" -> {{Automatic}}, "ColumnsIndexed" -> {},
          "Rows" -> {{1.}}, "RowsIndexed" -> {}},
GridBoxSpacings->{"Columns" -> {
Offset[0.27999999999999997`], {
Offset[0.84]},
Offset[0.27999999999999997`]}, "ColumnsIndexed" -> {}, "Rows" -> {
Offset[0.2], {
Offset[0.4]},
Offset[0.2]}, "RowsIndexed" -> {}},
Selectable->True]}
},
GridBoxAlignment->{
      "Columns" -> {{Left}}, "ColumnsIndexed" -> {},
       "Rows" -> {{Baseline}}, "RowsIndexed" -> {}},
GridBoxItemSize->{
      "Columns" -> {{Automatic}}, "ColumnsIndexed" -> {},
       "Rows" -> {{1.}}, "RowsIndexed" -> {}},
GridBoxSpacings->{"Columns" -> {
Offset[0.27999999999999997`], {
Offset[0.35]},
Offset[0.27999999999999997`]}, "ColumnsIndexed" -> {}, "Rows" -> {
Offset[0.2], {
Offset[0.4]},
Offset[0.2]}, "RowsIndexed" -> {}}],
"Piecewise",
DeleteWithContents->True,
Editable->False,
SelectWithContents->True,
Selectable->False]\);
MagicNumberApothem[n_, A_] := Sqrt[(4 Cot[\[Pi]/n])/n]*Sqrt[A] // N;
MagicNumberRadius[n_, A_] :=
 Sqrt[2/(n Sin[(2 \[Pi])/n])]*Sqrt[A] //
  N; (* This section is dedicated to finding the different dimensions \
of a regular polygon including sides,apothem,and radius. We are also \
able to find the magic numbers for apothem and radius depending on \
the shape and area of a certain polygon. *)
numberofcols[s_, A_, l_, w_, p_] :=
 Floor[1/(2 apothem[s,
    p A])] (* calculate number of columns for even non-multiple of 4 \
n-gons *)
evenrow[s_, A_, l_, w_, p_] :=
```

```
 If[l - 2 numberofcols[s, A, l, w, p] apothem[p s, A] -
    apothem[p s, A] >= 0, True,
  False] (* checks to see if it is possible to fit another n-gon in \
every other row *)
numberofrows[s_, A_, l_, w_, p_] :=
 Do[i = -1; x = w;
  While[If[EvenQ[i], x, x - side[s, p A]/2] >= 0,
   x = x - If[OddQ[i], 2*side[s, p A], side[s, p A]]; i++];
  Return[i], {1}] (* finds number of rows for even non-multiple of 4 \
n-gons in tessellating pattern *)
numberofhexs[s_, A_, l_, w_, p_] :=
 If[evenrow[s, A, l, w, p] == True,
  numberofrows[s, A, l, w, p]*numberofcols[s, A, l, w, p],
  numberofrows[s, A, l, w, p]*numberofcols[s, A, l, w, p] -
   Floor[numberofrows[s, A, l, w, p]/
    2]] (* calculates the number of hexagons for a particular w and l \
oven *)
latticearrange1[s_, A_, l_, w_] :=
 Floor[w/If[IntegerQ[s/4], height[s, A] // N, (2 radius[s, A])]]*
  Floor[l/height[s, A]] (* first lattice arrangement of n-gons *)
latticearrange2[s_, A_, l_, w_] :=
 Floor[w/height[s, A]]*
  Floor[l/If[IntegerQ[s/4],
     height[s, A] //
      N, (2 radius[s, A])]] (* second lattice arrangement of n-gons *)


hexadaptarrange1[s_, A_, l_, w_, p_] :=
 numberofhexs[s, A, l, w,
  p] (* first tessellating arrangement of n-gons *)
hexadaptarrange2[s_, A_, l_, w_, p_] :=
 numberofhexs[s, A, w, l,
  p] (* second tessellating arrangement of n-gons *)
(* calculates number of n-gons that can fit in a particular w and l \
rectangular oven *)

numberofpolys[s_, A_, l_, w_, p_] :=
 If[s != 6 && s != 4,
  If[latticearrange1[s, A, l, w] >= latticearrange2[s, A, l, w],
   latticearrange1[s, A, l, w], latticearrange2[s, A, l, w]],

  If[s != 4,
   If[
    If[hexadaptarrange1[s, A, l, w, p] >=
       hexadaptarrange2[s, A, l, w, p],
      hexadaptarrange1[s, A, l, w, p],
      hexadaptarrange2[s, A, l, w, p]] >=
     If[latticearrange1[s, A, l, w] >= latticearrange2[s, A, l, w],
      latticearrange1[s, A, l, w], latticearrange2[s, A, l, w]],

    If[hexadaptarrange1[s, A, l, w, p] >=
      hexadaptarrange2[s, A, l, w, p],
     hexadaptarrange1[s, A, l, w, p], hexadaptarrange2[s, A, l, w, p]],

    If[latticearrange1[s, A, l, w] >= latticearrange2[s, A, l, w],
     latticearrange1[s, A, l, w], latticearrange2[s, A, l, w]]],

   If[latticearrange1[s, p A, l, w] >= latticearrange2[s, p A, l, w],
    latticearrange1[s, p A, l, w],
    latticearrange2[s, p A, l,
     w]]]] (* calculates number of n-gons that can fit in a \
particular w and l rectangular oven *)
Eff[s_, A_, w_, l_, p_] := (numberofpolys[s, A, l, w, p]*A)/(
 l*w) (* calculates the oven space efficiency for one shelf for \
particular n-gon and rectangular oven *)
(*Effheat[s_,A_]:=.0114944*s+.340489/Sqrt[A]+.801163*)
Effheat[s_, A_] :=
 Min[1, .055657869822367416*
    ArcTan[.2259800337169932*s -
      1.6457785931252638] + .10716283573624863/(
   Sqrt[A] - 2.462771976211951) + .9045771916045253]

ranknumber[shaperank_, heatrank_, w_] := shaperank w + (1 - w) heatrank

Effgenerate2[A_, l_, w_, p_, weight_, polylist_] :=
 Do[polylistlocal = Sort[polylist];
  If[Length[polylistlocal] < 6, Break[]];

  rank = {}; spatialeff = 0;
  For[gg = 1, gg <= Length[polylistlocal], gg++,
   rank = Append[
```

```
      rank, {polylistlocal[[gg]],
       Eff[polylistlocal[[gg]], A, l, w, p] // N}]];

  spatialeff = RankedMax[Flatten[rank], Length[rank] + 1];
  For[o = 1, o <= Length[rank], o++, rank[[o]][[2]] /= spatialeff];

  SortBy[rank, Last];
  For[qq = 1; heateff = {}, qq <= Length[rank], qq++,
   heateff = Append[heateff, Effheat[rank[[qq]][[1]], A] // N]];

  For[ss = 1; shape = {}; sideorder = {}, ss <= Length[polylistlocal],
    ss++, sideorder = Append[sideorder, rank[[ss]][[1]]];
   shape = Append[shape,
     ranknumber[rank[[ss]][[2]], heateff[[ss]], weight]]];

  Return[sideorder[[Part[Flatten[Position[shape, Max[shape]]], 1]]]
  , {1}]

PrintResults[A_, l_, w_, p_, weight_, polylist_] :=
 Print["The most efficient pan based on the parameters is a ",
  Effgenerate1[A, l, w, p, weight, polylist], "-sided polygon"]
```