

# COLLOQUY

## Cooking Up the Optimal Baking Algorithm

---

*Tony Burand, Michael Tetzlaff, and Jacob Smith*

# Appendix: Source Code

## Heating Model

```
1  (* Function to generate a table of surface areas for the data points, \
2  where an edge has a value of 1, a corner has a value greater than or \
3  equal to one, and an interior data point or a point outside the \
4  simulation area has a value of 0. More or less brute force. *)
5  CalculateBorder[n_, subdiv_] :=
6  Module[{\[CapitalTheta]1, \[CapitalTheta]2, X1, X2, Y1, Y2, Slope,
7  HalfSubdiv, Result, iMin, iMax, jMin, jMax, temp, i, j},
8  Result = Table[Table[0, {j, 1, subdiv}], {i, 1, subdiv}];
9  For[k = 1, k <= n, k = k + 1,
10 \[CapitalTheta]1 = 2 \[Pi]/n*(k - 1);
11 \[CapitalTheta]2 = 2 \[Pi]/n*k;
12 X1 = Cos[\[CapitalTheta]1];
13 Y1 = Sin[\[CapitalTheta]1];
14 X2 = Cos[\[CapitalTheta]2];
15 Y2 = Sin[\[CapitalTheta]2];
16 HalfSubdiv = Floor[subdiv/2];
17 iMin = HalfSubdiv + Round[HalfSubdiv*X1] + 1;
18 iMax = HalfSubdiv + Round[HalfSubdiv*X2] + 1;
19 jMin = HalfSubdiv + Round[HalfSubdiv*Y1] + 1;
20 jMax = HalfSubdiv + Round[HalfSubdiv*Y2] + 1;
21
22 Result[[iMin]][[jMin]] = N[2 - Sin[(n - 2)/n \[Pi] - \[Pi]/2]];
23
24 If[X2 == X1,
25 If[jMin > jMax,
26 For[j = jMax + 1, j < jMin, j = j + 1,
27 Result[[iMin]][[j]] = 1;
28 ],
29 (* else *)
30 For[j = jMin + 1, j < jMax, j = j + 1,
31 Result[[iMin]][[j]] = 1;
32 ],
33 ],
34 (* else *)
35 Slope = (Y2 - Y1)/(X2 - X1);
36 If[Abs[Slope] > 1,
37 If[jMin > jMax,
38 For[j = jMax + 1, j < jMin, j = j + 1,
39 i =
40 HalfSubdiv +
41 Round[HalfSubdiv*((X1 - X2)*(j - jMax)/(jMin - jMax) +
42 X2)] + 1;
43 Result[[i]][[j]] = 1;
44 ],
45 (* else *)
46 For[j = jMin + 1, j < jMax, j = j + 1,
47 i =
48 HalfSubdiv +
49 Round[HalfSubdiv*((X2 - X1)*(j - jMin)/(jMax - jMin) +
50 X1)] + 1;
51 Result[[i]][[j]] = 1;
52 ],
53 ],
54 (* else *)
55 If[iMin > iMax,
56 For[i = iMax + 1, i < iMin, i = i + 1,
57 j =
58 HalfSubdiv +
59 Round[HalfSubdiv*((Y1 - Y2)*(i - iMax)/(iMin - iMax) +
60 Y2)] + 1;
61 Result[[i]][[j]] = 1;
62 ],
63 (* else *)
64 For[i = iMin + 1, i < iMax, i = i + 1,
65 j =
66 HalfSubdiv +
67 Round[HalfSubdiv*((Y2 - Y1)*(i - iMin)/(iMax - iMin) +
68 Y1)] + 1;
69 Result[[i]][[j]] = 1;
70 ],
71 ],
72 ],
73 ];
74 Result
75
```

```

76     ];
77
78     (* Function to generate a mask table where the corner data points \
79     have an value of 1 and all other data point have a a value of 0 *)
80     CalculateCornerMask[n_, subdiv_] :=
81     Module[{\[CapitalTheta]1, \[CapitalTheta]2, X1, X2, Y1, Y2, Slope,
82             HalfSubdiv, Result, iMin, iMax, jMin, jMax, temp, i, j},
83     Result = Table[Table[0, {j, 1, subdiv}], {i, 1, subdiv}];
84     For[k = 1, k <= n, k = k + 1,
85     \[CapitalTheta]1 = 2 \[Pi]/n*(k - 1);
86     \[CapitalTheta]2 = 2 \[Pi]/n*k;
87     X1 = Cos\[CapitalTheta]1;
88     Y1 = Sin\[CapitalTheta]1;
89     X2 = Cos\[CapitalTheta]2;
90     Y2 = Sin\[CapitalTheta]2;
91     HalfSubdiv = Floor[subdiv/2];
92     iMin = HalfSubdiv + Round[HalfSubdiv*X1] + 1;
93     iMax = HalfSubdiv + Round[HalfSubdiv*X2] + 1;
94     jMin = HalfSubdiv + Round[HalfSubdiv*Y1] + 1;
95     jMax = HalfSubdiv + Round[HalfSubdiv*Y2] + 1;
96
97     Result[[iMin]][[jMin]] = 1;
98     ];
99     Result
100    ];
101
102     (* Function to generate a mask table where the non-corner edge data \
103     points have an value of 1 and all other data point have a a value of \
104     0 *)
105     CalculateEdgeMask[n_, subdiv_] :=
106     Module[{\[CapitalTheta]1, \[CapitalTheta]2, X1, X2, Y1, Y2, Slope,
107             HalfSubdiv, Result, iMin, iMax, jMin, jMax, temp, i, j},
108     Result = Table[Table[0, {j, 1, subdiv}], {i, 1, subdiv}];
109     For[k = 1, k <= n, k = k + 1,
110     \[CapitalTheta]1 = 2 \[Pi]/n*(k - 1);
111     \[CapitalTheta]2 = 2 \[Pi]/n*k;
112     X1 = Cos\[CapitalTheta]1;
113     Y1 = Sin\[CapitalTheta]1;
114     X2 = Cos\[CapitalTheta]2;
115     Y2 = Sin\[CapitalTheta]2;
116     HalfSubdiv = Floor[subdiv/2];
117     iMin = HalfSubdiv + Round[HalfSubdiv*X1] + 1;
118     iMax = HalfSubdiv + Round[HalfSubdiv*X2] + 1;
119     jMin = HalfSubdiv + Round[HalfSubdiv*Y1] + 1;
120     jMax = HalfSubdiv + Round[HalfSubdiv*Y2] + 1;
121
122     If[X2 == X1,
123     If[jMin > jMax,
124     For[j = jMax + 1, j < jMin, j = j + 1,
125     Result[[iMin]][[j]] = 1;
126     ],
127     (* else *)
128     For[j = jMin + 1, j < jMax, j = j + 1,
129     Result[[iMin]][[j]] = 1;
130     ]
131     ],
132     (* else *)
133     Slope = (Y2 - Y1)/(X2 - X1);
134     If[Abs[Slope] > 1,
135     If[jMin > jMax,
136     For[j = jMax + 1, j < jMin, j = j + 1,
137
138     i = HalfSubdiv +
139     Round[HalfSubdiv*((X1 - X2)*(j - jMax)/(jMin - jMax) +
140     X2)] + 1;
141     Result[[i]][[j]] = 1;
142     ],
143     (* else *)
144     For[j = jMin + 1, j < jMax, j = j + 1,
145
146     i = HalfSubdiv +
147     Round[HalfSubdiv*((X2 - X1)*(j - jMin)/(jMax - jMin) +
148     X1)] + 1;
149     Result[[i]][[j]] = 1;
150     ]
151     ],
152     (* else *)
153     If[iMin > iMax,
154     For[i = iMax + 1, i < iMin, i = i + 1,
155
156     j = HalfSubdiv +

```

```

157         Round[HalfSubdiv*((Y1 - Y2)*(i - iMax)/(iMin - iMax) +
158           Y2)] + 1;
159     Result[[i]][[j]] = 1;
160     ],
161     (* else *)
162     For[i = iMin + 1, i < iMax, i = i + 1,
163
164       j = HalfSubdiv +
165       Round[HalfSubdiv*((Y2 - Y1)*(i - iMin)/(iMax - iMin) +
166         Y1)] + 1;
167       Result[[i]][[j]] = 1;
168     ]
169   ]
170 ]
171 ]
172 Result[[iMin]][[jMin]] = 0;
173 ];
174 Result
175 ];
176
177 (* Function to generate a mask table where the corner data points \
178 have an value of 1 and all other data point have a value of 0 *)
179 CalculateStencil[n_, subdiv_] :=
180 Module[{[CapitalTheta]1, [CapitalTheta]2, X1, X2, Y1, Y2, Slope,
181   HalfSubdiv, Result, iMin, iMax, jMin, jMax, temp, i, j},
182   Result = Table[Table[0, {j, 1, subdiv}], {i, 1, subdiv}];
183   For[k = 1, k <= n, k = k + 1,
184     \[CapitalTheta]1 = 2 \[Pi]/n*(k - 1);
185     \[CapitalTheta]2 = 2 \[Pi]/n*k;
186     X1 = Cos\[CapitalTheta]1];
187     Y1 = Sin\[CapitalTheta]1];
188     X2 = Cos\[CapitalTheta]2];
189     Y2 = Sin\[CapitalTheta]2];
190     HalfSubdiv = Floor[subdiv/2];
191     iMin = HalfSubdiv + Round[HalfSubdiv*X1] + 1;
192     iMax = HalfSubdiv + Round[HalfSubdiv*X2] + 1;
193     jMin = HalfSubdiv + Round[HalfSubdiv*Y1] + 1;
194     jMax = HalfSubdiv + Round[HalfSubdiv*Y2] + 1;
195
196     If[X2 == X1,
197       If[jMin > jMax,
198         For[j = jMax, j <= jMin, j = j + 1,
199           If[iMin > HalfSubdiv,
200             For[i = iMin, i >= HalfSubdiv, i = i - 1,
201               Result[[i]][[j]] = 1;
202             ],
203             (* else iMin <= HalfSubdiv *)
204             For[i = iMin, i <= HalfSubdiv, i = i + 1,
205               Result[[i]][[j]] = 1;
206             ]
207           ]
208         ]
209       (* else jMin <= jMax *)
210       For[j = jMin, j <= jMax, j = j + 1,
211         If[iMin > HalfSubdiv,
212           For[i = iMin, i >= HalfSubdiv, i = i - 1,
213             Result[[i]][[j]] = 1;
214           ],
215           (* else iMin <= HalfSubdiv *)
216           For[i = iMin, i <= HalfSubdiv, i = i + 1,
217             Result[[i]][[j]] = 1;
218           ]
219         ]
220       ]
221     ],
222     (* else X2 != X1 *)
223     Slope = (Y2 - Y1)/(X2 - X1);
224     If[Abs[Slope] > 1,
225       If[jMin > jMax,
226         For[j = jMax, j <= jMin, j = j + 1,
227           i =
228           HalfSubdiv +
229           Round[HalfSubdiv*((X1 - X2)*(j - jMax)/(jMin - jMax) +
230             X2)] + 1;
231         If[i > HalfSubdiv,
232           For[i = i, i >= HalfSubdiv, i = i - 1,
233             Result[[i]][[j]] = 1;
234           ],
235           (* else i <= HalfSubdiv *)
236           For[i = i, i <= HalfSubdiv, i = i + 1,
237             Result[[i]][[j]] = 1;

```

```

238     ]
239   ]
240 ],
241 (* else jMin <= jMax *)
242 For[j = jMin, j <= jMax, j = j + 1,
243   i =
244     HalfSubdiv +
245     Round[HalfSubdiv*((X2 - X1)*(j - jMin)/(jMax - jMin) +
246       X1)] + 1;
247   If[i > HalfSubdiv,
248     For[i = i, i >= HalfSubdiv, i = i - 1,
249       Result[[i]][[j]] = 1;
250     ],
251     (* else i <= HalfSubdiv *)
252     For[i = i, i <= HalfSubdiv, i = i + 1,
253       Result[[i]][[j]] = 1;
254     ]
255   ]
256 ]
257 ],
258 (* else Abs[Slope]<=1 *)
259 If[iMin > iMax,
260   For[i = iMax, i <= iMin, i = i + 1,
261     j =
262       HalfSubdiv +
263       Round[HalfSubdiv*((Y1 - Y2)*(i - iMax)/(iMin - iMax) +
264         Y2)] + 1;
265     If[j > HalfSubdiv,
266       For[j = j, j >= HalfSubdiv, j = j - 1,
267         Result[[i]][[j]] = 1;
268       ],
269       (* else j <= HalfSubdiv *)
270       For[j = j, j <= HalfSubdiv, j = j + 1,
271         Result[[i]][[j]] = 1;
272       ]
273     ]
274   ],
275   (* else iMin <= iMax *)
276   For[i = iMin, i <= iMax, i = i + 1,
277     j =
278       HalfSubdiv +
279       Round[HalfSubdiv*((Y2 - Y1)*(i - iMin)/(iMax - iMin) +
280         Y1)] + 1;
281     If[j > HalfSubdiv,
282       For[j = j, j >= HalfSubdiv, j = j - 1,
283         Result[[i]][[j]] = 1;
284       ],
285       (* else j <= HalfSubdiv *)
286       For[j = j, j <= HalfSubdiv, j = j + 1,
287         Result[[i]][[j]] = 1;
288       ]
289     ]
290   ]
291 ]
292 ]
293 ];
294 ];
295 Result
296 ];
297
298 (* Function to calculate the change in temperature using Newton's Law \
299 of Heating and Cooling *)
300 HeatTransfer[\[Beta]_, T0_, dx_, dy_, dz_, AreaGrid_,
301   Temperatures_] := \[Beta]/(dx dy dz)*
302   AreaGrid*(T0 - Temperatures) /. Indeterminate -> 0.0 /.
303   ComplexInfinity -> 0.0;
304
305 (* Function to calculate the change in temperature due to diffusion, \
306 using a FTCS method *)
307 Diffusion[\[Alpha]_, dx_, dy_, dz_, Stencil_, Temperatures_] :=
308   Table[Table[Table[
309     \[Alpha]*(
310     If[
311       i > 1 && i < Dimensions[Temperatures][[2]] &&
312       Stencil[[i + 1]][[j]] == 1 && Stencil[[i - 1]][[j]] == 1,
313
314       Temperatures[[k]][[i + 1]][[j]] -
315       2 Temperatures[[k]][[i]][[j]] +
316       Temperatures[[k]][[i - 1]][[j]], 0]/dx^2
317     +
318     If[j > 1 && j < Dimensions[Temperatures][[3]] &&

```

```

319         Stencil[[i]][[j + 1]] == 1 && Stencil[[i]][[j - 1]] == 1,
320
321         Temperatures[[k]][[i]][[j + 1]] -
322         2 Temperatures[[k]][[i]][[j]] +
323         Temperatures[[k]][[i]][[j - 1]], 0]/dy^2
324     + If[k > 1 && k < Dimensions[Temperatures][[1]],
325
326         Temperatures[[k + 1]][[i]][[j]] -
327         2 Temperatures[[k]][[i]][[j]] +
328         Temperatures[[k - 1]][[i]][[j]], 0]/dz^2
329     ),
330     {j, 1, Dimensions[Temperatures][[3]]},
331     {i, 1, Dimensions[Temperatures][[2]]},
332     {k, 1, Dimensions[Temperatures][[1]]};
333
334 (* The simulation method *)
335 (* n: number of sides *)
336 (* \[Alpha]: Diffusion parameter *)
337 (* \[Beta]: Pan heating parameter *)
338 (* \[Gamma]: Air heating parameter *)
339 (* w: Pan width *)
340 (* h: Brownie height *)
341 (* T: Time of simulation *)
342 (* nx: Number of subdivisions along each horizontal axis *)
343 (* nz: Number of subdivisions along the vertical axis *)
344 (* nt: Number of time steps *)
345 (* Temp: Oven/pan temperature *)
346 Simulation[n_, \[Alpha]_, \[Beta]_, \[Gamma]_, w_, h_, T_, nx_, nz_,
347     nt_, Temp_] := Module[
348     {A, P, dw, dh, dt, Stencil, Border, AreaGrid3D, AreaGrid3DTop,
349     TempSums, Temperatures, i},
350     (* Calculate size of spatial and time steps *)
351     dw = w/nx;
352     dh = h/nz;
353     dt = T/nt;
354
355     (* Calculate the correct area and perimeter *)
356     A = 1/4 n w^2 Sin[(1 - (n - 2)/n) \[Pi]/
357     2] Cos[(1 - (n - 2)/n) \[Pi]/2];
358     P = n w Sin[(1 - (n - 2)/n) \[Pi]/2];
359
360     (* Generate stencil and border tables *)
361     Stencil = CalculateStencil[n, nx];
362     Border = CalculateBorder[n, nx];
363
364     (* Use 2D stencil and border to Calculate 3D surface area tables \
365     for the pan and the air *)
366     AreaGrid3DTop =
367     Join[Table[
368         Table[Table[0.0, {y, 1, nx}], {x, 1, nx}], {z, 1,
369         nz - 1}], {Stencil}];
370     AreaGrid3DTop = AreaGrid3DTop*A/Total[Total[Stencil]];
371     AreaGrid3D = Table[Border, {i, 1, nz}];
372     AreaGrid3D = AreaGrid3D*P*h/(Total[Total[Border]]*nz);
373     AreaGrid3D[[1]] = AreaGrid3D[[1]] + AreaGrid3DTop[[nz]];
374
375     (* Setup temperature table *)
376     Temperatures =
377     Table[Table[
378         Table[22.4*Stencil[[i]][[j]], {j, 1, nx}], {i, 1, nx}], {k, 1,
379         nz}];
380     TempSums =
381     Table[Table[
382         Table[22.4*Stencil[[i]][[j]], {j, 1, nx}], {i, 1, nx}], {k, 1,
383         nz}];
384
385     (* Advance the simulation through all the time steps *)
386     For[i = 1, i <= nt, i = i + 1,
387         Temperatures = Temperatures +
388         (HeatTransfer[\[Beta], Temp, dw, dw, dh, AreaGrid3D,
389         Temperatures] +
390         HeatTransfer[\[Gamma], Temp, dw, dw, dh, AreaGrid3DTop,
391         Temperatures] +
392         Diffusion[\[Alpha], dw, dw, dh, Stencil, Temperatures])*dt;
393         TempSums = TempSums + Temperatures;
394     ];
395
396     (* Return the final temperatures and the average temperature for \
397     every data point *)
398     {Temperatures, TempSums/(nt + 1)}
399 ];

```

## Space Optimization, Fitted Heat Equation, and Weighted Evaluation

```
1
2 L = 30; W = 60; a = 81; percent = 1.08; (* constants used in pan area \
3 models *)
4 side[n_, A_] := Sqrt[(
5 4 A Tan[\[Pi]/n])/n]; \[Theta][n_] := (n - 2)/n \[Pi];
6 apothem[n_, A_] := Sqrt[A/(n Tan[\[Pi]/n])];
7 radius[n_, A_] := Sqrt[(2 A)/(n Sin[2 \[Pi]/n])];
8 perimeter[n_, A_] := n*side[n, A]; height[n_, A_] := !\(\(*
9 TagBox[GridBox[{
10 {\[Piecewise]", GridBox[{
11 {
12 RowBox[{"2",
13 RowBox[{"apothem", "["],
14 RowBox[{"n", ",", "A"}], "]"}}}],
15 RowBox[{"Mod", "["],
16 RowBox[{"n", ",", "2"}], "]"}, {"=", "0"}]}],
17 {
18 RowBox[{"apothem", "["],
19 RowBox[{"n", ",", "A"}], "]"}, {"+",
20 RowBox[{"radius", "["],
21 RowBox[{"n", ",", "A"}], "]"}}}],
22 RowBox[{"Mod", "["],
23 RowBox[{"n", ",", "2"}], "]"}, {"=", "1"}]}],
24 },
25 AllowedDimensions->{2, Automatic},
26 Editable->True,
27 GridBoxAlignment->{
28 "Columns" -> {Left}, "ColumnsIndexed" -> {},
29 "Rows" -> {Baseline}, "RowsIndexed" -> {}},
30 GridBoxItemSize->{
31 "Columns" -> {Automatic}, "ColumnsIndexed" -> {},
32 "Rows" -> {1.}, "RowsIndexed" -> {}},
33 GridBoxSpacings->{"Columns" -> {
34 Offset[0.27999999999999997`], {
35 Offset[0.84]},
36 Offset[0.27999999999999997`], "ColumnsIndexed" -> {}, "Rows" -> {
37 Offset[0.2], {
38 Offset[0.4]},
39 Offset[0.2]}, "RowsIndexed" -> {}},
40 Selectable->True]}],
41 },
42 GridBoxAlignment->{
43 "Columns" -> {Left}, "ColumnsIndexed" -> {},
44 "Rows" -> {Baseline}, "RowsIndexed" -> {}},
45 GridBoxItemSize->{
46 "Columns" -> {Automatic}, "ColumnsIndexed" -> {},
47 "Rows" -> {1.}, "RowsIndexed" -> {}},
48 GridBoxSpacings->{"Columns" -> {
49 Offset[0.27999999999999997`], {
50 Offset[0.35]},
51 Offset[0.27999999999999997`], "ColumnsIndexed" -> {}, "Rows" -> {
52 Offset[0.2], {
53 Offset[0.4]},
54 Offset[0.2]}, "RowsIndexed" -> {}},
55 "Piecewise",
56 DeleteWithContents->True,
57 Editable->False,
58 SelectWithContents->True,
59 Selectable->False});
60 MagicNumberApothem[n_, A_] := Sqrt[(4 Cot[\[Pi]/n])/n]*Sqrt[A] // N;
61 MagicNumberRadius[n_, A_] :=
62 Sqrt[2/(n Sin[2 \[Pi]/n])]*Sqrt[A] //
63 N; (* This section is dedicated to finding the different dimensions \
64 of a regular polygon including sides, apothem, and radius. We are also \
65 able to find the magic numbers for apothem and radius depending on \
66 the shape and area of a certain polygon. *)
67 numberOfcols[s_, A_, l_, w_, p_] :=
68 Floor[l/(2 apothem[s,
69 p A])] (* calculate number of columns for even non-multiple of 4 \
70 n-gons *)
71 evenrow[s_, A_, l_, w_, p_] :=
```

```

75  If[1 - 2 numberofcols[s, A, l, w, p] apothem[p s, A] -
76  apothem[p s, A] >= 0, True,
77  False] (* checks to see if it is possible to fit another n-gon in \
78  every other row *)
79  numberofrows[s_, A_, l_, w_, p_] :=
80  Do[i = -1; x = w;
81  While[If[EvenQ[i], x, x - side[s, p A]/2] >= 0,
82  x = x - If[OddQ[i], 2*side[s, p A], side[s, p A]]; i++];
83  Return[i, {l}] (* finds number of rows for even non-multiple of 4 \
84  n-gons in tessellating pattern *)
85  numberofhexs[s_, A_, l_, w_, p_] :=
86  If[evenrow[s, A, l, w, p] == True,
87  numberofrows[s, A, l, w, p]*numberofcols[s, A, l, w, p],
88  numberofrows[s, A, l, w, p]*numberofcols[s, A, l, w, p] -
89  Floor[numberofrows[s, A, l, w, p]/
90  2]] (* calculates the number of hexagons for a particular w and l \
91  oven *)
92  latticearrangel[s_, A_, l_, w_] :=
93  Floor[w/If[IntegerQ[s/4], height[s, A] // N, (2 radius[s, A])]]*
94  Floor[l/height[s, A]] (* first lattice arrangement of n-gons *)
95  latticearrange2[s_, A_, l_, w_] :=
96  Floor[w/height[s, A]]*
97  Floor[l/If[IntegerQ[s/4],
98  height[s, A] //
99  N, (2 radius[s, A])]] (* second lattice arrangement of n-gons *)
100
101
102  hexadaptarrangel[s_, A_, l_, w_, p_] :=
103  numberofhexs[s, A, l, w,
104  p] (* first tessellating arrangement of n-gons *)
105  hexadaptarrange2[s_, A_, l_, w_, p_] :=
106  numberofhexs[s, A, w, l,
107  p] (* second tessellating arrangement of n-gons *)
108  (* calculates number of n-gons that can fit in a particular w and l \
109  rectangular oven *)
110
111  numberofpolys[s_, A_, l_, w_, p_] :=
112  If[s != 6 && s != 4,
113  If[latticearrangel[s, A, l, w] >= latticearrange2[s, A, l, w],
114  latticearrangel[s, A, l, w], latticearrange2[s, A, l, w]],
115
116  If[s != 4,
117  If[
118  If[hexadaptarrangel[s, A, l, w, p] >=
119  hexadaptarrange2[s, A, l, w, p],
120  hexadaptarrangel[s, A, l, w, p],
121  hexadaptarrange2[s, A, l, w, p]] >=
122  If[latticearrangel[s, A, l, w] >= latticearrange2[s, A, l, w],
123  latticearrangel[s, A, l, w], latticearrange2[s, A, l, w]],
124
125  If[hexadaptarrangel[s, A, l, w, p] >=
126  hexadaptarrange2[s, A, l, w, p],
127  hexadaptarrangel[s, A, l, w, p], hexadaptarrange2[s, A, l, w, p]],
128
129  If[latticearrangel[s, A, l, w] >= latticearrange2[s, A, l, w],
130  latticearrangel[s, A, l, w], latticearrange2[s, A, l, w]]],
131
132  If[latticearrangel[s, p A, l, w] >= latticearrange2[s, p A, l, w],
133  latticearrangel[s, p A, l, w],
134  latticearrange2[s, p A, l,
135  w]]] (* calculates number of n-gons that can fit in a \
136  particular w and l rectangular oven *)
137  Eff[s_, A_, w_, l_, p_] := (numberofpolys[s, A, l, w, p]*A)/(
138  l*w) (* calculates the oven space efficiency for one shelf for \
139  particular n-gon and rectangular oven *)
140  (*Effheat[s_, A_] := .0114944*s + .340489/Sqrt[A] + .801163*)
141  Effheat[s_, A_] :=
142  Min[1, .055657869822367416*
143  ArcTan[.2259800337169932*s -
144  1.6457785931252638] + .10716283573624863/(
145  Sqrt[A] - 2.462771976211951) + .9045771916045253]
146
147  ranknumber[shaperank_, heatrank_, w_] := shaperank w + (1 - w) heatrank
148
149  Effgenerate2[A_, l_, w_, p_, weight_, polylist_] :=
150  Do[polylistlocal = Sort[polylist];
151  If[Length[polylistlocal] < 6, Break[]];
152
153  rank = {}; spatialeff = 0;
154  For[gg = 1, gg <= Length[polylistlocal], gg++,
155  rank = Append[

```



```

156     rank, {polylistlocal[[gg]],
157     Eff[polylistlocal[[gg]], A, l, w, p] // N}}];
158
159 spatialeff = RankedMax[Flatten[rank], Length[rank] + 1];
160 For[o = 1, o <= Length[rank], o++, rank[[o]][[2]] /= spatialeff];
161
162 SortBy[rank, Last];
163 For[qq = 1; heateff = {}, qq <= Length[rank], qq++,
164   heateff = Append[heateff, Effheat[rank[[qq]][[1]], A] // N]];
165
166 For[ss = 1; shape = {}; sideorder = {}, ss <= Length[polylistlocal],
167   ss++, sideorder = Append[sideorder, rank[[ss]][[1]]];
168   shape = Append[shape,
169     ranknumber[rank[[ss]][[2]], heateff[[ss]], weight]];
170
171 Return[sideorder[[Part[Flatten[Position[shape, Max[shape]]], 1]]]
172   , {1}]
173
174 PrintResults[A_, l_, w_, p_, weight_, polylist_] :=
175 Print["The_most_efficient_pan_based_on_the_parameters_is_a_",
176   Effgeneratel[A, l, w, p, weight, polylist], "-sided_polygon"]

```