Trabajo Práctico Final
Bin packing de polígonos simples

Análisis de lenguajes de programación

Marzorati, Denise
M-6219/7

2017

Docentes de la materia

Mauro Jaskelioff
Cecilia Mazino
Eugenia Simich
Juan Martín Rabasedas

Definiendo el problema

Motivación

Dada una plancha de madera de un ancho A y largo L, en los aserraderos tan solo se realizan cortes longitudinales o transversales de la misma para la venta al público. Por este motivo es necesario que el cliente que se acerca a realizar su compra minimice el "área rectangular" que ocuparán las figuras con las cuales luego trabajará. Al hablar de área rectangular se hace referencia al área del rectángulo de menor área que contiene a todas las figuras dadas por el cliente.

Consideraciones

- No se trabajará con figuras arbitrarias, sino que simplificará el problema considerando tan solo polígonos simples; es decir, polígonos cuyos lados no adyacentes no se intersecan (ya que el caso de polígonos complejos puede reducirse a un problema de bin packing de polígonos simples). Por otro lado, la elección de polígonos como elementos de trabajo se basa en la posibilidad de chequear fácilmente si un par de ellos se encuentran solapados o no (es decir, si hay intersección entre sus lados).
- Si bien en el caso real se cuentan con varias planchas del mismo tamaño (con lo cual si las figuras no caben en una sola de ellas se utilizan las restantes), aquí se determina si se pueden acomodar todas las figuras dentro de una única plancha, devolviendo la correspondiente solución o bien un error.
- Se tiene en cuenta el material que quitará la herramienta de corte del cliente al cortar cada polígono, dado por el espesor y el tipo de la herramienta. A este valor se le denomina "kerf".

Instalación

Los módulos de Haskell que deben estar instalados son los siguientes:

+ AI.GeneticAlgorithm.Simple
+ Algorithms.Geometry
+ Control.DeepSeq
+ Control.Exception
+ Control.Lens
+ Control.Monad
+ Data.ByteString
+ Data.Char
+ Data.Ext
+ Data.Geometry
+ Data.List
+ Data.Maybe
+ Data.Set
+ Data.String
+ Data.Text
+ GHC.Float
+ Graphics.Gloss
+ System.Console.Readline
+ System.Directory
+ System.Environment
+ System.IO
+ System.Random
+ Text.Blaze
+ Text.PrettyPrint.HughesPJ
+ Text.XmlHtml

Para hacer uso del software tan solo es necesario compilar el archivo Main.hs con ghc, y ejecutar el ejecutable generado a partir de dicha compilación.

Organización de los archivos

- *Common.hs*: en este archivo se encuentran las definiciones de tipos de datos referidos a polígonos, rectángulos, máquina (que es un conjunto de polígonos y rectángulos junto con el kerf), y tipos de datos para los elementos pertinentes de archivos SVG.
- *Draw.hs*: aquí se encuentran las funciones encargadas de crear el texto que conformará el archivo SVG que se devuelve como resultado. Cabe destacar que en el archivo SVG que devuelve como resultado la máquina de corte debe ir por las líneas verdes, y con el material que quita adicionalmente quedara troquelado el polígono correspondiente que se encuentra dibujado en negro.
- *Eval.hs*: se evalúan las definiciones, obteniendo los polígonos resultantes de aplicar las transformaciones que correspondan, y chequeando que los mismos sean válidos (es decir, tengan al menos tres lados y sean polígonos simples). Además se le agrega el valor del kerf a cada polígono de manera tal que al ejecutar el algoritmo genético lo tenga en cuenta.
- *GA.hs*: etapa del algoritmo genético, para revisar los detalles acerca del mismo ver [1].
- *Main.hs*: desarrolla el intérprete del programa que acepta como entrada archivos cuya sintaxis describe polígonos, como así también archivos SVG que tengan definidos polígonos o rectángulos. Se utilizó como guía [2].
- *Parse.hs*: parsea sentencias del DSL diseñado (de "máquinas").
- *ParseSVGElements.hs*: parsea las componentes que son de interés de los elementos deseados de un archivo SVG.
- *ParseSVG.hs*: utiliza bibliotecas ya desarrolladas para "pre parsear" un archivo SVG, quedándose tan solo con los elementos que son de interés del mismo, y devolviendo texto con la sintaxis que ParseSVGElements acepta como argumento de entrada.
- *Polygon.hs*: contiene funciones utilizadas en la última etapa del proceso donde una vez que el algoritmo genético ha finalizado se vuelven a "introducir" los polígonos dentro de su respectivo contenedor.
- *PrettyPrint.hs*
- *Shrink.hs*: se encarga de la etapa de acercamiento de los polígonos.

DSL de "máquinas"

Sintaxis abstracta

$$\langle floatExp \rangle ::= \langle float \rangle \mid -_1 \langle float \rangle$$

$$\langle point \rangle ::= \langle floatExp \rangle \sim \langle floatExp \rangle$$

$$\langle polygon \rangle ::= \langle point \rangle \mid \langle point \rangle \, , \langle point \rangle$$

⟨*container*⟩ ::= ⟨*float*⟩ × ⟨*float*⟩

⟨*defExp*⟩ ::= **pol** ⟨*name*⟩ = ⟨*polygon*⟩
| **con** ⟨*name*⟩ = ⟨*container*⟩

⟨*defs*⟩ ::= ⟨*empty*⟩
| ⟨*defExp*⟩ ⟨*defs*⟩
| **pol** ⟨*name*⟩ = ⟨*polygon*⟩ **copy** ⟨*natural*⟩ **scale** ⟨*float*⟩ ⟨*defs*⟩
| **con** ⟨*name*⟩ = ⟨*container*⟩ **copy** ⟨*natural*⟩ **scale** ⟨*float*⟩ ⟨*defs*⟩

⟨*machine*⟩ ::= **kerf** ⟨*float*⟩ ⟨*defs*⟩

Sintaxis concreta

⟨*digit*⟩ ::= '0' | '1' | ... | '9'

⟨*natural*⟩ ::= ⟨*digit*⟩
| ⟨*digit*⟩ ⟨*natural*⟩

⟨*float*⟩ ::= ⟨*natural*⟩ '.' ⟨*natural*⟩

⟨*floatExp*⟩ ::= ⟨*float*⟩
| '-' ⟨*float*⟩

⟨*point*⟩ ::= ⟨*floatExp*⟩ '∼' ⟨*floatExp*⟩

⟨*polygon*⟩ ::= ⟨*point*⟩
| ⟨*point*⟩ ',' ⟨*polygon*⟩

⟨*bPolygon*⟩ ::= '[' ']'
| '[' ⟨*polygon*⟩ ']'

⟨*container*⟩ ::= ⟨*float*⟩ '×' ⟨*float*⟩

⟨*defExp*⟩ ::= **pdef** ⟨*name*⟩ '=' ⟨*bPolygon*⟩
| **'con'** ⟨*name*⟩ '=' ⟨*container*⟩

⟨*defs*⟩ ::= ⟨*defExp*⟩ ⟨*defs*⟩
| **pdef** ⟨*name*⟩ '=' ⟨*bPolygon*⟩ **'copy'** ⟨*natural*⟩ **'scale'** ⟨*float*⟩ ⟨*defs*⟩
| **cdef** ⟨*name*⟩ '=' ⟨*container*⟩ **'copy'** ⟨*natural*⟩ **'scale'** ⟨*float*⟩ ⟨*defs*⟩
|

⟨*machine*⟩ ::= **'kerf'** ⟨*float*⟩ ⟨*defs*⟩

DSL de SVG

SVG cuenta con varios tipos de elementos que describen distintas figuras; los elementos que pueden resultar de interés para este trabajo son los siguientes: <rect>, <polygon>, <polyline> y <path>. Se utilizó el módulo Text.XmlHtml para obtener los elementos que son de importancia para el objetivo planteado (por ejemplo: la rotación de una figura no es relevante), con lo cual luego dicho resultado debe ser parseado. A continuación se listan los campos relevantes de cada elemento, en el orden en que serán pasados al parser:

- <rect>: width, height, transform, id. A este elemento solo se le permiten las transformaciones de translación, rotación o escalamiento (ya que transformaciones arbitrarias pueden dar como resultado otro tipo de cuadriláteros que no son rectángulos).
- <polygon>: points, transform, id.
- <polyline>: points, transform, id.
- <path>: d, transform, id.

Para más detalles acerca de la sintaxis de dicho lenguaje ver las referencias.

Sintaxis abstracta

⟨*floatExp*⟩ ::= ⟨*float*⟩
| $-_1$ ⟨*float*⟩

⟨*cPoint*⟩ ::= ⟨*floatExp*⟩ , ⟨*floatExp*⟩

⟨*point*⟩ ::= ⟨*floatExp*⟩ ⟨*floatExp*⟩
| ⟨*cPoint*⟩

⟨*rectList*⟩ ::= ⟨*empty*⟩
| **translate** ⟨*floatExp*⟩ ⟨*rectList*⟩
| **translate** ⟨*cPoint*⟩ ⟨*rectList*⟩
| **rotate** ⟨*floatExp*⟩ ⟨*rectList*⟩
| **scale** ⟨*floatExp*⟩ ⟨*rectList*⟩

⟨*sufixR*⟩ ::= ⟨*rectList*⟩ ⟨*name*⟩

⟨*rect*⟩ ::= ⟨*floatExp*⟩ ⟨*floatExp*⟩ ⟨*sufixR*⟩

⟨*transform*⟩ ::= ⟨*translate*⟩ ⟨*floatExp*⟩
| **translate** ⟨*floatExp*⟩ ⟨*cPoint*⟩
| **translate** ⟨*cPoint*⟩ ⟨*rectList*⟩
| **rotate** ⟨*floatExp*⟩
| **scale** ⟨*floatExp*⟩
| **skewX** ⟨*floatExp*⟩
| **skewY** ⟨*floatExp*⟩
| **matrix** ⟨*floatExp*⟩ , ⟨*floatExp*⟩ , ⟨*floatExp*⟩ , ⟨*floatExp*⟩ , ⟨*floatExp*⟩ , ⟨*floatExp*⟩

⟨*transformList*⟩ ::= ⟨*empty*⟩
| ⟨*transform*⟩ ⟨*transformList*⟩

⟨*sufix*⟩ ::= ⟨*transformList*⟩ ⟨*name*⟩

⟨*pointList*⟩ ::= ⟨*empty*⟩
| ⟨*point*⟩ ⟨*pointList*⟩

⟨*polygon*⟩ ::= ⟨*pointList*⟩ ⟨*sufix*⟩

⟨*polyline*⟩ ::= **poly** ⟨*pointList*⟩ ⟨*sufix*⟩

⟨*startPath*⟩ ::= **M** ⟨*point*⟩ ⟨*pathLines*⟩
| **m** ⟨*point*⟩ ⟨*pathLines*⟩

⟨*pathLines*⟩ ::= **h** ⟨*floatExp*⟩ ⟨*pathLines*⟩
| **H** ⟨*floatExp*⟩ ⟨*pathLines*⟩
| **v** ⟨*floatExp*⟩ ⟨*pathLines*⟩
| **V** ⟨*floatExp*⟩ ⟨*pathLines*⟩
| **l** ⟨*floatExp*⟩ ⟨*pathLines*⟩
| **L** ⟨*floatExp*⟩ ⟨*pathLines*⟩
| ⟨*point*⟩ ⟨*pathLines*⟩
| **z**

⟨*path*⟩ ::= ⟨*startPath*⟩ ⟨*sufix*⟩

Sintaxis concreta

⟨*digit*⟩ ::= '0' | '1' | ... | '9'

⟨*natural*⟩ ::= ⟨*digit*⟩
| ⟨*digit*⟩ ⟨*natural*⟩

⟨*float*⟩ ::= ⟨*natural*⟩ '.' ⟨*natural*⟩

⟨*floatExp*⟩ ::= ⟨*float*⟩
| $-_1$ ⟨*float*⟩

⟨*char*⟩ ::= 'a' | 'b' | ... | 'z' | 'A' | 'B' | ... | 'Z'

⟨*name*⟩ ::= ⟨*empty*⟩
| ⟨*char*⟩ ⟨*name*⟩
| ⟨*digit*⟩ ⟨*name*⟩

⟨*cPoint*⟩ ::= ⟨*floatExp*⟩ ',' ⟨*floatExp*⟩

⟨*point*⟩ ::= ⟨*floatExp*⟩ ⟨*floatExp*⟩
| ⟨*cPoint*⟩

⟨*rectList*⟩ ::= **'translate'** ⟨*floatExp*⟩ ⟨*rectList*⟩
| **'translate'** ⟨*cPoint*⟩ ⟨*rectList*⟩
| **'rotate'** ⟨*floatExp*⟩ ⟨*rectList*⟩
| **'scale'** ⟨*floatExp*⟩ ⟨*rectList*⟩
|

⟨*sufixR*⟩ ::= ⟨*rectList*⟩ ⟨*name*⟩

⟨*rect*⟩ ::= ⟨*floatExp*⟩ ⟨*floatExp*⟩ ⟨*sufixR*⟩

⟨*transform*⟩ ::= ⟨*translate*⟩ ⟨*floatExp*⟩
| **'translate'** ⟨*floatExp*⟩ ⟨*cPoint*⟩
| **'translate'** ⟨*cPoint*⟩ ⟨*rectList*⟩
| **'rotate'** ⟨*floatExp*⟩
| **'scale'** ⟨*floatExp*⟩
| **'skewX'** ⟨*floatExp*⟩
| **'skewY'** ⟨*floatExp*⟩
| **'matrix'** ⟨*floatExp*⟩ ',' ⟨*floatExp*⟩ ',' ⟨*floatExp*⟩ ',' ⟨*floatExp*⟩ ',' ⟨*floatExp*⟩ ',' ⟨*floatExp*⟩

⟨*transformList*⟩ ::= ⟨*transform*⟩ ⟨*transformList*⟩
    |

⟨*sufix*⟩ ::= ⟨*transformList*⟩ ⟨*name*⟩

⟨*pointList*⟩ ::= ⟨*point*⟩ ⟨*pointList*⟩
    |

⟨*polygon*⟩ ::= ⟨*pointList*⟩ ⟨*sufix*⟩

⟨*polyline*⟩ ::= **'poly'** ⟨*pointList*⟩ ⟨*sufix*⟩

⟨*startPath*⟩ ::= **'M'** ⟨*point*⟩ ⟨*pathLines*⟩
    | **'m'** ⟨*point*⟩ ⟨*pathLines*⟩

⟨*pathLines*⟩ ::= **'h'** ⟨*floatExp*⟩ ⟨*pathLines*⟩
    | **'H'** ⟨*floatExp*⟩ ⟨*pathLines*⟩
    | **'v'** ⟨*floatExp*⟩ ⟨*pathLines*⟩
    | **'V'** ⟨*floatExp*⟩ ⟨*pathLines*⟩
    | **'l'** ⟨*floatExp*⟩ ⟨*pathLines*⟩
    | **'L'** ⟨*floatExp*⟩ ⟨*pathLines*⟩
    | ⟨*point*⟩ ⟨*pathLines*⟩
    | **'z'**

⟨*path*⟩ ::= ⟨*startPath*⟩ ⟨*sufix*⟩

Observaciones

- En varios de los módulos donde se deben realizar cálculos se utilizan cotas superiores o inferiores debido a la inexactitud de las operaciones (por ejemplo al considerar la intersección de dos rectas, si esta coincide con un punto dado).

References

[1] Stefan Jakobs, *On genetic algorithms for the packing of polygons*, European journal of operational research, 1996.

[2] Mauro Jaskelioff, Cecilia Manzino, Juan Manuel Rabasedas, Eugenia Simich *Intérprete de lambda cálculo*, FCEIA, 2017.

[3] W3C, *Basic shapes*, W3, 16 August 2011, https://www.w3.org/TR/SVG/shapes, 2017.