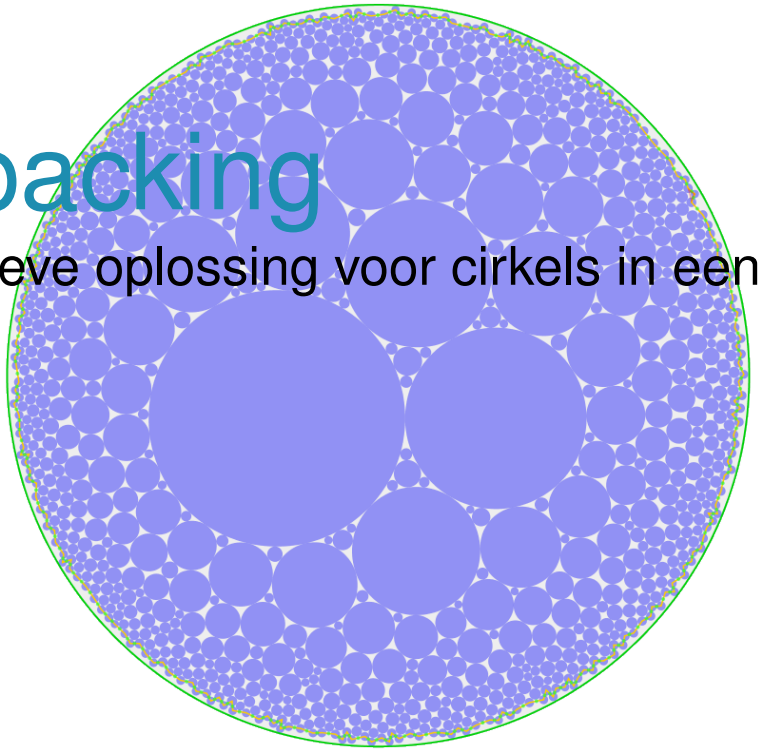
KU LEUVEN

FACULTEIT WETENSCHAPPEN

Circle packing

Een constructieve oplossing voor cirkels in een cirkel



Pablo BOLLANSÉE

Promotor: Prof. P. De Causmaecker
Affiliatie *(facultatief)*

Co-promotor: *(facultatief)*
Affiliatie *(facultatief)*

Begeleider: *(facultatief)*
Affiliatie *(facultatief)*

Proefschrift ingediend tot het behalen van de graad van Master of Science in de toegepaste informatica

Academiejaar 2015-2016

Voorwoord

Het Circle-Packing probleem bestaat er uit om een aantal cirkels, met gekende radii, in een zo klein mogelijke container te plaatsen. De vorm van deze container kan verschillen, meestal is het een driehoek, rechthoek of cirkel. In deze thesis bekijken we het packings van cirkels in een cirkel.

Wiskunde is dit een relatief eenvoudig probleem om voor te stellen, maar computationeel is het zeer zwaar om exact op te lossen. Bestaande pogingen om dit probleem op te lossen geven zeer dichte packings, maar vragen zeer veel tijd. In deze thesis stel ik een nieuwe constructieve heuristiek voor om deze packings te maken, die het mogelijk maakt zeer snel oplossingen te genereren.

Ik wil hierbij Patrick De Causmaeker bedanken voor alle hulp en ondersteuning tijdens het verwezenlijken van dit werk.

Abstract

TODO

Inhoud

Lijst van figuren

Lijst van tabellen

Hoofdstuk 1

Inleiding

Het Circle-Packing probleem (CPP), voor cirkels in een cirkel, bestaat uit het plaatsen van n cirkels in een zo klein mogelijke cirkelvormige container. Het is de bedoeling om voor de gegeven cirkels de coordinaten van de middelpunten te vinden zodat deze niet overlappen en de radius van de omcirkel zo klein mogelijk is.

Circle-Packing is zowel theoretisch als practisch een zeer interessant probleem. Het kan gebruikt worden om verschillende real-world problemen op te lossen, zoals het plaatsen van zendmasten, stokage van cilindrische voorwerpen, en het combineren van verschillende kabels.

Mathematisch is het redelijk eenvoudig als een optimalisatie probleem te omschrijven:

$$\begin{array}{lll} \text{minimaliseer} & r & \\ \text{onderhevig aan} & x_i^2 + y_i^2 \leq (r - r_i)^2, & i = 1, ..., n \\ & (x_i - x_j)^2 + (y_i + y_j)^2 \geq (r_i + r_j)^2, & i \neq j \end{array}$$

Hierin is r_i de radius, en (x_i, y_i) de coördinaten van het centrum van cirkel i. Hierbij wordt verondersteld dat de omcirkel het nulpunt als middelpunt heeft. De eerste formule verzekerd dat de cirkels in de omcirkel liggen, en de tweede dat ze elkaar niet overlappen. In het geval dat alle cirkels de zelfde grootte hebben wordt meestal r_i altijd gelijk aan 1 genomen.

Hoewel dit wiskundig zeer eenvoudig te omschrijven is, blijft het toch een zeer moeilijk probleem om exact op te lossen. Het is een NP-hard probleem. Daarom zoekt men naar andere technieken om het toch op te kunnen lossen. In [5] stellen ze een Monotonic Basin Hopping algoritme voor. Hierin beschrijven ze dat er teveel lokale optima zijn voor een eenvoudige multi-start behandeling, en stellen een variant voor waarin ze op een slimme manier de begin punten proberen genereren. In [2] wordt gebruik gemaakt van de combinatorische eigenschappen van circle-packing in combinatie met een taboo-search en een off-the-shelf non-linear optimizer. Hierin plaatsen ze één voor één elke cirkel en laat de non-linear optimizer hiervoor telkens een lokaal extremum berekenen. Ze zoeken van met de taboo-search naar de beste volgorde om de cirkels te plaatsen. Hoewel deze oplossingen zeer goede packings maken, regelmatig tot op heden de best gekende oplossingen [6], vragen ze zeer veel reken tijd. Constructieve algoritmen voor het oplossen van circle-packings zijn veel minder onderzocht. De enige voor mij bekende is [1], waarin ze een alternatieve vorm van circle-packings oplossen: de omtrek van de container ligt vast, en je moet zo veel mogelijk cirkels van gelijke grootte er in plaatsen.

In deze thesis stel ik een nieuw constructieve heuristiek voor om het circle-packing probleem op te lossen. Het is een best-fit heuristiek gebaseerd op een oplossing voor het Orthogonal Stock-Cutting Problem voorgesteld in [4]. Zij stellen een heuristiek voor die de volgende balk om te plaatsen kiest uit een lijst, en deze plaatst op de *beste* positie. Dit in tegenstelling tot cirkels plaatsen in een vooraf bepaalde volgorde zoals in [5] en [2]. Op een gelijkaardige manier kiest mijn algoritme de volgende cirkel die best past in de huidige packing.

In hoofdstuk 3 bespreek ik hoe de heuristiek opgebouwd is. Ik bespreek de twee basis concepten voor mijn best-fit heuristiek: *holes* en de *shell*. Ik bespreek hoe deze werken, en op welke manier gekozen wordt welke cirkel best past in de packing. Hierbij bespreek ik ook de implementatie. In hoofdstuk 4 worden de verkregen resultaten besproken. Hier vergelijk ik de packings met de best gekende resultaten zoals gerapporteerd op de Packomania website ([6]). Ik doe hier een vergelijking zowel op omtrek van de verkregen omcirkel, als op nodige tijd om deze packing te berekenen tegenover de best gekende oplossingen. Ook toon ik resultaten voor packings voor veel meer cirkels dan getoond op de Packomania website. In hoofdstuk 5 bespreek ik mogelijke verbeteringen, de *losse eindjes* en ideeën voor verdere uitbreidingen en onderzoek. In hoofdstuk 2 wordt kort verduidelijkt hoe u de visualisaties gebruikt doorheen deze thesis kan interpreteren.

Hoofdstuk 2

Handleiding voor het lezen van de visualisaties

Doorheen deze thesis zal ik gebruik maken visualisaties gegenereerd door de java implementatie van het algoritme. Dit om de concepten grafisch te verduidelijken. Twee voorbeelden van zulke visualisaties zijn figuur 2.1 en figuur 2.2.

Deze figuren kan u op de volgende manier interpreteren:

- De **reeds geplaatste cirkels** worden getoond als **licht blauwe cirkels**.
- De **shell** is een **gele lijn** aan de buitenste rand van de packing.
- De kleine **groene bolletjes** op de shell geven de posities aan waarop mogelijk een cirkel geplaatst zal worden.
- **Holes** worden getoond als **rode driehoeken**.
- De **omcirkel** van de huidige packing wordt getoond als een **groene cirkel**.

Op figuur 2.1 is zijn er duidelijk drie holes te zien. Elk van de drie holes word gedefinieerd door de centrale cirkel en twee van de buitenste cirkels. Ook is er een kleine shell te zien, die bestaat uit de buitenste drie cirkels. In figuur 2.2 wordt een verder gevorderde packing getoond waarop één hole te zien is, en een veel grotere shell. Op beide figuren kan je ook de omcirkel zien.

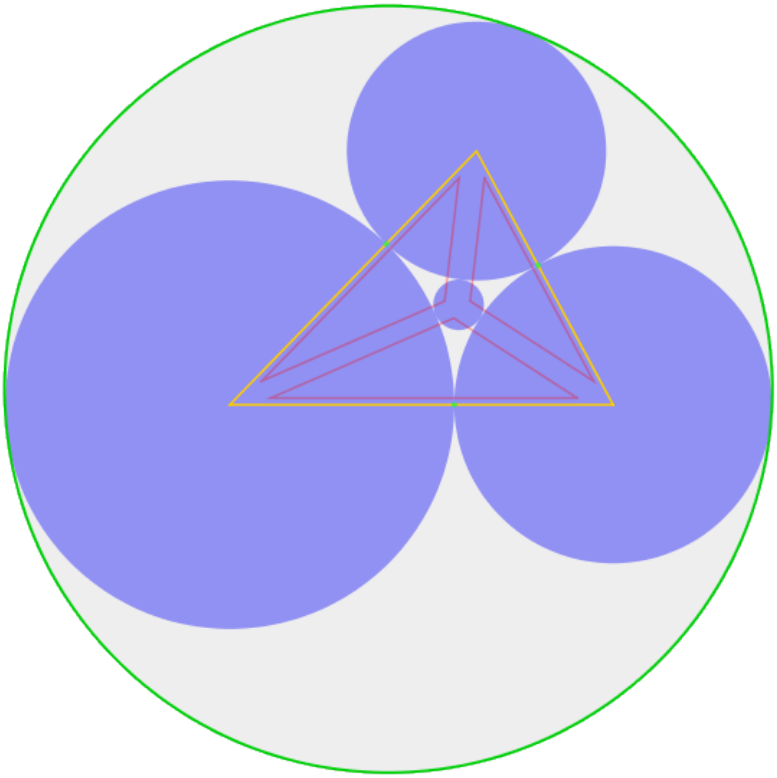


Figure 2.1: Voorbeeld visualisatie met drie duidelijke holes

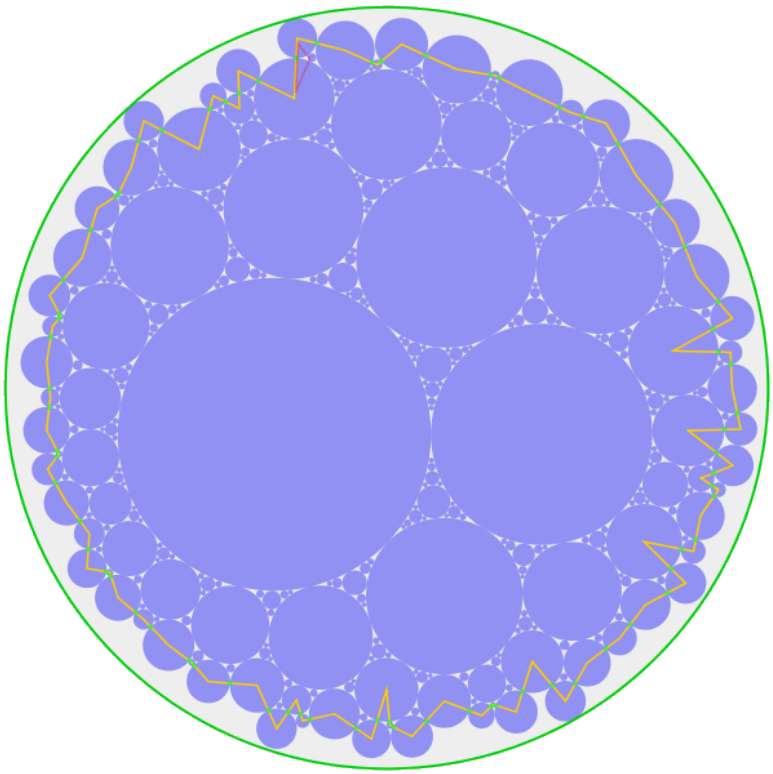


Figure 2.2: Voorbeeld visualisatie met grote shell

Hoofdstuk 3

Algoritme

In dit hoofdstuk bespreek ik de werking van de heuristiek. Eerst geef ik een korte beschrijving van het basis idee van het algoritme, gevolgd door de structuur van de code. Vervolgens leg ik stelselmatig de volledige werking uit, alle veronderstellingen die gemaakt worden en implementatie details waar nodig. De volledige implementatie is beschikbaar op GitHub [3] en in gebeurt in Java.

3.1 Basis idee

Het basis idee van de heuristiek is om stelselmatig een packing op te bouwen. Er is één grote lus waarin telkens eerst een plek gekozen wordt om een cirkel te plaatsen (in een hole, of op de shell, later hier meer over). Eenmaal een cirkel geplaatst is wordt deze nooit meer verplaatst. Dit laat toe om intelligente structuren op te bouwen en deze op een zeer efficiënte manier te gebruiken.

Het algoritme bouwt dus cirkel per cirkel een packing op. Dit door in elke stap een positie te kiezen, en hierin een cirkel te proberen plaatsen. Indien er geen cirkel geplaatst kan worden wordt de interne structuur van het probleem vernieuwd om deze nieuwe informatie te reflecteren. Dit gebeurt op verschillende manieren voor de holes en de shell. Meer hierover vindt u terug in sectie 3.5 en sectie 3.6. Als er wel een cirkel geplaatst kan worden dan wordt deze uit de lijst van nog-te-plaatsen cirkels verwijderd, en krijgt deze een permanente positie daar. Dit geeft ook aanleiding tot aanpassen van de holes en/of shell. Hierdoor wordt er een nieuwe tussentijdse packing gemaakt. Deze wordt dan door gegeven naar de volgende stap, waarin het algoritme opnieuw zal proberen een cirkel te plaatsen. Op deze manier word een volledige packing opgebouwd voor alle cirkels.

3.2 Structuur

De implementatie van het algoritme is op te delen in drie belangrijke delen.

- Problem of probleem
- Solution of oplossing
- Solver of oplosser

Bovenop deze delen komen dan ook nog enkele voor de hand liggende hulp klassen, zoals *circle* en *vector*2. Ook zijn er klassen voorzien om de uiteindelijke oplossing te visualiseren, tussen stappen te visualiseren en automatisch verschillende tests uit te voeren.

Een *problem* of probleem is een lijst van cirkels. Deze hebben nog geen positie, en worden gesorteerd van groot naar klein. Dit is wat de *solver* als input krijgt.

Een *solution* of oplossing is een lijst van cirkels met hun positie. Dit kan een tussen oplossing zijn, waar nog niet voor alle cirkels uit een probleem een positie gevonden is. Ook geeft dit geen garanties van correctheid, er kan dus bijvoorbeeld overlap zijn, maar voorziet functionaliteit om dit na te gaan. Dit is wat de solver als output geeft. Een correcte solver geeft natuurlijk wel altijd goede oplossingen.

Een *solver* of oplosser is het object dat een packing zoekt voor een gegeven probleem. Dit is dus het belangrijkste deel van de code, en hier is de nieuwe heuristiek geïmplementeerd. De best-fit solver, zoals beschreven in deze thesis, doet dit stap voor stap. In elke stap wordt er één cirkel geplaats op zijn finale positie, dit aan de hand van enkele keuzes die verder in dit hoofdstuk toegelicht zullen worden.

3.3 Structuur van de solver

Zoals hierboven gezegd is de solver het hart van de implementatie. Deze maakt effectief een packing voor een gegeven probleem. De solver bevat een lijst van *holes* en de *shell*. Het bevat ook een lijst van de nog te plaatsen cirkels, en een tussen-oplossing met de cirkels die reeds een plaats gekregen hebben. Ook heeft hij interne omcirkel voor deze oplossing. Een oplossing kan zelf ook een omcirkel berekenen, maar de solver gebruikt een interne omcirkel die enkel vernieuwd wordt als het nodig is. Bovendien heeft de solver extra informatie die de oplossing niet heeft, waardoor deze omcirkel efficiënter berekend kan worden. Zie **??** voor meer uitleg hierover.

De best-fit solver uit deze thesis kan stap voor stap de oplossing genereren. Dit zorgt er voor dat tussentijdse stappen gevisualiseerd kunnen worden. Het is dus niet nodig een packing volledig te maken, het kan zeer nuttig zijn tussentijdse packings te zien, zeker bij het debuggen of implementeren van nieuwe functionaliteit.

// TODO Voeg code toe?

3.4 Initialisatie

Zoals eerder gezegd bouwt het algoritme steeds verder op een packing uit de vorige stap. Hierdoor is het dus nodig om een initiële packing te maken van een aantal cirkels waarop de volgende stappen kunnen verder bouwen. Deze initiële packing is eenvoudigweg een packing van de drie grootste cirkels in het probleem. Deze drie cirkels worden zo geplaatst dat ze alle drie aan elkaar raken, zoals getoond in figuur 3.1. De licht blauwe cirkels tonen de drie eerste-geplaatste cirkels. Meer uitleg over hoe u deze figuur kan interpreteren kan u vinden in hoofdstuk 2

// TODO Interessant om deze code toe te voegen?

```
1    private void packFirstThree() {
2        //Initially place the two biggest circles next to eachother
3        Circle first = circlesToPack.get(0);
```

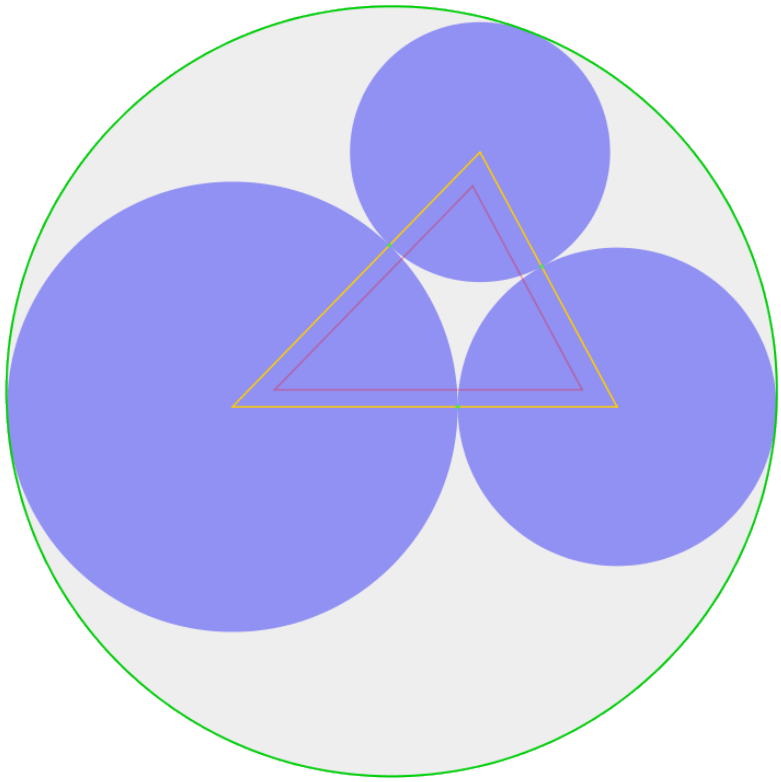


Figure 3.1: Voorbeeld van initiële packing

```
4           Circle second = circlesToPack.get(1);
5
6           Vector2 firstPos = new Vector2(0, 0);
7           Vector2 secondPos = new Vector2(first.getRadius() + second.getRadius(),
                0);
8
9           Location firstLoc = new Location(firstPos, first);
10          Location secondLoc = new Location(secondPos, second);
11
12          getSolution().add(firstLoc);
13          getSolution().add(secondLoc);
14
15          // Place the third biggest circle on top of the first two (assuming
                they are positioned clockwise)
16          Circle third = circlesToPack.get(2);
17          Vector2 thirdPos = Helpers.getMountPositionFor(third, firstLoc,
                secondLoc);
18          Location thirdLoc = new Location(thirdPos, third);
19          getSolution().add(thirdLoc);
20
21          circlesToPack.remove(first);
22          circlesToPack.remove(second);
23          circlesToPack.remove(third);
24
25          // Create first hole
```

```
26          holes.add(new NHole(firstLoc, secondLoc, thirdLoc));
27          // Create the initial shell
28          // IMPORTANT: must be clock-wise
29          shell.add(firstLoc);
30          shell.add(thirdLoc);
31          shell.add(secondLoc);
32
33          enclosingCircle =
                Location.calculateEnclosingCircle(Arrays.asList(firstLoc,
                secondLoc, thirdLoc));
34      }
```

// TODO Licht de code toe

3.5 Holes

Het eerste van de twee belangrijkste concepten van het algoritme is *holes*. Dit zijn plaatsen tussen andere, reeds geplaatste, cirkels, maar potentieel nog een cirkel tussen kan passen.

3.6 Shell

Hoofdstuk 4

Resultaten

Hoofdstuk 5

Opmerkingen en verder werk

Hoofdstuk 6

Conclusie

Bibliography

[1] Hakim Akeb and Yu Li. A basic heuristic for packing equal circles into a circular container. *Comput. Oper. Res*, 33:2125–2142, 2006.

[2] I Al-Mudahka, Mhand Hifi, and Rym M'Hallah. Packing circles in the smallest circle: an adaptive hybrid algorithm. *Journal of the Operational Research Society*, 62(11):1917–1930, 2011.

[3] P. Bollansée. Circle packing. `https://github.com/circle-packing/best-fit`.

[4] Edmund K Burke, Graham Kendall, and Glenn Whitwell. A new placement heuristic for the orthogonal stock-cutting problem. *Operations Research*, 52(4):655–671, 2004.

[5] Andrea Grosso, ARMJU Jamali, Marco Locatelli, and Fabio Schoen. Solving the problem of packing equal and unequal circles in a circular container. *Journal of Global Optimization*, 47(1):63–81, 2010.

[6] E. Specht. Packomania. `http://www.packomania.com/`. Accessed: 2016-05-23.

AFDELING
Straat nr bus 0000
3000 LEUVEN, BELGIË
tel. + 32 16 00 00 00
fax + 32 16 00 00 00
www.kuleuven.be