



Een nieuwe constructieve heuristiek voor het *circle-packing* probleem

Gebaseerd op een *best-fit* methodiek

Pablo BOLLANSÉE

Promotor:
Prof. dr. Patrick De Causmaecker
KU Leuven, Departement
Computerwetenschappen,
CODeS@KULAK

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

Academiejaar 2015-2016

© Copyright by KU Leuven

Zonder voorafgaande schriftelijke toestemming van zowel de promotor(en) als de auteur(s) is overnemen, kopiëren, gebruiken of realiseren van deze uitgave of gedeelten ervan verboden. Voor aanvragen tot of informatie i.v.m. het overnemen en/of gebruik en/of realisatie van gedeelten uit deze publicatie, wendt u tot de KU Leuven, Faculteit Wetenschappen, Geel Huis, Kasteelpark Arenberg 11 bus 2100, 3001 Leuven (Heverlee), Telefoon +32 16 32 14 01.

Voorafgaande schriftelijke toestemming van de promotor(en) is eveneens vereist voor het aanwenden van de in dit afstudeerwerk beschreven (originele) methoden, producten, schakelingen en programma's voor industrieel of commercieel nut en voor de inzending van deze publicatie ter deelname aan wetenschappelijke prijzen of wedstrijden.

Voorwoord

Het *circle-packing* probleem bestaat er uit 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 stel ik een nieuwe heuristiek voor het plaatsen van cirkels in een cirkel voor, gebaseerd op het *best-fit*-principe. Het is een constructieve heuristiek waarin stapsgewijs telkens de best-passende cirkel geplaatst zal worden. Alle broncode is beschikbaar op GitHub [3]

Hoewel dit probleem wiskundig relatief eenvoudig voor te stellen is, is het uitwerken van een computationeel exacte oplossing wél een uitdaging. Bestaande implementaties zijn niet tijds-geoptimaliseerd, en vragen dus veel berekeningstijd. In deze thesis stel ik een nieuwe heuristiek voor die het mogelijk maakt zéér snel oplossingen te genereren.

Dit werk werd gedaan als masterproef voor de *Master of Science in de toegepaste informatica* te KU Leuven onder de CODES onderzoeksgroep. Het hield 18 studiepunten in van deze éénjarige master.

Ik wil hierbij Patrick De Causmaeker danken voor alle hulp en ondersteuning bij het ontwerpen van deze heuristiek en verwezenlijken van dit werk. Ook wil ik Jim Bollansée, Marie Julia Bollansée en Pieter Van de Walle danken voor hun hulp bij het schrijven van deze tekst.

Abstract

Het *circle-packing* probleem bestaat er uit 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 stel ik een nieuwe *best-fit* gebaseerde heuristiek voor het plaatsen van cirkels in een cirkel voor. Het is een constructieve heuristiek waarin stapsgewijs telkens de *best-passende* cirkel geplaatst zal worden.

Wiskundig 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 vragen zeer veel tijd om het te berekenen. In deze thesis stel ik een nieuwe heuristiek voor die het mogelijk maakt zéér snel oplossingen te genereren.

Oplossingen gegenereerd door het algoritme uit deze thesis hebben een omschreven cirkel die gemiddeld minder dan 6% groter is in vergelijking met de best gekende oplossingen. Dit is een zeer aanvaardbare uitbreiding van de omschreven cirkel, toch als je in rekening brengt dat deze oplossingen veel sneller bekomen zijn. Bestaande pogingen vragen uren en soms dagen tijd om slechts tientallen cirkels te plaatsen. De heuristiek voorgesteld in deze thesis lost het probleem op in slechts enkele milliseconden.

Daarbovenop maakt deze heuristiek het mogelijk om nieuwe problemen op te lossen. Zo zouden problemen met duizenden cirkels met voorgaande methoden onrealistisch veel tijd vragen, tientallen cirkels vragen al uren of dagen tijd. Dit soort problemen kunnen wel opgelost worden met deze nieuwe heuristiek.

Abstract (English)

The *circle-packing* problem consists of packing a number of circles, with given radii, in a container that is as small as possible. The shape of this container can differ, but is most often triangular, rectangular or circular. In this thesis I propose a new *best-fit* based heuristic to solve the problem of packing circles in a circle. It's a new constructive heuristic that builds a solution step-by-step by iteratively placing the circle that *fits best*.

Mathematically this is a relatively simple problem to model, but computationally it is very complex to solve. Existing methods to solve this problem are very time consuming. In this thesis I propose a new heuristic that can solve the problem in just a few milliseconds.

Solutions generated by the algorithm proposed in this thesis will result in a slight increase of radius of the circumscribed circle. Compared to the best-known solutions this method results in a radius that is on average less then 6% larger. This is a very acceptable increase when considering that the results are computed much faster. Existing methods require hours or days to compute a solution for just tens of circles. The heuristic in this thesis will give a solution in just milliseconds for the same problems.

On top of that this new heuristic allows us to solve new problems. Problems with thousands of circles would take an unrealistic amount of time for older algorithms. However for this new heuristic these problems aren't out of scope.

Inhoudsopgave

| | |
|--|-----------|
| Voorwoord | i |
| Abstract | iii |
| Abstract (English) | v |
| Lijst van figuren | ix |
| Lijst van tabellen | xi |
| 1 Inleiding | 1 |
| 2 Definities en termen | 5 |
| 3 Handleiding voor het lezen van de visualisaties | 7 |
| 4 Algoritme | 9 |
| 4.1 Basis-idee | 9 |
| 4.2 Structuur van de implementatie | 10 |
| 4.3 Structuur van de <i>solver</i> | 11 |
| 4.4 Initialisatie | 12 |
| 4.5 Een cirkel tegen twee andere cirkels plaatsen | 14 |
| 4.6 <i>Holes</i> | 15 |
| 4.6.1 Grootste cirkel zoeken die past in een <i>hole</i> | 16 |
| 4.6.2 Bepalen of een cirkel past in een <i>hole</i> | 19 |
| 4.7 <i>Shell</i> | 20 |
| 4.7.1 Efficiënter de omcirkel berekenen gebaseerd op de <i>shell</i> | 21 |
| 4.7.2 Bepalen of een cirkel past op de <i>shell</i> | 23 |
| 4.7.3 Een cirkel plaatsen op de <i>shell</i> | 24 |
| 4.8 Conclusie | 26 |
| 5 Bedenkingen bij implementatie | 27 |
| 5.1 Precisie | 27 |
| 5.2 Veronderstellingen | 27 |
| 6 Resultaten | 29 |
| 6.1 Packomania vergelijking | 30 |
| 6.1.1 Cirkels met gelijke grootte | 30 |
| 6.1.2 Packomania Benchmark | 33 |

| | | |
|----------|--|-----------|
| 6.1.3 | Packomania Machten | 34 |
| 6.2 | Grotere aantallen cirkels | 36 |
| 6.3 | Conclusie | 36 |
| 7 | Verder werk | 39 |
| 7.1 | Constante-tijd omschreven cirkel | 39 |
| 7.2 | Best-Fit | 39 |
| 8 | Conclusie | 41 |
| A | Packomania Benchmark Verdelingen | 45 |
| B | Packomania Vergelijking Tabellen | 47 |
| B.1 | Cirkels met gelijke grootte | 47 |
| B.2 | Packomania Macht problemen | 49 |
| B.3 | Packomania Benchmark problemen | 62 |
| B.4 | Grotere aantallen cirkels | 63 |
| C | Extra <i>packing</i> figuren | 67 |

Lijst van figuren

| | | |
|------|--|----|
| 3.1 | Voorbeeld visualisatie met drie duidelijke <i>holes</i> | 8 |
| 3.2 | Voorbeeld visualisatie met grote <i>shell</i> | 8 |
| 4.1 | Voorbeeld van initiële <i>packing</i> | 12 |
| 4.2 | Verkregen intersectie punt van <i>getMountPositionFor</i> | 15 |
| 4.3 | <i>Packing</i> voor het opvullen van een <i>hole</i> | 16 |
| 4.4 | <i>Packing</i> na het opvullen van een <i>hole</i> | 17 |
| 4.5 | <i>Packing</i> na het opvullen van een tweede <i>hole</i> | 17 |
| 4.6 | <i>Packing</i> als het opvullen van een tweede <i>hole</i> mislukt | 18 |
| 4.7 | Het plaatsen van de grootste cirkel op de <i>shell</i> | 22 |
| 4.8 | Het plaatsen van een kleinere cirkel op de <i>shell</i> | 22 |
| 4.9 | <i>Shell</i> aanpassen als geen enkele cirkel past | 23 |
| 4.10 | Een cirkel veroorzaakt een tegen-de-klok <i>shell</i> | 25 |
| 4.11 | Mogelijke fout indien de <i>shell</i> niet met de klok mee gesorteerd is | 25 |
| 6.1 | Packomania verdelingen | 30 |
| 6.2 | Vergelijking van <i>packings</i> van cirkels met gelijke grootte | 31 |
| 6.3 | <i>Packing</i> voor 500 cirkels met gelijke grootte | 32 |
| 6.4 | Vergelijking van Packomania Benchmark problemen | 33 |
| 6.5 | Vergelijking van Packomania Macht problemen | 34 |
| 6.6 | <i>Packing</i> voor 1000 cirkels met verdeling $r_i = i^{-1/2}$ | 35 |
| C.1 | <i>Packing</i> voor 50 cirkels met gelijke grootte | 68 |
| C.2 | <i>Packing</i> voor 500 cirkels met gelijke grootte | 68 |
| C.3 | <i>Packing</i> voor 2000 cirkels met gelijke grootte | 69 |
| C.4 | <i>Packing</i> voor 5000 cirkels met gelijke grootte | 69 |
| C.5 | <i>Packing</i> voor 100 cirkels met verdeling $r_i = i^{1/2}$ | 70 |
| C.6 | <i>Packing</i> voor 100 cirkels met verdeling $r_i = i^{-1/2}$ | 70 |
| C.7 | <i>Packing</i> voor 100 cirkels met verdeling $r_i = i^{-2/3}$ | 71 |
| C.8 | <i>Packing</i> voor 100 cirkels met verdeling $r_i = i^{-1/5}$ | 71 |
| C.9 | <i>Packing</i> voor 1000 cirkels met verdeling $r_i = i^{1/2}$ | 72 |
| C.10 | <i>Packing</i> voor 1000 cirkels met verdeling $r_i = i^{-1/2}$ | 72 |
| C.11 | <i>Packing</i> voor 1000 cirkels met verdeling $r_i = i^{-2/3}$ | 73 |
| C.12 | <i>Packing</i> voor 1000 cirkels met verdeling $r_i = i^{-1/5}$ | 73 |
| C.13 | <i>Packing</i> voor 5000 cirkels met verdeling $r_i = i^{1/2}$ | 74 |
| C.14 | <i>Packing</i> voor 5000 cirkels met verdeling $r_i = i^{-1/2}$ | 74 |
| C.15 | <i>Packing</i> voor 5000 cirkels met verdeling $r_i = i^{-2/3}$ | 75 |
| C.16 | <i>Packing</i> voor 5000 cirkels met verdeling $r_i = i^{-1/5}$ | 75 |

Lijst van tabellen

| | | |
|------|---|----|
| 6.1 | Packomania Machten gemiddelde vergroting | 35 |
| A.1 | Packomania Benchmark Verdelingen | 45 |
| A.1 | Packomania Benchmark Verdelingen | 46 |
| B.1 | Packomania $r_i = 1$ | 47 |
| B.2 | Packomania $r_i = i$ | 50 |
| B.3 | Packomania $r_i = i^{1/2}$ | 54 |
| B.4 | Packomania $r_i = i^{-1/5}$ | 57 |
| B.5 | Packomania $r_i = i^{-1/2}$ | 58 |
| B.6 | Packomania $r_i = i^{-2/3}$ | 60 |
| B.7 | Packomania Benchmark Instances | 62 |
| B.8 | Grotere aantallen gelijke cirkels | 63 |
| B.9 | Grotere aantallen cirkels: $r_i = i^{1/2}$ | 64 |
| B.10 | Grotere aantallen cirkels: $r_i = i^{-1/2}$ | 64 |
| B.11 | Grotere aantallen cirkels: $r_i = i^{-2/3}$ | 65 |
| B.12 | Grotere aantallen cirkels: $r_i = i^{-1/5}$ | 66 |

Hoofdstuk 1

Inleiding

In deze thesis stel ik een nieuwe heuristiek voor, gebaseerd op het *best-fit*-principe, als oplossing voor het *circle-packing* probleem. De heuristiek is specifiek ontworpen voor het sub-probleem van het plaatsen van cirkels in een cirkel. Het *circle-packing* probleem bestaat uit het plaatsen van n cirkels in een zo klein mogelijke cirkelvormige *container*. Het is de bedoeling om voor de gegeven cirkels de coördinaten van de middelpunten te vinden zodat de cirkels niet overlappen en de radius van de omschreven cirkel zo klein mogelijk is.

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

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

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

Hierin is r_i de radius, en (x_i, y_i) de coördinaten van het centrum van cirkel i . Hierbij wordt verondersteld dat de omschreven cirkel het nulpunt als middelpunt heeft. De eerste formule verzekert dat de cirkels in de omschreven cirkel liggen, en de tweede dat ze elkaar niet overlappen. Wanneer alle cirkels dezelfde grootte hebben, wordt meestal r_i gelijk aan 1 genomen. Het *circle-packing* probleem voor andere containers heeft gelijkaardige, relatief eenvoudige, wiskundige omschrijvingen.

Hoewel dit wiskundig eenvoudig te omschrijven is, blijft het toch een zeer moeilijk probleem om exact op te lossen. Het is een NP-moeilijk probleem. Er is reeds veel onderzoek gebeurd naar het oplossen van het *circle-packing* probleem voor zowel cirkels van gelijke grootte, als voor cirkels van verschillende grootte.

In [7] en [15] probeert men vaste patronen te vinden die een optimale *packing* van cirkels met gelijke grootte geeft. In [8] en [19] worden fysisch geïnspireerde simulaties gebruikt om *packings* te bekomen.

In [6] worden verschillende meta-heuristieken, waaronder een genetisch algoritme, uitgeprobeerd en vergeleken. Zij ondervinden dat dit genetisch algoritme, alsook een quasi-random techniek, in vergelijking met de andere uitgeprobeerde meta-heuristieken, resul-

taten met de kleinste omschreven cirkel geven. In [10] en [11] worden respectievelijk een genetisch en een *simulated-annealing* algoritme voorgesteld.

Een recentere poging is het Monotonic Basin Hopping algoritme voorgesteld in [9]. Hierin beschrijven de onderzoekers dat er te veel lokale optima zijn voor een eenvoudige multi-start behandeling, en ze stellen als mogelijke oplossing een variant voor waarin ze op een slimme manier de beginpunten proberen te genereren. Meer recent zijn [2] (2011), [21] (2013) en [22] (2016). Zij gebruiken de combinatorische eigenschappen van *circle-packing* in combinatie met een zoekmethode zoals *tabu-search* en *iterated local search* om een goede volgorde te vinden waarin de cirkels geplaatst kunnen worden. In [2] plaatsen de onderzoekers zo één voor één elke cirkel en berekent een *non-linear optimizer* hiervoor telkens een lokaal extremum voor de *packing*.

Constructieve algoritmen voor het oplossen van *circle-packings* zijn veel minder onderzocht en gedocumenteerd. Een constructief algoritme is een algoritme dat, uitgaand van een zekere beginsituatie, stapsgewijs de oplossing uitbreidt tot deze volledig is. Eén van de weinige constructieve methoden wordt beschreven in [1], waarin een alternatieve vorm van het *circle-packing* probleem wordt opgelost. Bij deze variant ligt de grootte van de container vast, en er moeten zo veel mogelijk cirkels van gelijke grootte in deze container geplaatst worden. In [10] wordt er een aanpassing gedaan op de klassieke *bottom-left-first* heuristiek, voor het plaatsen van rechthoeken. Deze heuristiek plaats rechthoeken zo ver mogelijk in de linker benedenhoek. Dit is een zeer eenvoudige heuristiek die vaak goede resultaten geeft. De onderzoekers hebben deze dan aangepast om cirkels in een rechthoek te plaatsen.

Hoewel veel van deze oplossingen zeer goede *packings* maken, en regelmatig hun voorgangers verbeteren, vragen ze veel rekentijd en beperken ze zich tot een klein aantal cirkels. Voor slechts tientallen cirkels kan de nodige rekentijd oplopen tot tientallen uren. In deze thesis stel ik een nieuwe constructieve heuristiek voor om het *circle-packing* probleem op te lossen. Deze nieuwe heuristiek laat toe om *packings* te maken in een fractie van de tijd die eerdere algoritmen daarvoor nodig hebben. Ook is het mogelijk om veel grotere aantallen cirkels te plaatsen. De omcirkel van *packings* verkregen met deze nieuwe heuristiek is echter iets groter dan deze verkregen in eerder vernoemd onderzoek. Deze uitbreiding van de omcirkel blijft echter beperkt en laat toe de *packings* in slechts enkele ogenblikken te maken, waar andere oplossingen uren rekentijd vragen.

De heuristiek voorgesteld in deze thesis is een *best-fit* heuristiek gebaseerd op een oplossing voor het *orthogonal stock-cutting problem* voorgesteld in [5]. In dit probleem worden balken geplaatst in een container met een vaste breedte, zodat de hoogte zo klein mogelijk blijft. Deze heuristiek gebruikt het verschil tussen de breedte van een balk en een gat in de *packing* als maatstaf voor de beste positie. Hun heuristiek kiest dus de volgende balk om te plaatsen uit een lijst, en plaatst deze op de beste positie. Op een gelijkaardige manier kiest mijn algoritme de volgende cirkel die best past in de huidige *packing*. Deze aanpak verschilt van het plaatsen van cirkels in een vooraf bepaalde volgorde zoals in [9] en [2].

In hoofdstuk 4 bespreek ik hoe mijn heuristiek opgebouwd is. Ik beschrijf de twee basisconcepten voor mijn *best-fit* heuristiek, de begrippen *holes* en *shell*, in detail. Ik bespreek hoe deze concepten werken, en op welke manier gekozen wordt welke cirkel best past in de *packing*. Ik illustreer deze concepten verder door naar de implementatie te verwijzen. In hoofdstuk 6 worden de verkregen resultaten besproken. Hier vergelijk ik de *packings* met de best gekende resultaten zoals gerapporteerd op de Packomania website

([17]). Deze website verzamelt de best gekende oplossingen voor verschillende *circle-packing* problemen, en bevat informatie over hun effectiviteit. Ik vergelijk de omtrek van de verkregen omcirkel en de nodige tijd om de *packing* te berekenen van mijn eigen heuristiek met de beste oplossingen op de Packomania website. Ook toon ik resultaten voor *packings* voor veel meer cirkels dan getoond op de Packomania website. In hoofdstuk 5 bespreek ik enkele bedenkingen die ik heb bij mijn onderzoek en de implementatie van de onderzochte heuristiek. In hoofdstuk 7 bespreek ik mogelijke verbeteringen en ideeën voor verder onderzoek. In hoofdstuk 3 wordt kort verduidelijkt hoe de visualisaties, die doorheen deze thesis gebruikt worden, dienen geïnterpreteerd te worden.

In het eerstvolgende hoofdstuk (hoofdstuk 2) worden echter eerst enkele termen toegelicht die gebruikt worden in de verdere tekst, sommigen uniek binnen dit onderzoek.

Hoofdstuk 2

Definities en termen

Cirkel: In de literatuur rond *circle-packing* en in deze thesis wordt het woord *cirkel* of *circle* gebruikt, maar eigenlijk zou het correcter zijn om *schijf* te gebruiken. Dit geeft (onder andere) aan dat er een overlap tussen cirkels is wanneer een cirkel volledig omringd is door een andere. Voor twee cirkels met radii r_a, r_b en coördinaten $(x_a, y_a), (x_b, y_b)$ is er een overlap wanneer $r_a + r_b > \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$.

Packing: Vertaald als **plaatsing**. Een **packing** is een collectie cirkels met elk een toegewezen positie. Een *packing* is correct wanneer er geen overlap is tussen de cirkels in deze *packing*.

Hole: Eén van de concepten uniek aan deze heuristiek. Op een *high-level* niveau is een **hole** een plek tussen drie cirkels waarin een andere cirkel kan geplaatst worden. Dit concept wordt in detail uitgelegd in sectie 4.6.

Shell: Het tweede nieuwe concept in dit onderzoek. Een **shell** is, op *high-level* niveau, de collectie cirkels die aan de buitenkant van een *packing* liggen. Het gegeven wordt verder verduidelijkt in sectie 4.7.

Solver: Een programma dat, gegeven een aantal cirkels, een *packing* maakt voor deze cirkels.

Omschreven cirkel of **omcirkel:** De kleinst mogelijke cirkel (tenzij anders vermeld) waarvoor geldt dat alle cirkels in de *packing* volledig in deze cirkel liggen. Een *packing* is beter dan een andere *packing* wanneer deze een kleinere omschreven cirkel heeft.

Binair zoeken: Een klassieke zoektechniek die werkt op gesorteerde lijsten. Hierin wordt de lijst steeds in twee verdeeld door telkens het middelste element te testen. Er wordt dan verder gezocht in één van de twee delen, afhankelijk van het geteste element kleiner of groter is dan de gezochte waarde. Dit geeft een logaritmische complexiteit.

Hoofdstuk 3

Handleiding voor het lezen van de visualisaties

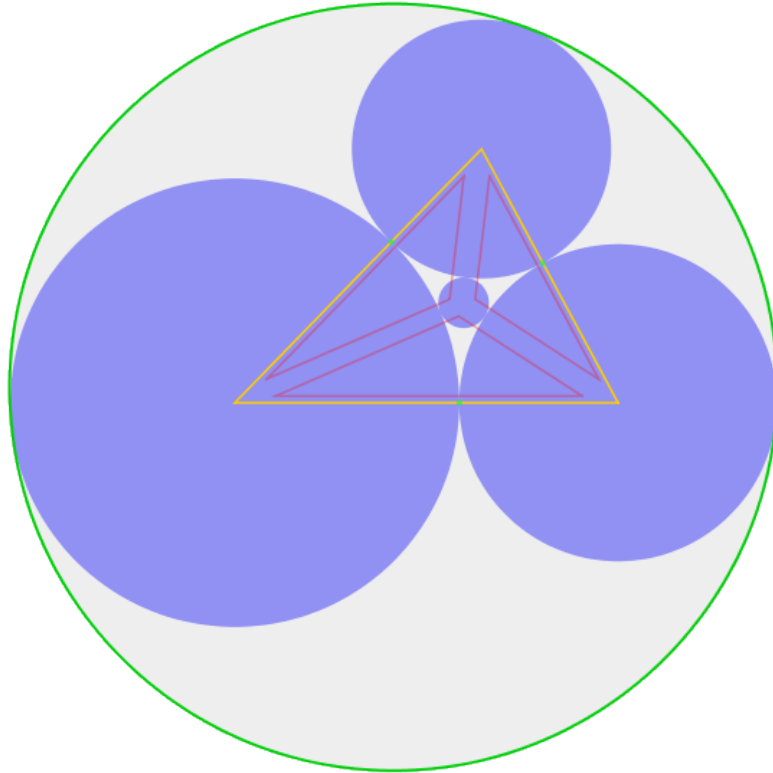
Doorheen deze thesis zal ik gebruik maken van visualisaties die gegenereerd zijn door de implementatie (in Java) van het algoritme, ter illustratie van de besproken en onderzochte concepten. Twee voorbeelden van zulke visualisaties zijn figuur 3.1 en figuur 3.2.

Deze figuren kan u op de volgende manier interpreteren:

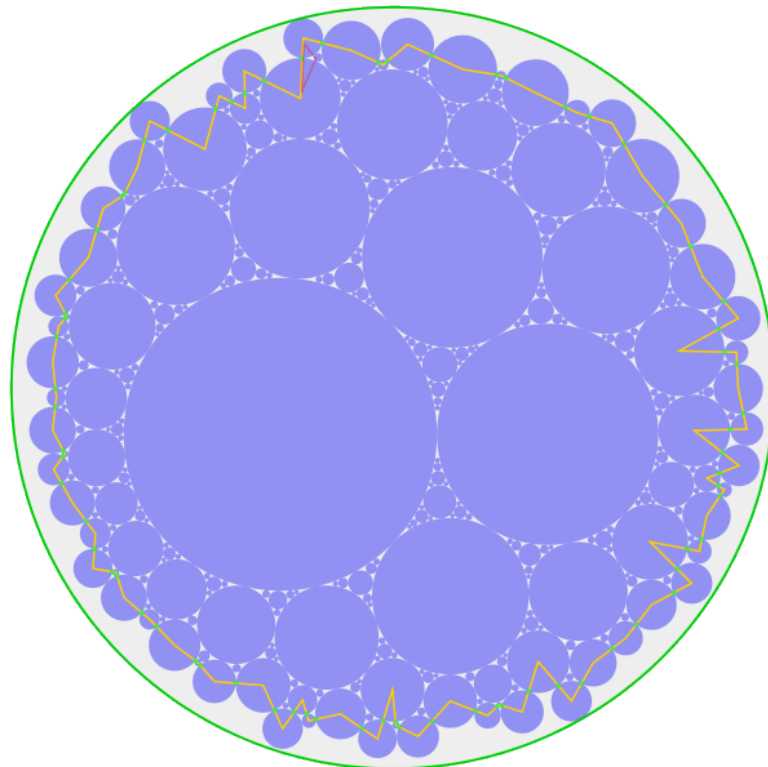
- De **reeds geplaatste cirkels** worden getoond als **blauwe cirkels**.
- De *shell* is een **gele lijn** aan de buitenrand van de *packing*. Deze verbindt de middelpunten van de cirkels op de *shell*.
- *Holes* worden getoond als **rode driehoeken**. De hoekpunten liggen dichtbij de middelpunten van de drie cirkels die de *hole* definiëren.
- De **omschreven cirkel** van de huidige *packing* wordt getoond als een **groene cirkel**.

Merk op dat deze figuren de oorsprong niet als midden hebben, maar steeds gecentreerd zijn rond het midden van de omschreven cirkel van de packing die ze tonen.

Op figuur 3.1 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 3.2 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.



Figuur 3.1: Voorbeeld visualisatie met drie duidelijke *holes*



Figuur 3.2: Voorbeeld visualisatie met grote *shell*

Hoofdstuk 4

Algoritme

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

Om het algoritme zeer snel te maken worden enkele veronderstellingen gemaakt omtrent de nodige overlap-checks bij elke stap. Deze veronderstellingen zijn niet theoretisch bewezen, maar wel empirisch getest. Voor de meeste verdelingen van cirkels lijken deze veronderstellingen goed stand te houden, maar er zijn nog enkele *edge-cases* waarin er toch nog fouten gebeuren. Deze problemen worden besproken in hoofdstuk 5 en de snelheid en kwaliteit van de oplossingen wordt verder besproken in hoofdstuk 6.

4.1 Basis-idee

Het basis-idee van de heuristiek is om stelselmatig een *packing* op te bouwen, door telkens cirkels uit een reeks gegeven cirkels te zoeken die het best passen. Bij elke stap wordt telkens eerst een positie gekozen om een cirkel te plaatsen in een *hole*, of op de *shell* (zie voor verdere informatie respectievelijk sectie 4.6 en sectie 4.7). Hier wordt dan de best-passende cirkel geplaatst, gekozen uit de lijst cirkels die nog geen positie gekregen hebben. Eenmaal een cirkel geplaatst is wordt deze nooit meer verplaatst. Dit laat toe om deze *holes* en *shell* efficiënt op te bouwen en gebruiken. Als de cirkels wel nog verplaatst konden worden zou het veel moeilijker, en rekenintensiever, zijn om deze structuren op te bouwen. Dan zouden in elke stap alle cirkels met elkaar vergeleken moeten worden om, bijvoorbeeld, te vinden waar nog *holes* zitten. Zie hoofdstuk 6 voor verdere bespreking van de efficiëntie van mijn algoritme.

Het algoritme bouwt dus cirkel per cirkel een *packing* op. Dit gebeurt door in elke stap een positie te kiezen, en hierin een cirkel proberen te plaatsen. Eerst worden alle *holes* uitgeprobeerd, in de volgorde waarin ze aangemaakt zijn, dan wordt er gekeken naar een plek op de *shell*. 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 4.6 en sectie 4.7. 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 op de gekozen locatie. Het plaatsen van een nieuwe cirkel geeft op zijn beurt ook aanleiding tot aanpassen van

de beschikbare *holes* en/of de *shell*. Hierdoor wordt er een nieuwe tussentijdse *packing* gemaakt. Deze wordt dan doorgegeven naar de volgende stap, waarin het algoritme opnieuw zal proberen een cirkel te plaatsen. Op deze manier wordt een volledige *packing* opgebouwd voor alle cirkels.

4.2 Structuur van de implementatie

De implementatie van het algoritme bevat enkele belangrijke (programmeer-)klassen die regelmatig zullen terugkomen in de verdere uitleg, vooral in codefragmenten:

- Cirkel (*Circle*)
- Vector2
- Locatie (*Location*)
- Probleem (*Problem*)
- Oplossing (*Solution*)
- Oplosser (*Solver*)
- Gat (*Hole*)
- Schil (*Shell*)

Een *circle* is voor de hand liggend. Deze heeft een radius, maar echter geen positie. *Vector2* is een 2D positie. Deze bevat een x en y coördinaat.

Een *location* of locatie is de combinatie van een cirkel met zijn positie. Deze bevat dus een referentie naar een *circle* en een *vector2*.

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 tussenoplossing zijn, waarbij nog niet voor alle cirkels uit een gegeven probleem een positie bestaat. Ook geeft een *solution* geen garanties van correctheid, er kan dus bijvoorbeeld overlap zijn. De *solution*-klasse voorziet echter 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 hierin 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 geplaatst op zijn finale positie, dit aan de hand van enkele keuzes die verder in dit hoofdstuk toegelicht zullen worden.

Hole en *shell* worden verder uitgelegd in respectievelijk sectie 4.6 en sectie 4.7.

4.3 Structuur van de *solver*

Zoals hierboven beschreven is de *solver* het hart van de implementatie. Deze klasse tracht een *packing* te genereren voor een gegeven probleem. De *solver* bevat een lijst van *holes* en de *shell*. Hij 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 een interne omcirkel voor deze oplossing. Een oplossing kan zelf ook een omcirkel berekenen, maar de *solver* gebruikt een interne omcirkel die enkel vernieuwd wordt wanneer het nodig is. Bovendien heeft de *solver* extra informatie die de oplossing niet heeft, waardoor deze omcirkel efficiënter berekend kan worden. Zie sectie 4.7 voor meer uitleg hierover.

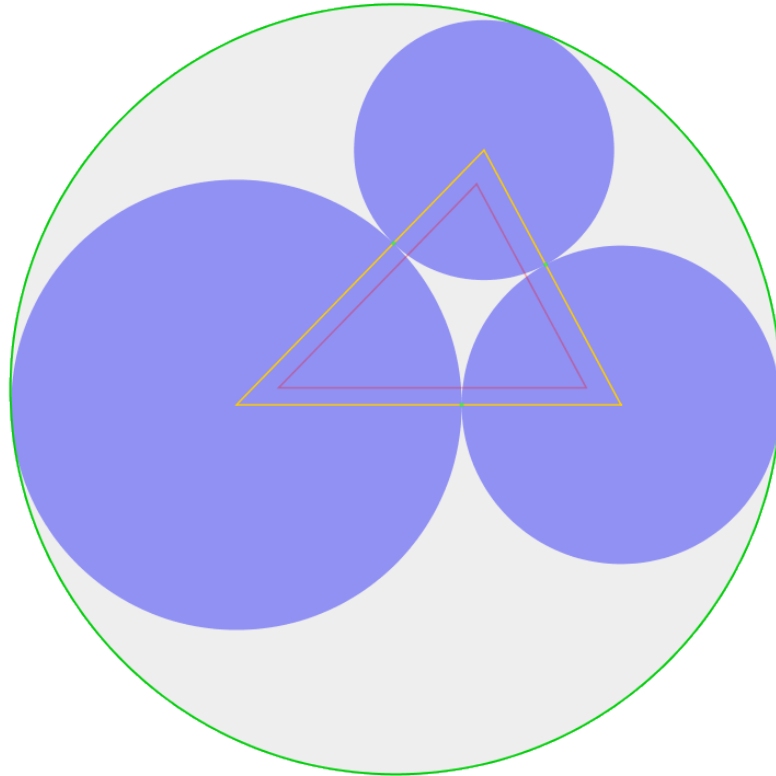
In de implementatie ziet de code van de *solver* er als volgt uit (vereenvoudigde versie):

```

1 List<Circle> circlesToPack;
2 Queue<Hole> holes;
3 List<Location> shell;
4 Location enclosingCircle;
5
6 Solution solution;
7
8 void solve() {
9     init();
10    packFirstThree();
11
12    while(!circlesToPack.isEmpty()) {
13        boolean ok = bestFitStep();
14        if (!ok) break;
15    }
16 }
17
18 boolean bestFitStep() {
19     if (circlesToPack.isEmpty()) {
20         return false;
21     }
22
23     if(!holes.isEmpty()) {
24         ...
25         // Probeer een cirkel in een hole te plaatsen
26         ...
27         return true;
28     }
29     else if (!shell.isEmpty()) {
30         ...
31         // Probeer een cirkel op de shell te plaatsen
32         ...
33         return true;
34     }
35 }

```

De *solver* bevat alle nodige informatie over de *shell* en de *holes*, alsook de cirkels die nog geplaatst moeten worden en de huidige tussen-oplossing (lijn 1 tot lijn 6). Om een



Figuur 4.1: Voorbeeld van initiële *packing*

probleem op te lossen wordt de `solve()` methode (lijn 8) aangeroepen. Deze initialiseert eerst alle nodige variabelen, doet dan de initiële *packing* (meer hierover in sectie 4.4) en voert dan *best-fit*-stappen uit tot een oplossing bereikt is (vanaf lijn 12).

De best-fit *solver* uit deze thesis kan stap voor stap de oplossing genereren en eveneens de tussentijdse oplossingen visualiseren. Het is dus niet nodig een *packing* volledig te maken en het kan zeer nuttig zijn tussentijdse oplossingen te zien, zeker bij het debuggen of implementeren van nieuwe functionaliteit.

4.4 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 verderbouwen. Deze initiële *packing* is de optimale *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 4.1. De blauwe cirkels tonen de drie cirkels die het eerst geplaatst zijn. Meer uitleg over interpreteren van deze figuur kan u vinden in hoofdstuk 3

Het exacte proces om deze initiële *packing* te bekomen wordt verduidelijkt aan de hand van code uit de implementatie:

```

1 private void packFirstThree() {
2     // Plaatst eerst de twee grootste cirkels naast elkaar
3     Circle first = circlesToPack.get(0);

```

```

4     Circle second = circlesToPack.get(1);
5
6     Vector2 firstPos = new Vector2(0, 0);
7     Vector2 secondPos = new Vector2(first.getRadius() + second.getRadius(),
8         0);
9
10    Location firstLoc = new Location(firstPos, first);
11    Location secondLoc = new Location(secondPos, second);
12    getSolution().add(firstLoc);
13    getSolution().add(secondLoc);
14
15    // Plaats de derde grootste cirkel bovenop de eerste twee
16    Circle third = circlesToPack.get(2);
17    Vector2 thirdPos = Helpers.getMountPositionFor(third, firstLoc,
18        secondLoc);
19    Location thirdLoc = new Location(thirdPos, third);
20    getSolution().add(thirdLoc);
21
22    circlesToPack.remove(first);
23    circlesToPack.remove(second);
24    circlesToPack.remove(third);
25
26    // Maak de eerste hole
27    holes.add(new NHole(firstLoc, secondLoc, thirdLoc));
28    // Maak de initiele shell
29    // BELANGRIJK: Deze moeten met-de-klok mee op de shell geplaatst worden
30    shell.add(firstLoc);
31    shell.add(thirdLoc);
32    shell.add(secondLoc);
33
34    enclosingCircle =
35        Location.calculateEnclosingCircle(Arrays.asList(firstLoc, secondLoc,
36            thirdLoc));
37 }

```

Eerst worden de twee grootste cirkels naast elkaar geplaatst. Vanaf lijn 3 tot lijn 7 worden eerst de twee grootste cirkels uit het probleem opgevraagd. De lijst *circlesToPack* is gesorteerd van groot naar klein, dus dit zijn de eerste twee cirkels in deze lijst. De eerste wordt in de oorsprong geplaatst, en de tweede er tegen op de horizontale as. Deze worden ook reeds aan de tussentijdse oplossing toegevoegd (vanaf lijn 11). Vervolgens wordt de positie berekend voor de derde cirkel aan de hand van een helper functie op lijn 16. Deze helper functie komt regelmatig terug, en wordt verduidelijkt in sectie 4.5.

In de initialisatie wordt ook de eerste *hole* gemaakt, gedefinieerd door de eerste drie cirkels. Dit gat wordt toegevoegd aan de lijst van *holes* in de *solver* op lijn 25. Ook wordt de *shell* aangemaakt, vanaf lijn 28. Deze wordt met de klok mee (gezien vanuit het centrum van de huidige *packing*) bijgehouden. Verdere uitleg hierover is te vinden in sectie 4.7.

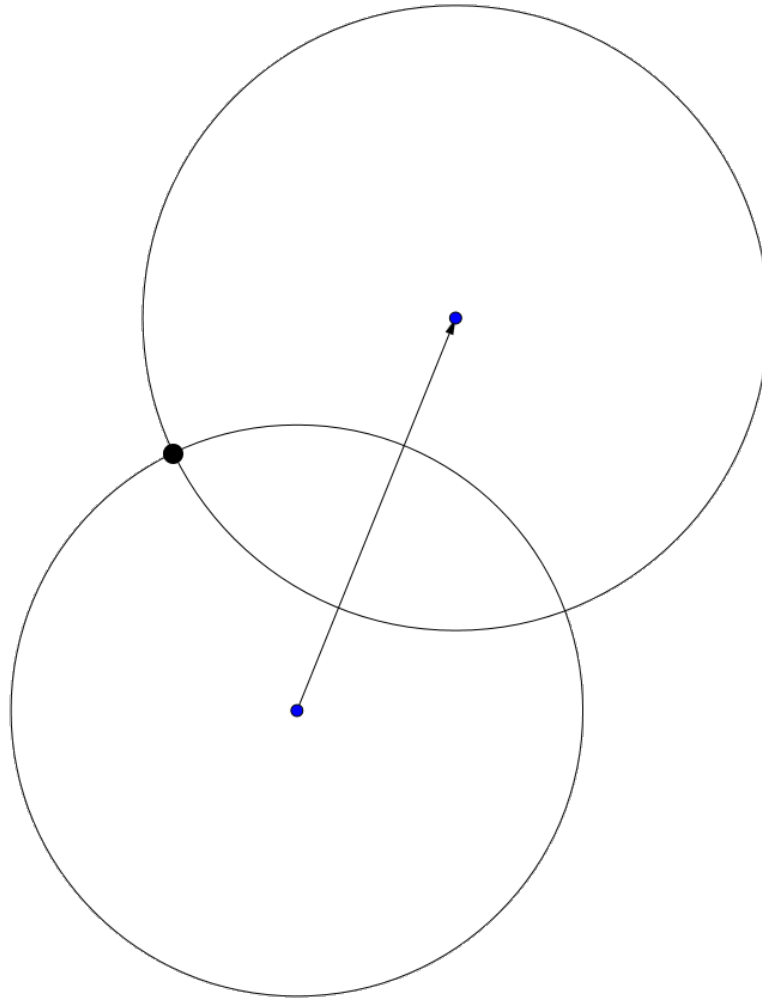
4.5 Een cirkel tegen twee andere cirkels plaatsen

In verschillende delen van de implementatie is het nodig om een cirkel c_i , met radius r_i , tegen twee andere cirkels te plaatsen. Deze twee cirkels noemen we c_{g1}, c_{g2} , en hun radii r_{g1}, r_{g2} . Het punt waarop deze cirkel moet staan om beide andere cirkels te raken wordt bepaald door een eenvoudige cirkel-cirkel intersectie, tussen twee cirkels met hun middelpunt gelijk aan het middelpunt van de cirkels c_{g1} en c_{g2} en als radii $r_{g1} + r_i$ en $r_{g2} + r_i$:

```

1 Vector2 getMountPositionFor(Circle cir, Location first, Location second) {
2     double x0 = first.getPosition().getX();
3     double y0 = first.getPosition().getY();
4     double r0 = first.getCircle().getRadius() + cir.getRadius();
5
6     double x1 = second.getPosition().getX();
7     double y1 = second.getPosition().getY();
8     double r1 = second.getCircle().getRadius() + cir.getRadius();
9
10    // dx en dy zijn de verticale en horizontale afstand tussen de
11        cirkel-centra.
12    double dx = x1 - x0;
13    double dy = y1 - y0;
14
15    // Bepaal de afstand tussen de centra
16    //d = sqrt((dy*dy) + (dx*dx));
17    double d = Math.hypot(dx, dy);
18
19    // 'Punt 2' is het punt waar de lijn door de cirkel-intersectie punten de
20        lijn tussen de cirkel-centra kruist
21    // We berekenen hier de coördinaten x2 en y2 van dit punt
22
23    // Bepaal eerst de afstand van tussen Punt 2 en het centrum van de eerste
24        cirkel
25    double a = ((r0*r0) - (r1*r1) + (d*d)) / (2.0 * d);
26
27    // Bepaal dan de coördinaten van Punt 2.
28    double x2 = x0 + (dx * a/d);
29    double y2 = y0 + (dy * a/d);
30
31    // Bepaal nu de afstand van Punt 2 naar een van de intersectie-punten
32    // Het tweede intersectie-punt ligt even ver
33    double h = Math.sqrt((r0*r0) - (a*a));
34
35    // Zet deze afstand om naar een vector met de juiste richting
36    double rx = -dy * (h/d);
37    double ry = dx * (h/d);
38
39    // Bepaal een van de tweede intersectie punten
40    return new Vector2(x2 - rx, y2 - ry);
41 }

```

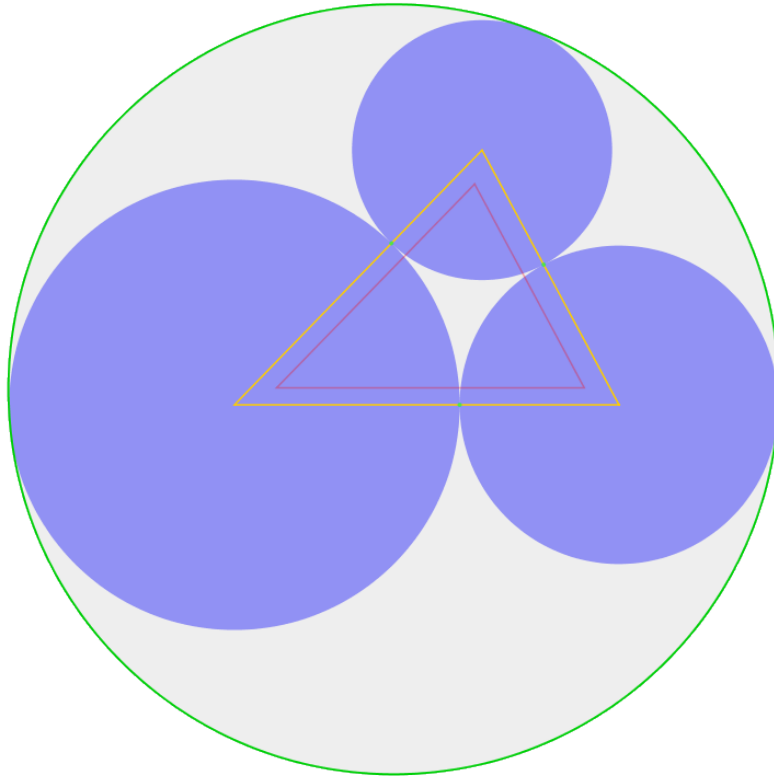
Figuur 4.2: Verkregen intersectie punt van *getMountPositionFor*(Gemaakt met web.geogebra.org)

In deze code wordt één van de intersectiepunten bepaald. Dit punt is het punt dat aan uw linker kant zou liggen indien u wandelt van het centrum van de eerste cirkel naar het centrum van de tweede cirkel. Dit is verduidelijkt in figuur 4.2, de onderste cirkel is de eerste, de bovenste cirkel de tweede. De pijl tussen deze cirkels geeft de *wandelrichting* aan.

Ik noem dit intersectiepunt het *negatieve punt*. Als op lijn 37 + gebruikt wordt in plaats van $-$ kan het tweede punt bekomen worden, het *positieve punt*. Het is ook mogelijk het andere intersectie punt te verkrijgen door de twee *location* parameters om te wisselen.

4.6 Holes

Het eerste van de twee belangrijkste concepten van de heuristiek is *holes* of gaten. Dit zijn plaatsen tussen andere, reeds geplaatste, cirkels waar potentieel nog een cirkel kan tussen passen. De heuristiek zal telkens eerst deze *holes* proberen op te vullen, alvorens



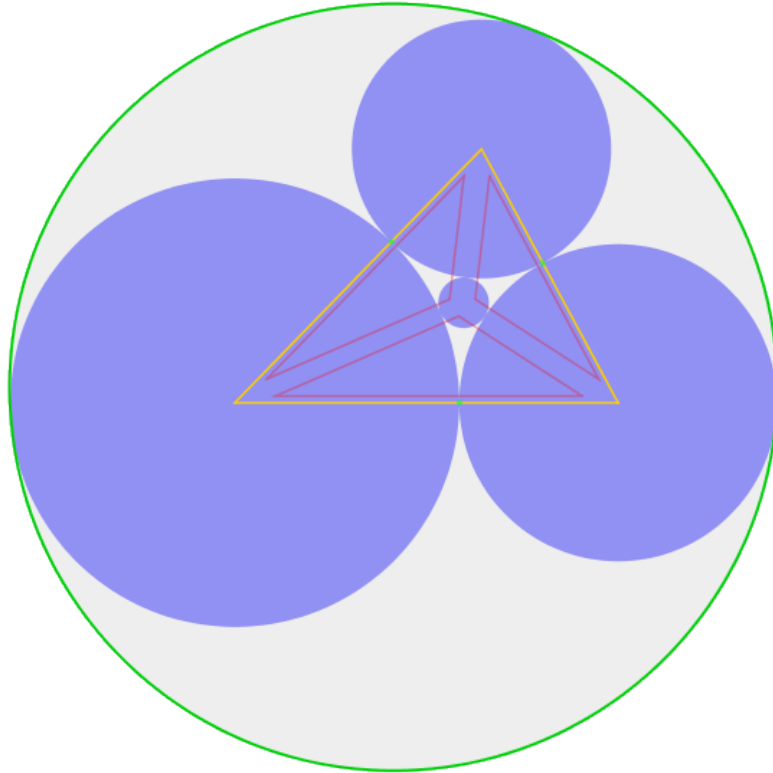
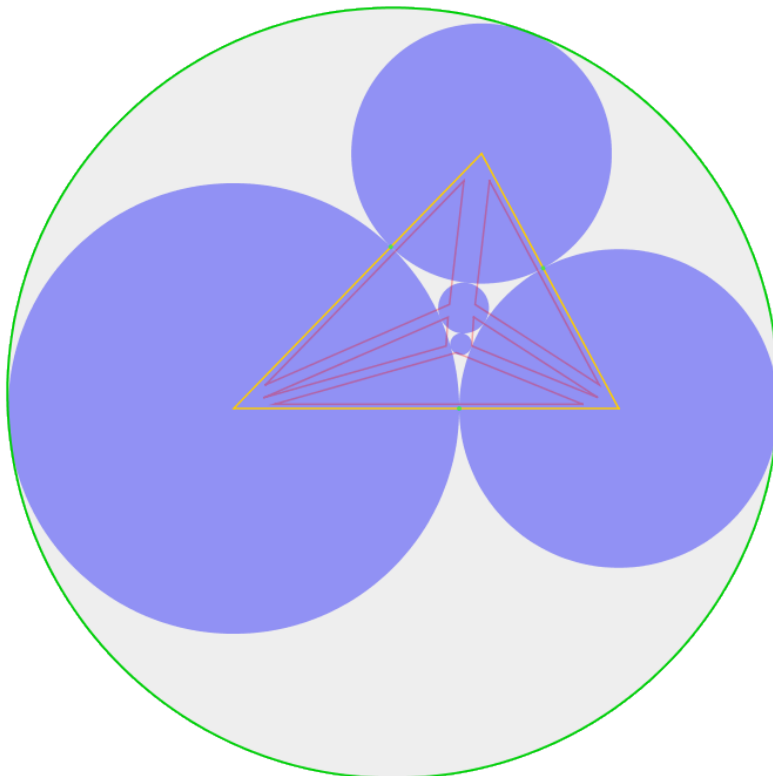
Figuur 4.3: *Packing* voor het opvullen van een *hole*

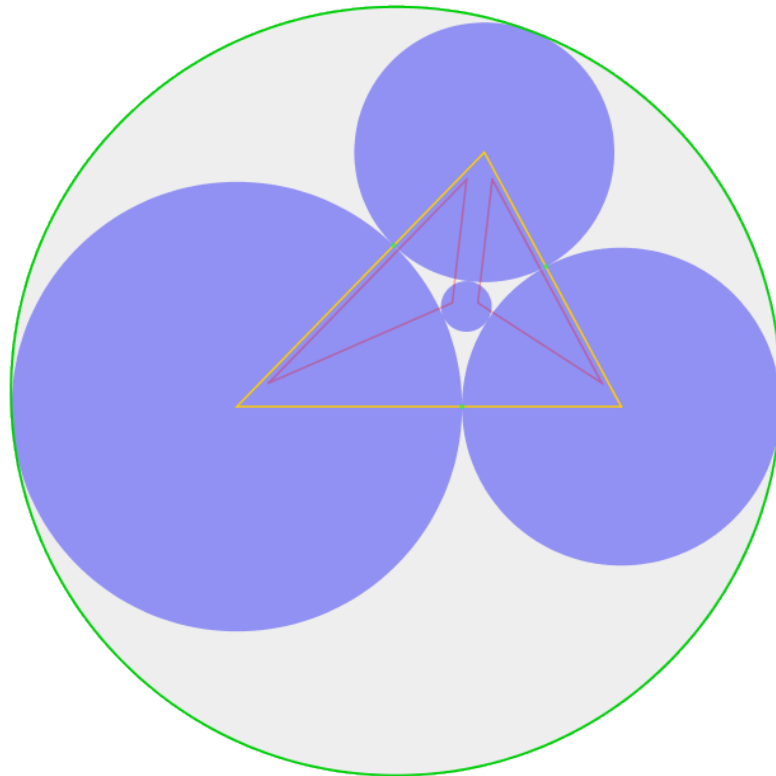
cirkels op de *shell* te plaatsen. Dit omdat het plaatsen van een cirkel in een *hole* nooit de omschreven cirkel kan vergroten.

Holes worden gedefinieerd door exact drie cirkels in de huidige *packing*. De *solver* houdt informatie bij voor elke *hole* waar mogelijk nog een cirkel in kan passen. Bij elke stap van de *solver* zal er eerst gekeken worden of er nog *holes* in de oplossing zijn. Indien er nog *holes* zijn zal het algoritme hier eerst een cirkel in proberen te plaatsen. Indien de *hole* te klein is voor alle nog-te-plaatsen cirkels wordt deze *hole* simpelweg verwijderd uit de lijst van *holes* in de *solver*. Op deze manier weet de *solver* in de volgende stap dat hij daar niet meer moet proberen om een cirkel te plaatsen, en zal hij een andere *hole* uitproberen. Indien er wel een cirkel in de *hole* past, wordt deze in de *hole* geplaatst. Dit zal leiden tot het creëren van drie nieuwe *holes*, zoals getoond in figuur 4.3 en figuur 4.4. De eerste figuur toont de *hole* waarin een cirkel geplaatst zal worden (de rode driehoek). De tweede figuur toont de nieuwe *packing*, nadat een cirkel geplaatst is in deze *hole*. Er zijn drie nieuwe kleinere *holes* gemaakt, die in de volgende stappen ook zullen opgevuld worden indien mogelijk. Het algoritme zal deze *holes* ook terug proberen op te vullen. In figuur 4.5 en figuur 4.6 wordt respectievelijk de tussen-oplossing getoond voor wanneer er nog een cirkel is die klein genoeg is, en wanneer dit niet het geval is, om de onderste *hole* op te vullen.

4.6.1 Grootste cirkel zoeken die past in een *hole*

Bij het plaatsen van een cirkel in een *hole* wordt een zo groot mogelijke cirkel gezocht die in deze *hole* past. Dit is de best-passende cirkel, vandaar *best-fit*. Meer uitleg over hoe

Figuur 4.4: *Packing* na het opvullen van een *hole*Figuur 4.5: *Packing* na het opvullen van een tweede *hole*



Figuur 4.6: *Packing* als het opvullen van een tweede *hole* mislukt

bepaald wordt of een cirkel past is terug te vinden sectie 4.6.2. Hier wordt het proces om te vinden welke cirkel past verder verduidelijkt. Er wordt binair gezocht door de lijst van cirkels om te bepalen welke cirkel de grootste is die past. Onderstaande code verduidelijkt dit proces.

```

1 Location findBestFitFor(Hole hole, List<Circle> sortedBigToSmall) {
2     // Probeer eerst de kleinste cirkel
3     int lower = sortedBigToSmall.size() - 1;
4     Circle smallestCir = sortedBigToSmall.get(lower);
5     Vector2 smallestPos = hole.tryFit(smallestCir);
6     if (smallestPos == null) {
7         return null;
8     }
9
10    // Probeer dan de grootste cirkel
11    int upper = 0;
12    Circle biggestCir = sortedBigToSmall.get(upper);
13    Vector2 biggestPos = hole.tryFit(biggestCir);
14    if (biggestPos != null) {
15        return new Location(biggestPos, biggestCir);
16    }
17
18    // Binair zoeken
19    Circle cir = null;
20    Vector2 pos = null;

```



```
21     while (lower - upper > 1) {
22         int middle = (upper + lower) / 2;
23         cir = sortedBigToSmall.get(middle);
24         pos = hole.tryFit(cir);
25
26         if (pos == null) {
27             upper = middle;
28         }
29         else {
30             lower = middle;
31         }
32     }
33
34     cir = sortedBigToSmall.get(lower);
35     pos = hole.tryFit(cir);
36     if (pos != null) {
37         return new Location(pos, cir);
38     }
39     else {
40         return null;
41     }
42 }
```

De *solver* houdt de lijst van cirkels bij gesorteerd op grootte, dat is cruciaal om snel de beste-passende cirkel te vinden. Eerst worden de grootste en kleinste cirkel uitgeprobeerd (lijn 3). Indien de kleinste niet past zal het algoritme direct rapporteren dat deze *hole* te klein is. De *hole* zal dan, zoals vermeld in sectie 4.6, verwijderd worden uit de lijst van mogelijk *holes*. Indien de grootste past (lijn 11) zal het algoritme onmiddellijk deze cirkel plaatsen in de *hole*. Er zijn immers geen grotere cirkels, dus deze is de cirkel die verondersteld wordt best te passen. Vervolgens wordt er een gebied bepaald in de overblijvende cirkels, waarin de best-passende cirkel zich bevindt. Initieel ligt de boven- en ondergrens van dit gebied op de uiteinden van de overblijvende cirkels. De cirkel in de midden van dit gebied wordt dan uitgeprobeerd. Afhangende of deze wel of niet past zal de boven- of ondergrens aangepast worden. Dit wordt telkens herhaald tot er nog maar één cirkel over blijft. Dit is dan de grootste cirkel die past, de *best-fit* voor deze *hole*.

4.6.2 Bepalen of een cirkel past in een *hole*

Er is geen exacte definitie van de grootte van een *hole*. Dit is niet mogelijk omdat de cirkels die de *hole* bepalen niet altijd aan elkaar raken. Het is echter wel mogelijk om te bepalen of een cirkel past.

Dit gebeurt door de cirkel die we willen testen te plaatsen in de *hole*. Eerst wordt de cirkel tegen twee van de cirkels in de *hole* geplaatst, met een cirkel-cirkel intersectie zoals beschreven in sectie 4.5. Deze cirkel-cirkel intersectie heeft natuurlijk altijd twee punten. Hiervan moet er één gekozen worden. De implementatie zorgt er voor dat de cirkels die de *hole* definiëren telkens tegen de klok gesorteerd zijn (ten opzichte van het middelpunt van deze drie cirkels). Dit maakt het mogelijk telkens het juiste punt te kiezen. Eenmaal dit punt bepaald is wordt de cirkel op deze plek gezet. Dan wordt gekeken of deze cirkel

wel effectief in de *hole* geplaatst is, en of deze niet overlapt met de derde cirkel die de *hole* definieert.

```

1 public Vector2 tryFit(Circle cir) {
2     // Zoek een positie voor de cirkel
3     Vector2 pos = Helpers.getMountPositionFor(cir, first, second);
4
5     // Kijk na of deze positie wel in de hole ligt
6     boolean inside = Vector2.isInsideTriableBy(first.getPosition(),
7         second.getPosition(), third.getPosition(), pos);
8     if (!inside) {
9         return null;
10    }
11
12    // Test of er geen overlap is met de derde cirkel
13    Location loc = new Location(pos, cir);
14    if (third.overlaps(loc)) {
15        return null;
16    }
17    return pos;
18 }
```

Op lijn 6 wordt verzekerd dat het middelpunt van de cirkel in de *hole* ligt. Dit voorkomt dat de geplaatste cirkel buiten de *hole* geplaatst wordt, en dus zeker niet kan overlappen met cirkels buiten de *hole* zonder ook te overlappen met één van de cirkels die de *hole* definiëren. Op lijn 13 wordt dan de overlap met de derde *hole*-definiërende cirkel nagekeken. Er kan geen overlap zijn met de eerste twee want de methode *getMountPositionFor* plaatst de cirkel zodanig dat deze de twee andere cirkels raakt, maar niet overlapt. Indien de cirkel in de *hole* past, en dus alle checks doorstaat, wordt de positie voor deze cirkel teruggegeven. De *solver* zal deze cirkel dan in zijn oplossing plaatsen.

Het is niet nodig om na te gaan of er overlap is met andere cirkels in de oplossing. Als er met een andere overlap zou zijn, moet dit zijn omdat de cirkel buiten de *hole* geplaatst is, of er is ook overlap met één van de cirkels in de *hole* zelf. Dit zorgt er voor dat er zeer weinig overlap-checks gedaan moeten worden, wat het algoritme zeer snel maakt.

4.7 Shell

De *shell* of *schil* is het tweede belangrijke concept in de heuristiek. Dit is de buitenste laag van cirkels in een (tussen-)oplossing van de *solver*. Deze wordt in de implementatie bijgehouden als een geordende lijst van cirkels. Cirkels die naast elkaar staan in de lijst, grenzen ook aan elkaar in de *shell*. De cirkels in deze laag zijn met de klok mee gesorteerd ten opzichte van het middelpunt van de omcirkel. De eerste en laatste cirkel in de lijst volgen elkaar ook op in de *shell*, en er is dus geen echt begin of einde van de *shell*.

De heuristiek voorgesteld in deze thesis probeert steeds eerst alle *holes* op te vullen. Maar wanneer er geen cirkels meer over zijn die klein genoeg zijn om te passen in *holes*, wordt er een cirkel op de *shell* geplaatst. Op de *shell* worden alle posities tegen twee cirkels van de *shell* overwogen. Deze aanpak beperkt het aantal mogelijke posities voor de geplaatste cirkel enorm en maakt het algoritme dus zeer snel (zie hoofdstuk 6 voor een

vergelijking in snelheid). Er wordt steeds een cirkel zo dicht mogelijk bij het centrum van de huidige tussen-oplossing geplaatst. Dit om het uitbreiden van de omschreven cirkel zo min mogelijk te houden en dus een nieuwe tussen-oplossing te bekomen die zo goed mogelijk is.

Eerst worden twee cirkels gekozen waartegen de nieuwe cirkel geplaatst zal worden. Hiervoor wordt gekeken naar het middelpunt tussen alle cirkels die naast elkaar staan op de *shell*. De twee cirkels waarvoor het middelpunt het dichtst bij het centrum van de huidige omschreven cirkel ligt, worden gekozen als kandidaten om de derde cirkel tegen te plaatsen. Dan wordt er gezocht naar een zo groot mogelijke cirkel die daar past op de *shell*.

Indien geen enkele cirkel past wordt één van de twee kandidaat cirkels verwijderd uit de *shell*. Welke verwijderd wordt is verduidelijkt in sectie 4.7.2. Dit zorgt er voor dat de *shell* verandert en *groeit* naar buiten. In de volgende stap van het algoritme zal dan ook een andere positie voor een cirkel uitgeprobeerd worden.

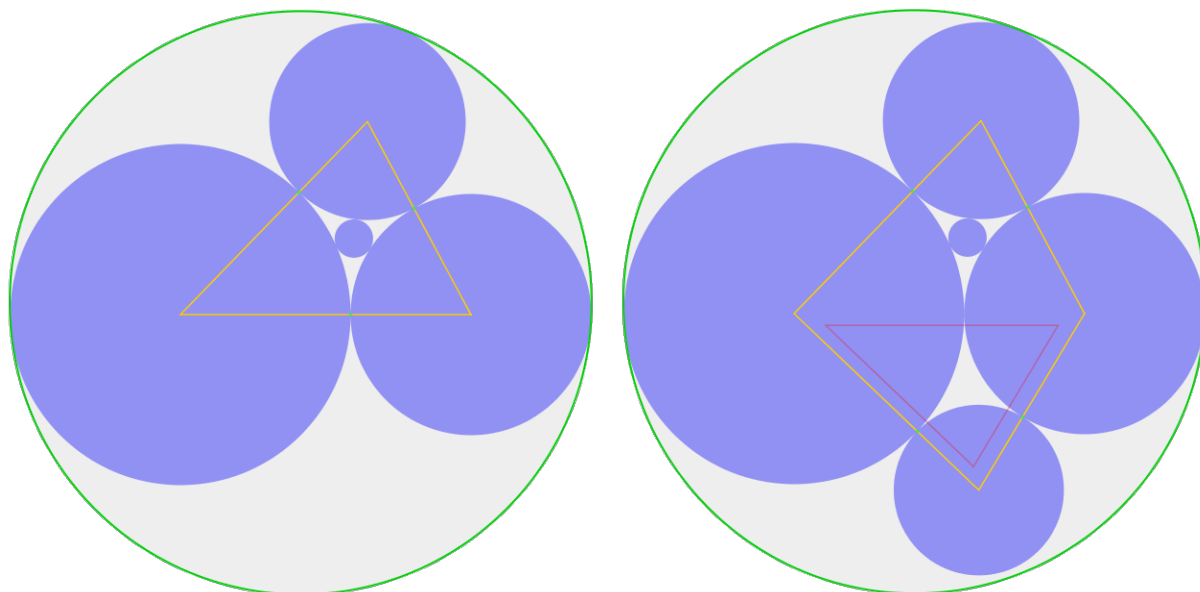
Indien er wel een cirkel past, wordt deze toegevoegd aan de *shell*, en geeft deze aanleiding tot een nieuwe *hole*. Deze *hole* zal vervolgens opgevuld worden indien mogelijk, zoals beschreven in sectie 4.6. Op figuur 4.7 wordt getoond hoe de *packing* verandert wanneer een cirkel geplaatst wordt op de *shell*. Eerst wordt een *packing* zonder *holes* getoond en vervolgens de *packing* nadat een cirkel op de *shell* geplaatst is. Hierop is duidelijk te zien hoe de *shell* aangepast werd, en deze plaatsing geleid heeft tot een nieuwe *hole*. De gele lijn, die de *shell* voorstelt, verbindt nu één extra cirkel, en de rode driehoek toont de nieuwe *hole*.

Zoals eerder aangegeven is het mogelijk dat de grootste cirkel niet past op deze positie op de *shell*. Het algoritme zal dan proberen een (beschikbare) kleinere cirkel op de *shell* te plaatsen, dit wordt geïllustreerd in figuur 4.8. Het is ook mogelijk dat geen enkele cirkel past op de positie in de *shell* die het dichtst bij het centrum van de huidige omschreven cirkel ligt. Dit wordt getoond in figuur 4.9. Dan zal de *shell* uitgebreid worden naar buiten, en zal in de volgende stap van het algoritme een andere positie op de *shell* uitgeprobeerd worden.

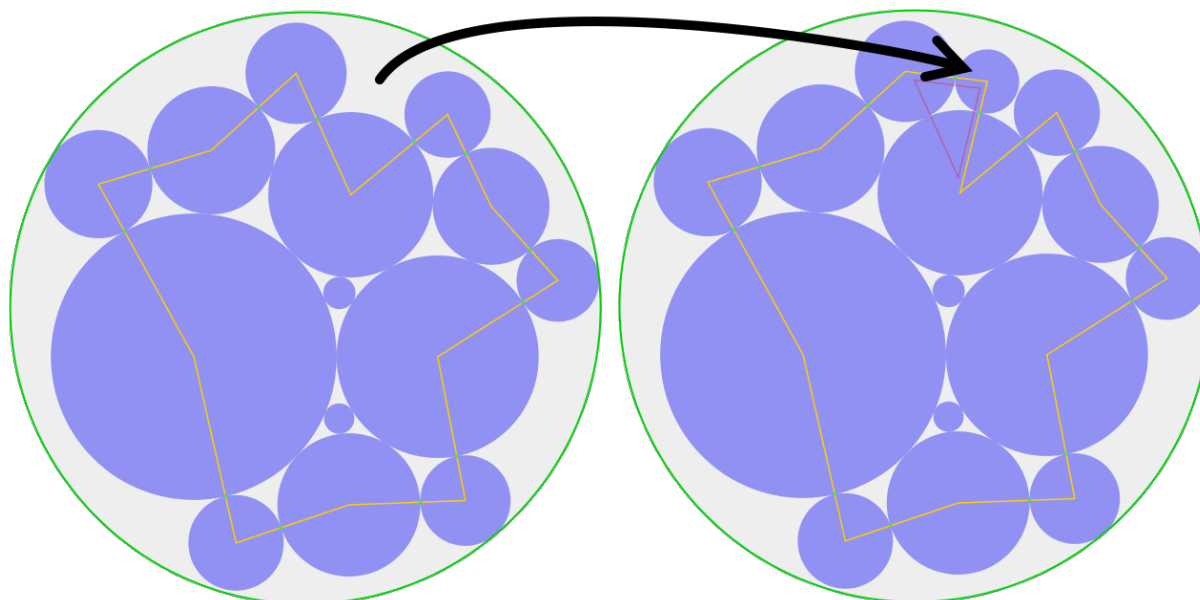
4.7.1 Efficiënter de omcirkel berekenen gebaseerd op de *shell*

Bij het kiezen van de kandidaat-cirkels om een cirkel tegen te plaatsen op de *shell* wordt gebruik gemaakt van de omschreven cirkel van de huidige tussen-oplossing. Zoals eerder vermeld kan voor elke (tussen-)oplossing de omcirkel berekend worden. Dit gebeurt door een aangepaste versie van het Welz algoritme voor de omcirkel van punten beschreven in [20]. De gebruikte implementatie is gebaseerd op de implementatie in [18]. Het is een recursief algoritme dat in lineaire tijd de omcirkel kan berekenen. Het basisidee bestaat eruit dat de omcirkel van een aantal cirkels (of punten) volledig gedefinieerd is door maximum drie van deze cirkels. Het algoritme vindt deze cirkels uit een gegeven lijst cirkels.

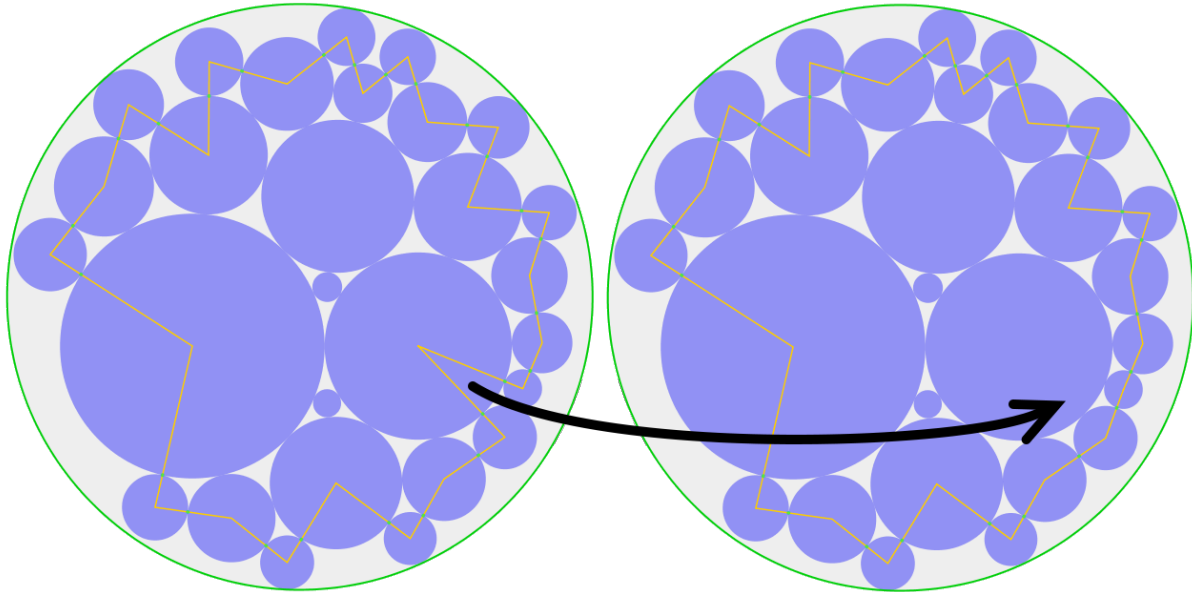
Het is intuïtief te begrijpen dat deze cirkels aan de buitenkant van een oplossing zullen liggen. Dat is ook net waaruit de *shell* bestaat, de cirkels aan de buitenkant van een oplossing. Het is dus mogelijk om in elke stap van de *solver* de omcirkel zeer efficiënt te berekenen. De complexiteit is dan slechts lineair in het aantal cirkels op de *shell* van de huidige (tussen-)oplossing, wat slechts een subset is van het totaal aantal cirkels in



Figuur 4.7: Het plaatsen van de grootste cirkel op de *shell*



Figuur 4.8: Het plaatsen van een kleinere cirkel op de *shell*



Figuur 4.9: *Shell* aanpassen als geen enkele cirkel past

de oplossing. Aangezien de omcirkel regelmatig moet worden herberekend doorheen het algoritme is dit een zeer interessante optimalisatie.

4.7.2 Bepalen of een cirkel past op de *shell*

Om te bepalen of een cirkel op de *shell* past, plaatsen we de cirkel eerst tegen twee cirkels op de *shell*. Dit gebeurt op dezelfde manier als het plaatsen van een cirkel tegen twee cirkels van een *hole*. De exacte methode is reeds uiteengezet in sectie 4.5. Eenmaal deze positie gekend is, wordt er nagekeken of dit niet tot overlap leidt. Indien er overlap is, is het niet mogelijk om de cirkel daar op de *shell* te plaatsen en wordt er informatie terug gegeven over welke cirkel op de *shell* voor problemen zorgt.

Om na te gaan of er overlap is, worden er systematisch een aantal cirkels op de *shell* nagekeken. Het is niet nodig andere cirkels, die niet op de *shell* zitten, na te kijken voor overlap. Dat is zo omdat een nieuwe cirkel steeds op de *shell* geplaatst wordt zodanig dat als er overlap zou zijn met cirkels in de oplossing, minstens één van die cirkels op de *shell* ligt. Omdat ik geen exacte wiskundige definitie van de *shell* heb (de *shell* is enkel gedefinieerd door het algoritme dat ze opbouwt) is het niet mogelijk om hier een echt bewijs van te geven. Uit eigen empirische tests blijkt de verwachting omtrent deze overlappingsen echter wel te kloppen. Verdere bedenkingen hierbij zijn terug vinden in hoofdstuk 5. Om deze overlappingstest zo snel mogelijk te kunnen uitvoeren, worden eerst cirkels dichtbij op de *shell* nagekeken. Er staat ook een limiet op het aantal cirkels dat getest wordt. Uit tests, tot 5000 cirkels, blijkt dat drie cirkels aan elke kant van de *shell* nakijken genoeg is. Voor de meeste gevallen is één cirkel nakijken genoeg, maar sommige randgevallen (wanneer meerdere kleine cirkels dichtbij elkaar staan op de *shell*) vragen deze extra checks.

Het aantal cirkels dat nagekeken wordt is een hyperparameter *checkRadius* van het algoritme. De volgende code toont hoe de overlap nagekeken wordt:

```
1 for (int i = 1; i <= checkRadius; ++i) {
```

```

2   int prevIndex = (firstIndex + shell.size() - i) % shell.size();
3   Location prev = shell.get(prevIndex);
4   if (loc.overlaps(prev)) {
5       toRemove = first;
6       break;
7   }
8   int nextIndex = (secondIndex + i) % shell.size();
9   Location next = shell.get(nextIndex);
10  if (loc.overlaps(next)) {
11      toRemove = second;
12      break;
13  }
14 }

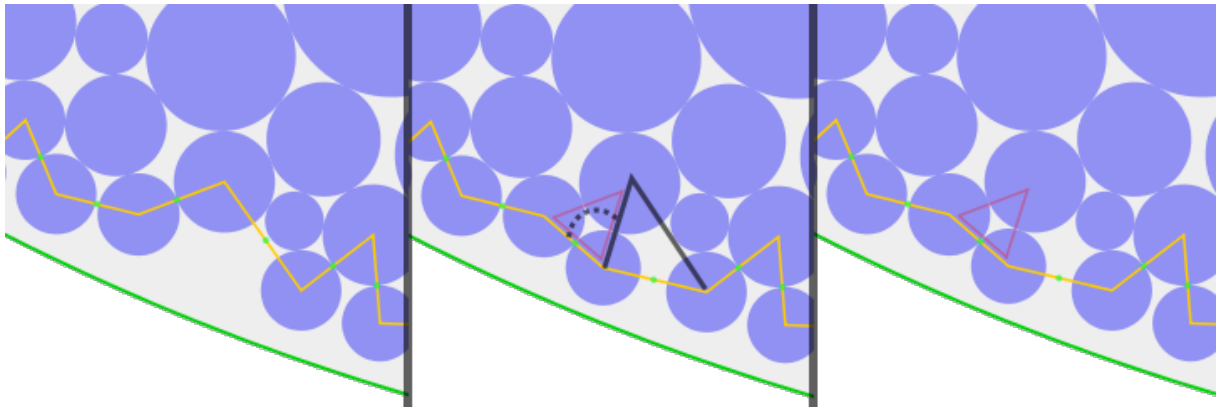
```

Er wordt dus om te beurt links en rechts van de huidige positie op de *shell* nagekeken voor overlap. Indien er overlap gevonden is, wordt bijgehouden aan welke kant van de *shell* dit gebeurd is. *toRemove* is dus uiteindelijk *first* of *second*, wat aangeeft of er aan de kant van de eerste cirkel, of van de tweede cirkel, overlap voorkomt. *toRemove* kan ook *null* zijn indien er geen overlap is. Deze informatie wordt dan gebruikt door de *solver* om te bepalen of de cirkel geplaatst kan worden (*toRemove* = *null*), of dat de *shell* aangepast moet worden. Deze stap wordt hieronder besproken in sectie 4.7.3.

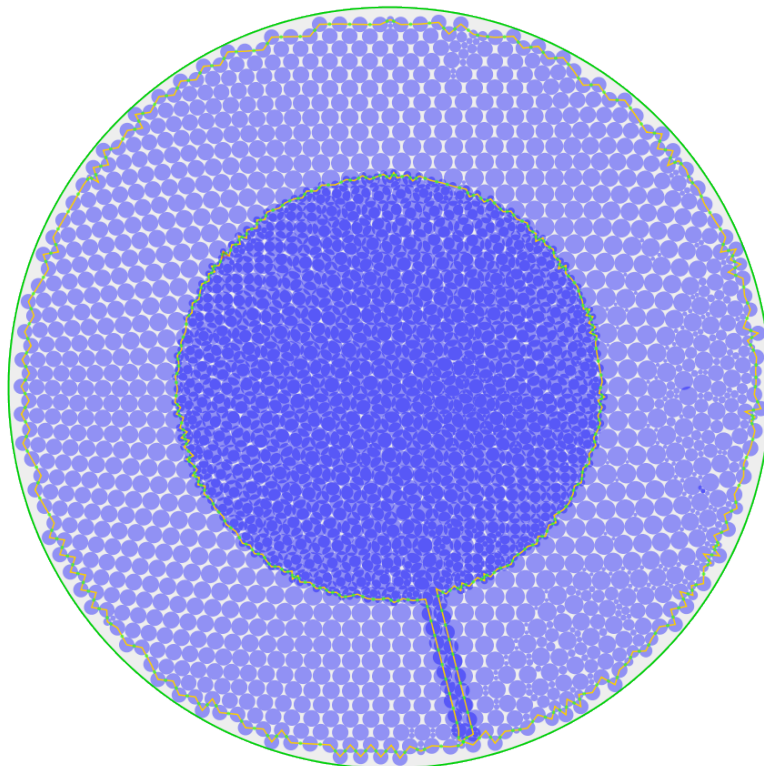
4.7.3 Een cirkel plaatsen op de *shell*

Om een cirkel op de *shell* te plaatsen wordt eerst de grootste cirkel die op de *shell* past binair gezocht in de lijst van nog-te-plaatsen cirkels. De gebruikte methode is vergelijkbaar met het zoeken naar de grootste cirkel die past in een *hole*, zoals beschreven in sectie 4.6.1. Indien er een cirkel gevonden wordt die past op de *shell* wordt de *shell* uitgebreid met deze cirkel. De cirkel wordt dan tussen de cirkels, waartegen hij geplaatst is, in de *shell* (voorgesteld als lijst) geplaatst. Dit wordt geïllustreerd in figuur 4.7, figuur 4.8. De gele lijn (die de *shell* voorstelt, zoals beschreven in hoofdstuk 3) wordt uitgebreid met een extra punt. Dit zorgt er ook voor dat een nieuwe *hole* gemaakt wordt waar mogelijk kleinere cirkels in passen.

Een laatste check gaat na of door het plaatsen van de cirkel de *shell* nog juist gevormd is. Indien een cirkel geplaatst wordt zodat de cirkels niet meer met de klok mee gesorteerd zijn, zou dit ervoor kunnen zorgen dat foutieve plaatsingen voorkomen. Om dit te voorkomen wordt de gerichte hoek tussen de geplaatste cirkel en de cirkels waartegen hij geplaatst is nagekeken. Indien deze hoek negatief is wordt de *shell* hieraan aangepast. Dit wordt getoond in figuur 4.10. Het eerste deel toont de configuratie waarop een cirkel geplaatst zal worden. Het laatste deel de uiteindelijke configuratie van de *shell*. Het middelste deel toont met een zwarte lijn het deel van de *shell* dat verwijderd is om er voor te zorgen dat deze kloksgewijs gesorteerd blijft. Op figuur 4.11 wordt getoond wat er kan gebeuren als deze check niet gebeurt. De cirkels worden doorzichtig getekend, het donker-blauwe deel toont dus overlappende cirkels.



Figuur 4.10: Een cirkel veroorzaakt een tegen-de-klok *shell*



Figuur 4.11: Mogelijke fout indien de *shell* niet met de klok mee gesorteerd is

4.8 Conclusie

Dit hoofdstuk gaf een volledig overzicht van de nieuwe constructieve heuristiek. Waar nodig werd de uitleg verduidelijkt met code uit de implementatie. Er werden ook de verschillende veronderstellingen geformuleerd die gebruikt worden om het algoritme zeer snel te maken. Zie ook hoofdstuk 5 voor verdere bedenkingen hieromtrent.

Hoofdstuk 5

Bedenkingen bij implementatie

5.1 Precisie

De implementatie gebruikt *double* variabelen. Ook geeft de methode gebruikt om cirkels tegen elkaar te plaatsen, beschreven in sectie 4.5, geen exacte oplossing. Deze posities worden gebruikt om cirkels te plaatsen, en later worden tegen deze cirkels nieuwe cirkels geplaatst. Hierdoor wordt er doorheen het algoritme dus een kleine fout opgebouwd. Deze fout uit zich in een kleine overlap tussen cirkels in de uiteindelijke verkregen *packing*. Deze overlap blijft echter beperkt tot 10^{-15} vierkante eenheden voor de grootste *packings*.

Dit is een inherent probleem van werken in eindige precisie. Andere algoritmen hebben dus te kampen met hetzelfde probleem, hoewel er niet altijd aandacht aan wordt besteed in de gepubliceerde papers. In [1], [21] en [16] worden resultaten verkregen met een precisie van respectievelijk 10^{-10} , 10^{-28} en 10^{-7} .

Aangezien de vergelijkingen in hoofdstuk 6 tot een precisie van 10^{-10} gebeuren heeft dit verder geen impact op de resultaten.

5.2 Veronderstellingen

Zoals eerder vermeld worden er bij de uitleg over de *holes* en de *shell* enkele veronderstellingen gedaan om het algoritme zeer snel te laten werken. Deze veronderstellingen lijken voor de meeste problemen correct te zijn. Er zijn echter sommige problemen, vooral als er gewerkt wordt met zeer hoge aantallen cirkels (duizenden), die wél significante fouten kunnen opleveren. Deze uit zich dan in *packings* met een grote overlap.

Uit huidige tests blijkt er vooral een probleem met de *holes* te zijn, de *shell* heeft voor zover mijn huidige tests uitwijzen geen duidelijke problemen. Doordat de cirkels die een *hole* definiëren niet altijd aan elkaar raken kan het gebeuren dat er toch nog een overlap is, als twee *holes* naast elkaar staan.

Een voorstel om dit op te lossen is een *hole* niet op te delen in drie *holes* als er een cirkel in geplaatst wordt, maar slechts in twee. Eén van die *holes* bestaat dan uit meer dan drie cirkels. Zo is het wel mogelijk om altijd aaneensluitende cirkels een *hole* te laten bepalen. Dit is een oplossing die mogelijk geëxploreerd kan worden in de toekomst, maar wegens tijdsbeperking geen deel is van deze thesis.

Hoofdstuk 6

Resultaten

In dit hoofdstuk worden de verkregen *packings* vergeleken met een reeks gekende *packings*, zoals gerapporteerd op de Packomania website [17]. Packomania is een website die de beste oplossingen (in radius van omschreven cirkel) voor verschillende *circle-packing* problemen probeert te verzamelen. De oplossingen worden onderling vergeleken, op zowel de verkregen radius van de omschreven cirkel, als ook de tijd nodig om tot deze *packing* te komen indien deze gekend is. De nodige rekentijd voor deze algoritmen is echter niet altijd even makkelijk te vinden. Veel van de *packings* op Packomania hebben geen bijhorende publicatie. Het gebruikte algoritme en de nodige tijd om de *packing* te bekomen is dus niet gekend, enkel de verkregen omschreven cirkel en coördinaten van de *packing*. De vergelijking in tijd zal dus een zeer ruwe vergelijking zijn gebaseerd op slechts enkele papers.

Ook geef ik resultaten voor *packings* met veel meer cirkels dan deze gerapporteerd op Packomania. Hiervoor heb ik geen andere resultaten gevonden in de literatuur om mee te vergelijken.

Packomania heeft resultaten voor verschillende verdelingen van cirkels. Deze verdelingen zijn als volgt:

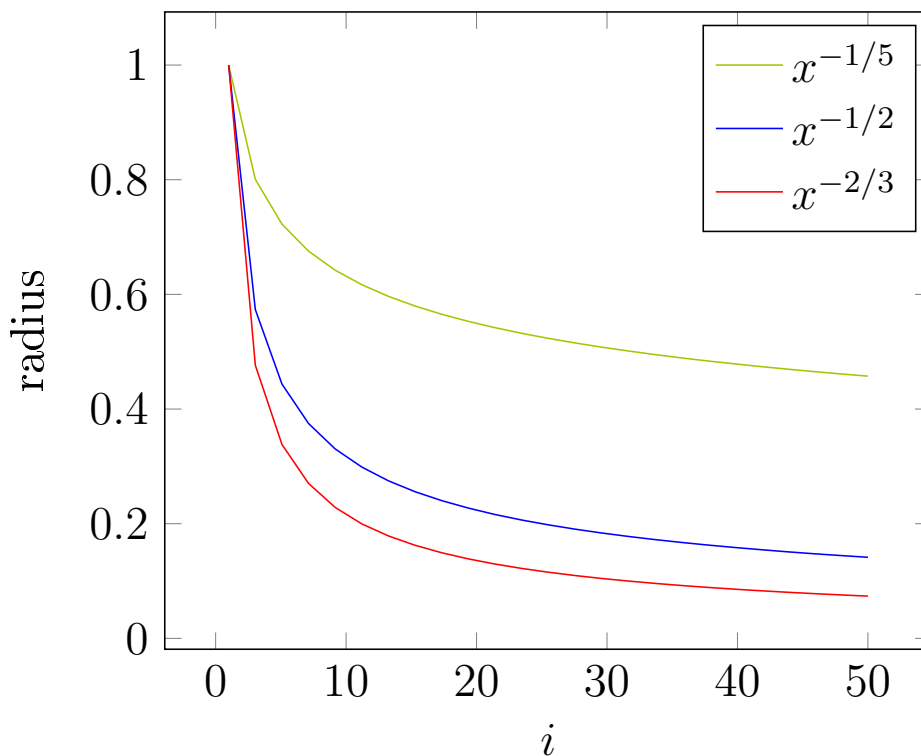
- $r_i = 1$ (alle cirkels gelijke grootte)
- $r_i = i$
- $r_i = i^{1/2}$
- $r_i = i^{-1/5}$
- $r_i = i^{-1/2}$
- $r_i = i^{-2/3}$

Hierbij zijn er telkens N cirkels in het probleem, en is $i \in (1, 2, \dots, N)$.

De laatste vier verdelingen hebben een vaste structuur: $r_i = i^X$. Hier is X steeds een andere macht. Deze problemen verzamel ik onder één noemer: de *Packomania macht* problemen.

De verhouding tussen enkele van deze verdelingen worden verduidelijkt in figuur 6.1.

Al deze berekeningen zijn uitgevoerd op een Mid 2009 Macbook-Pro 15 inch met een 2.8Ghz Core 2 Duo processor en 4GB ram.



Figuur 6.1: Packomania verdelingen

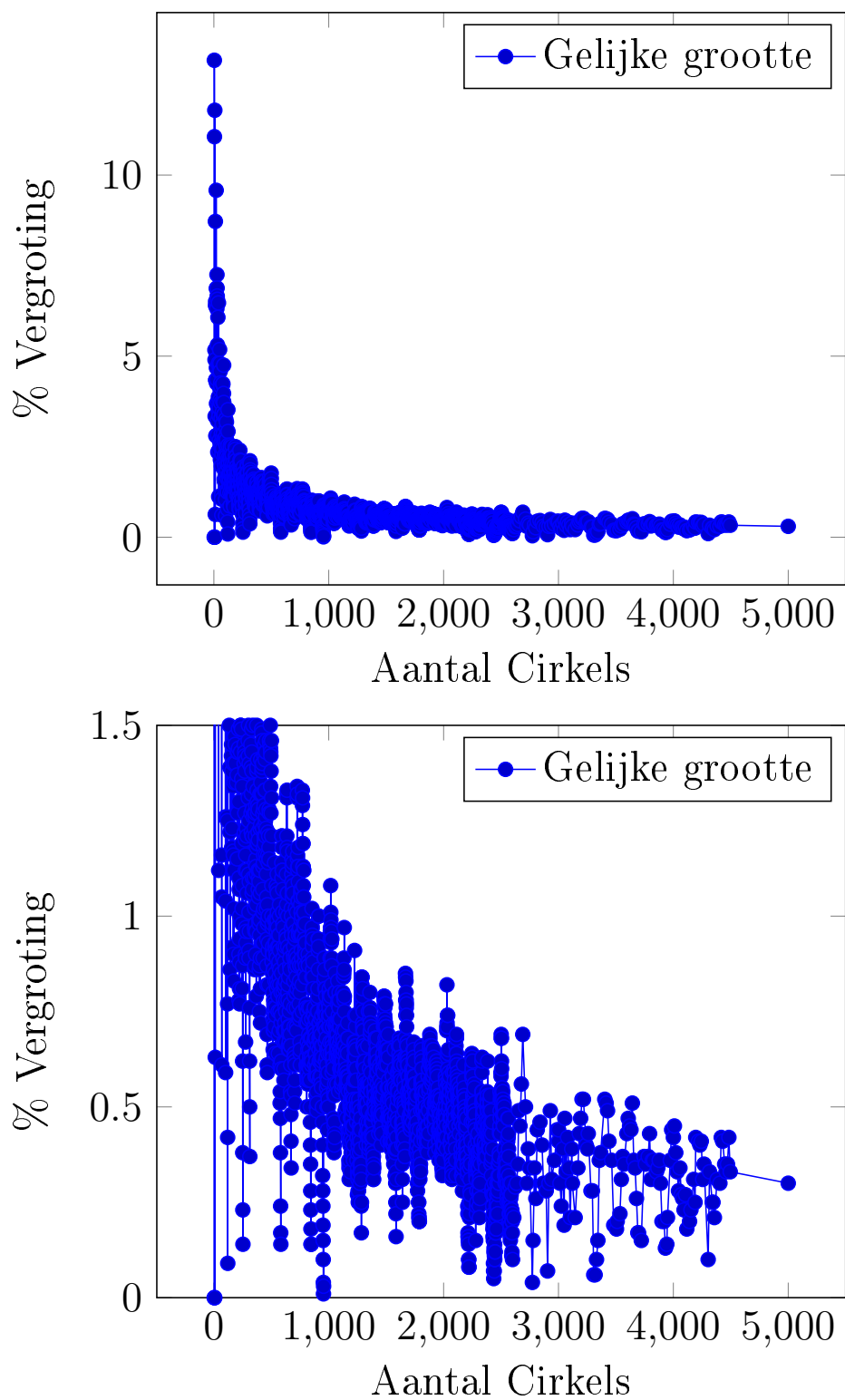
6.1 Packomania vergelijking

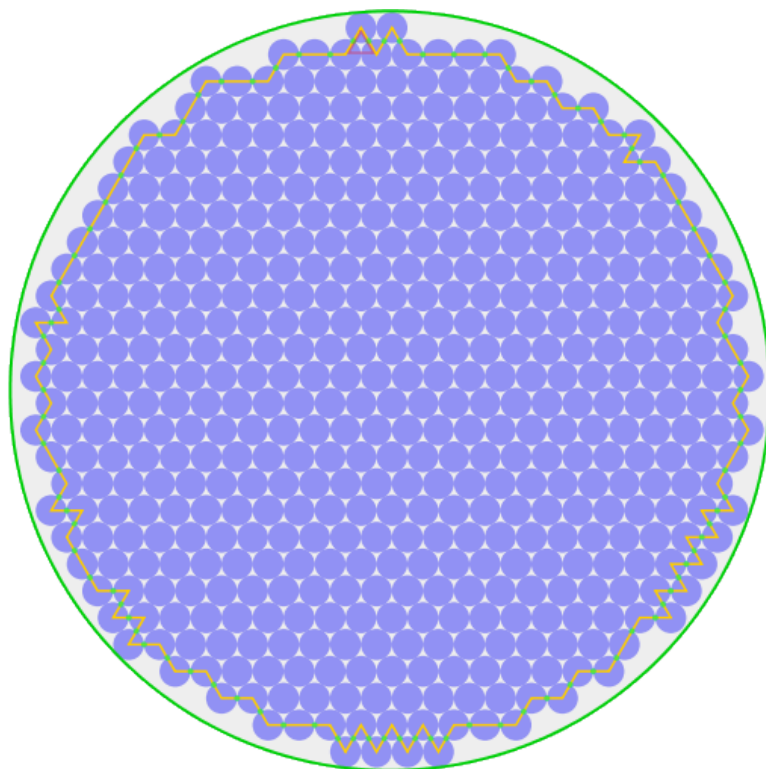
6.1.1 Cirkels met gelijke grootte

In figuur 6.2 wordt de procentuele vergroting van mijn oplossing ten opzichte van de best gekende weergegeven voor *packings* met N cirkels van gelijke grootte. Er wordt een volledig beeld gegeven, als ook een gedetailleerder beeld waar vergrotingen boven de 1,5% niet worden weergegeven. Deze vergroting is afgemeten ten opzichte van de best gekende oplossing op de Packomania website. Tabellen die mijn resultaten tonen vergeleken met de best gekende, zijn terug te vinden in bijlage B.1.

Gemiddeld geeft mijn algoritme een resultaat dat 0,79% groter is dan het best gekende resultaat voor problemen met cirkels van gelijke grootte. Er is echter een duidelijke neerwaartse trend. Voor problemen met meer dan 1000 cirkels is de gemiddelde vergroting nog slechts 0,48%. Voor meer dan 2000 cirkels nog 0,39% en voor meer dan 3000 nog 0,32%.

Dit komt mogelijk omdat hoe meer cirkels er zijn, hoe moeilijker het probleem wordt. Hierdoor zullen de best gekende oplossingen voor problemen met zulke grote aantallen cirkels minder dicht bij de absolute optimale oplossing liggen. Omdat mijn algoritme enkel nieuwe cirkels tegen twee reeds geplaatste cirkels plaatst, hebben al mijn oplossingen voor dit probleem een duidelijke structuur. Deze structuur is een zeshoekige structuur. Dit wordt geïllustreerd in figuur 6.3. Dit is een zeer eenvoudige structuur, maar benadert redelijk goed de best gekende oplossingen. Meer figuren zijn terug te vinden in bijlage C.

Figuur 6.2: Vergelijking van *packings* van cirkels met gelijke grootte



Figuur 6.3: *Packing* voor 500 cirkels met gelijke grootte

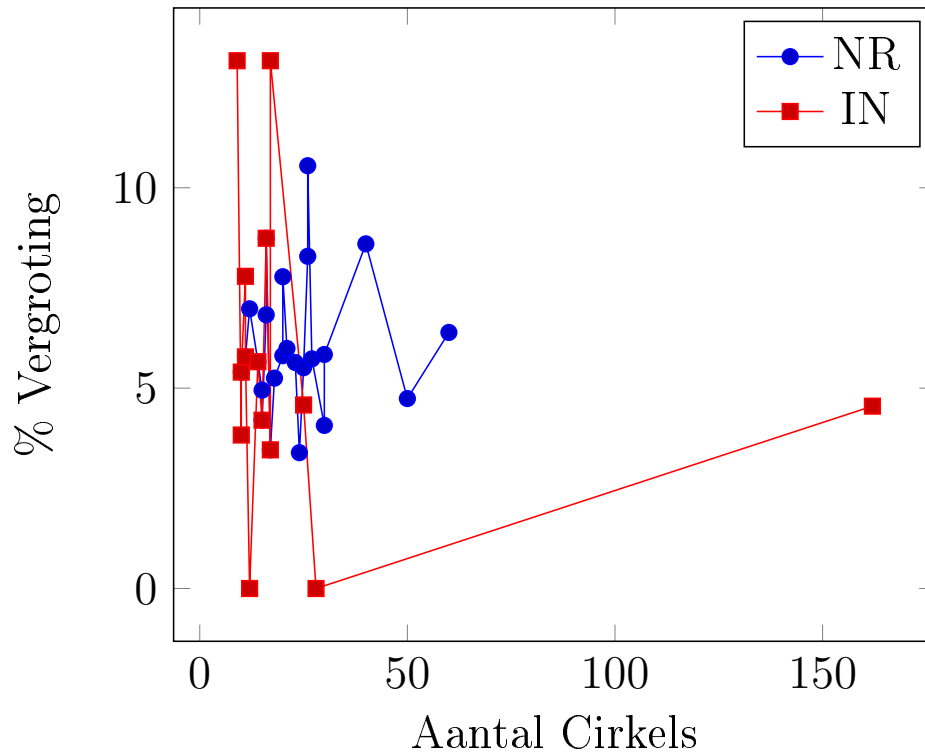
De benodigde tijd om tot een oplossing te komen blijft onder 10 milliseconden tot $N = 100$. Tot 500 cirkels blijft de nodige rekestijd onder 100 milliseconden. De rekestijd loopt pas op tot één seconde bij 2200 cirkels, en slechts 3,5 seconden voor 5000 cirkels.

Zoals eerder vermeld zijn vergelijkingen van berekeningstijden moeilijk te vinden. In [9] worden echter tijden gerapporteerd van 110 seconden voor 30 cirkels (mijn algoritme gebruikte hier minder dan een milliseconde), en 15311 seconden (meer dan 4 uur) voor 100 cirkels (mijn algoritme gebruikte voor hetzelfde probleem 7 milliseconden). Zij halen resultaten die zeer dichtbij de best gekende liggen (soms de beste).

Zelfs de constructieve algoritmen beschreven in [1] en [10] vragen meerdere seconden, en soms minuten, om te berekenen. Hun resultaten zijn echter moeilijk te vergelijken omdat [1] een alternatieve versie van *circle-packing* oplost, en [10] *packings* maakt in rechthoeken.

Bespreking

Over het algemeen zijn dit goede resultaten. Ze liggen behoorlijk dicht bij de best gekende oplossing, maar hebben een duidelijk en eenvoudig patroon. Dit patroon is duidelijk te zien in figuur 6.3, en is eenvoudig te genereren. Dit zijn dus zeker niet de meest indrukwekkende resultaten. Het probleem waar alle cirkels een gelijke grootte hebben is dan ook niet de kracht van mijn algoritme. Aangezien elke cirkel even groot is, zal er nooit een cirkel gevonden worden die in een *hole* past. Ook is dit het eenvoudigste probleem. Het feit dat er resultaten te vinden zijn tot 5000 cirkels op Packomania, maar problemen met verschillende radii slechts tot maximum 200 cirkels opgelost zijn, wijst



Figuur 6.4: Vergelijking van Packomania Benchmark problemen

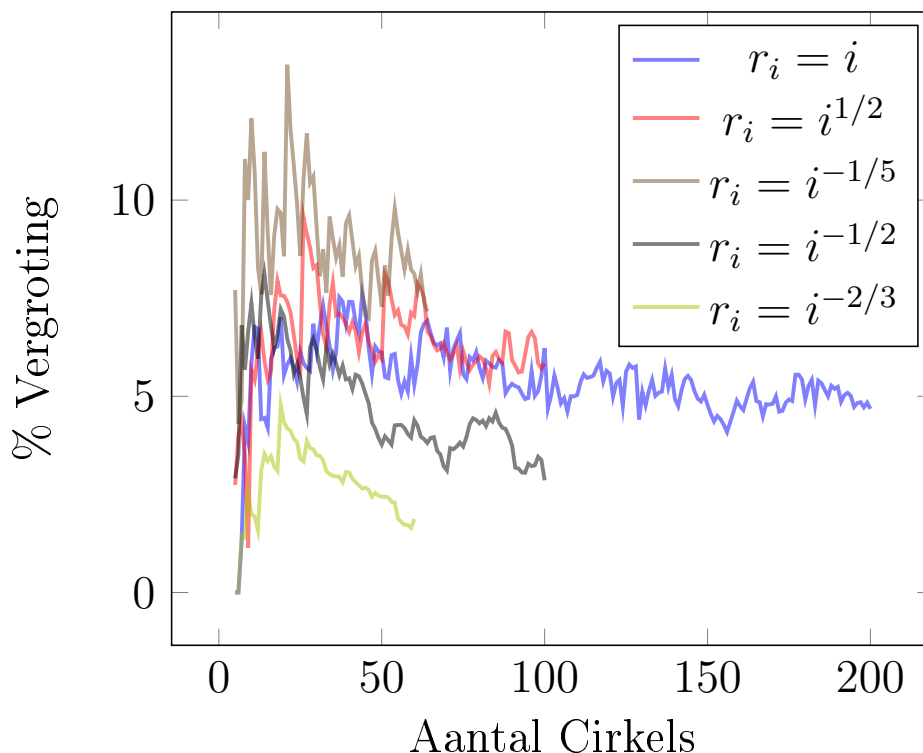
hier op. Hoewel mijn algoritme dus relatief goede resultaten geeft voor dit probleem, zijn de resultaten die volgen interessanter.

6.1.2 Packomania Benchmark

Packomania geeft ook resultaten voor enkele benchmark instanties. Deze benchmarks werden geïntroduceerd door WenQi Huang in verscheidene papers, en samengevat in [12]. De radii van de cirkels in deze benchmark instanties zijn terug te vinden in bijlage A. Een eerste set verdelingen (met prefix *NR*) heeft tussen 10 en 60 cirkels per probleem, een tweede set (met prefix *IN*) tussen 9 en 28 cirkels met nog één probleem van 162 cirkels. In elke verdeling zitten cirkels met verschillende raddii, maar ook cirkels met gelijke raddii.

In figuur 6.4 wordt de procentuele vergroting van de oplossing van mijn algoritme vergeleken met de best gekende oplossing. Gemiddeld geeft mijn algoritme een resultaat dat 5,94% groter is dan de best gekende resultaten. Door het zeer gelimiteerd aantal instanties is er echter geen duidelijke trend te zien zoals bij *packings* voor cirkels met gelijke grootte, en de Packomania Macht problemen (zie sectie 6.1.3).

De benodigde rekentijd bedraagt voor al deze problemen ook slechts enkele milliseconden. In [21] worden deze problemen opgelost in tijden tussen 1 seconde voor 10 cirkels, en meer dan 2 uur voor 60 cirkels. Ook in [13] kan de nodige tijd oplopen tot meerdere uren voor de grotere problemen. Zelfs de kleinere problemen (10 tot 15 cirkels) kunnen meerdere minuten vragen om berekend te worden. Mijn algoritme heeft slechts enkele milliseconden nodig voor dezelfde problemen.



Figuur 6.5: Vergelijking van Packomania Macht problemen

Bespreking

Ook deze resultaten zijn over het algemeen goed. Gemiddeld is er slechts een 5,94% vergroting van de radius van de omschreven cirkel, met slechts 3 van de 38 problemen die meer dan 10% vergroting hebben. Er zijn echter ook 2 problemen die even goed opgelost worden als de best gekende.

Dit zijn dus goede resultaten, maar het aantal probleeminstanties blijft zeer beperkt. In de volgende sectie wordt de kracht van dit nieuwe algoritme duidelijker met meer cirkel-verdelingen.

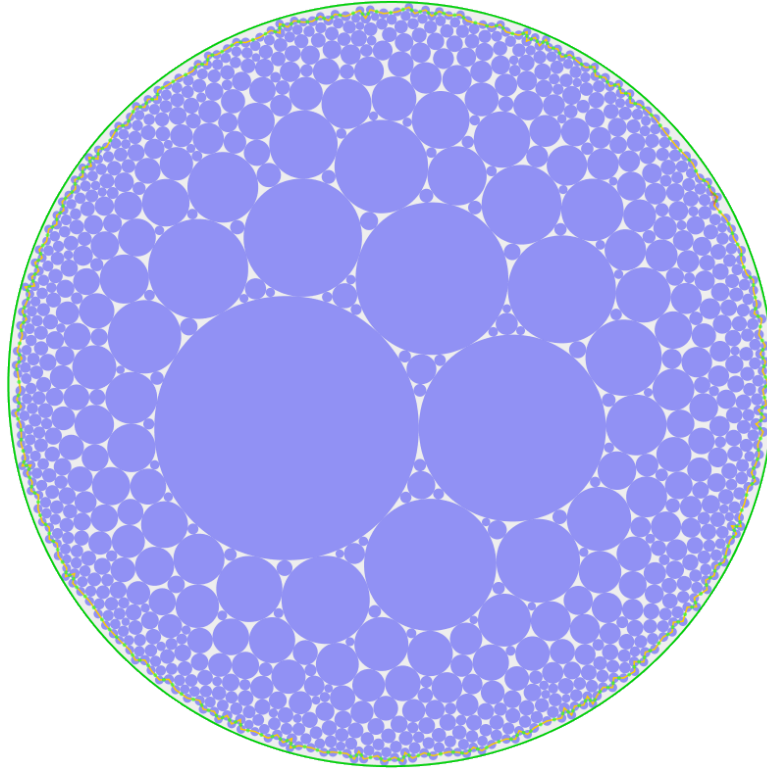
6.1.3 Packomania Machten

Zoals eerder vermeld neem ik enkele van de Packomania problemen onder één noemer: de *Packomania macht* problemen. Dit zijn de verdelingen met een vaste structuur: $r_i = i^X$. Hier is $X \in \{1/2, -1/5, -1/2, -2/3\}$. Tabellen die mijn resultaten met de best gekende vergelijken zijn terug te vinden in bijlage B.2.

In figuur 6.5 worden voor al deze macht problemen de procentuele vergroting getoond. Dit is de omtrek van de omschreven cirkel verkregen uit mijn algoritme ten opzichte van de best gekende oplossingen.

Gemiddeld geeft mijn algoritme een resultaat dat 5,61% groter is dan de best gekende resultaten voor deze problemen. Ook is er duidelijk een neerwaartse trend. Problemen met een groter aantal cirkels liggen dichterbij de best gekende oplossing.

In volgende tabel wordt de gemiddelde vergroting per macht probleem weergegeven:



Figuur 6.6: *Packing* voor 1000 cirkels met verdeling $r_i = i^{-1/2}$

Tabel 6.1: Packomania Machten gemiddelde vergroting

| Probleem | Aantal cirkels | % Vergroting |
|------------------|----------------|--------------|
| $r_i = i$ | 5 tot 200 | 5,41% |
| $r_i = i^{1/2}$ | 5 tot 100 | 6,42% |
| $r_i = i^{-1/2}$ | 5 tot 100 | 4,77% |
| $r_i = i^{-2/3}$ | 5 tot 60 | 2,78% |
| $r_i = i^{-1/5}$ | 5 tot 64 | 8,88% |

Nodige rekentijd is voor al deze problemen slechts enkele milliseconden voor mijn algoritme, terwijl de rekentijd voor de implementaties in de literatuur oploopt tot 6 uur in [2], tot 7 uur in [14] en zelfs meer dan een volledige dag (27 uur) in [22], voor slechts 30 cirkels. Het snelste andere algoritme, tot zover ik weet, werd gegeven in [21]. Zij hadden tijden tussen 1 seconden voor 5 cirkels, en 1 uur voor 30 cirkels. Zij hadden echter geen goed stopping-criteria, en deze tijd was dus de tijd nodig alvorens ze hun beste oplossing vonden, maar hun algoritme bleef ongeacht meer dan 2,5 uur zoeken.

In figuur 6.6 wordt een *packing* getoond voor het $r_i = i^{-1/2}$ probleem. Meer afbeeldingen van *packings* zijn terug te vinden in bijlage C

Bespreking

Deze resultaten zijn volgens mij zeer indrukwekkend. Resultaten voor de problemen $r_i = i$, $r_i = i^{1/2}$, $r_i = i^{-1/2}$ en $r_i = i^{-2/3}$ blijven steeds onder 10% vergroting tegenover de best

gekende. Enkel het probleem $r_i = i^{-1/5}$ heeft enkele uitschieters tot maximum 13,45% vergroting. Dit laatste resultaat komt waarschijnlijk omdat $r_i = i^{-1/5}$ het probleem is waarin de cirkels het minst verschillen van grootte (zie figuur 6.1). Dit zorgt er voor dat het algoritme minder gebruik kan maken van *holes*, aangezien meestal enkel de kleinere cirkels *holes* zullen opvullen. Het probleem met het meeste verschil in radii, $r_i = i^{-2/3}$, heeft dan ook de beste resultaten. Met maximum 4,77% vergroting en gemiddeld slechts 2,78%.

Ook is er een duidelijke neerwaartse trend zichtbaar in de vergroting van de omschreven cirkel als het aantal cirkels groter wordt. Dit komt omdat het algoritme dan meer keuzemogelijkheden heeft in elke stap. Hierdoor zal het algoritme vaker *holes* kunnen opvullen, en een dichtere *shell* opbouwen. De kracht van het algoritme is dus het duidelijkst zichtbaar wanneer er genoeg verschillende cirkels zijn om uit te kiezen.

Bovendien worden deze resultaten zeer snel bekomen. Waar andere algoritmen uit de literatuur uren nodig hebben, vraagt mijn algoritme slechts milliseconden.

Dit zijn denk ik de meest indrukwekkende resultaten van deze thesis, en tonen het best de kracht van het algoritme.

6.2 Grotere aantallen cirkels

Het algoritme uit deze thesis laat ook toe om problemen met veel grotere aantallen cirkels op te lossen. Resultaten voor 1000 tot 14000 cirkels zijn te vinden in bijlage B.4. Ik heb geen andere resultaten gevonden voor problemen met zo'n grote aantallen cirkels. De verkregen radii van de omschreven cirkel kan ik dus niet vergelijken. Ik heb echter wel de eigenaar van de Packomania website gecontacteerd om mijn resultaten aan zijn website toe te voegen. Zo kunnen in de toekomst andere resultaten vergeleken worden met die van mij. Op het moment dat ik dit schrijf heb ik hier echter nog geen antwoord op gekregen.

De nodige tijd om deze problemen op te lossen blijft ook beperkt. Problemen tot 1400 cirkels blijven onder één seconde. Problemen tot 9000 cirkels onder 10 seconden, en 14000 cirkels vragen minder dan één minuut tijd.

Bij deze grote aantallen cirkels komen echter wel enkele problemen naar boven, zoals beschreven in hoofdstuk 5. Ook was het niet mogelijk op mijn computer om problemen met meer dan 14000 cirkels op te lossen. Bij het nakijken van de oplossing wordt een recursief algoritme gebruikt op de volledige lijst cirkels. Dit geeft een *stack-overflow* en heb ik nog niet kunnen oplossen.

Ondanks de resterende problemen met overlap voor sommige van deze problemen met grote aantallen cirkels, zijn dit interessante resultaten. Dit is, voor zover ik weet, het eerste constructieve algoritme dat goede resultaten geeft voor kleine problemen (resultaten in sectie 6.1.2 en sectie 6.1.3) en dat ook gebruikt kan worden om grotere problemen aan te pakken.

6.3 Conclusie

In dit hoofdstuk heb ik oplossingen verkregen door mijn algoritme te vergelijken met de best gekende *packings*. Ook heb ik de nodige rekentijd vergeleken met enkele andere algoritmen.

De oplossingen verkregen met mijn nieuwe constructieve heuristiek hebben een gemiddeld minder dan 6% grotere omschreven cirkel. Dit lijkt mij een zeer aanvaardbare vergroting van de omschreven cirkel, rekening houdend met het feit dat mijn algoritme deze problemen in slechts milliseconden oplost. Zelfs de grootste problemen, bestaande uit duizenden cirkels, kunnen in slechts enkele seconden opgelost worden. Ook is er een interessante neerwaartse trend in de vergroting van de radius van de bekomen omschreven cirkel. Dit toont de kracht van het algoritme wanneer het meer cirkels heeft met verschillende radii om uit te kiezen.

Bovendien laat het algoritme toe om problemen met veel grotere aantallen cirkels, met verschillende radii, in aanvaardbare tijd op te lossen.

Hoofdstuk 7

Verder werk

In dit hoofdstuk geef ik enkele voorstellen voor verdere uitwerkingen van het algoritme. Mogelijke verbeteringen, zowel in snelheid als kwaliteit (de grootte van de bekomen omcirkel) van uitkomsten. Dit zijn mogelijke ideeën om in de toekomst op verder te werken.

7.1 Constante-tijd omschreven cirkel

In het algoritme wordt bij het plaatsen van een cirkel op de *shell* gebruik gemaakt van de omschreven cirkel van de huidige tussen-oplossing. Deze omschreven cirkel wordt berekend in lineaire tijd ten opzichte van het aantal cirkels op de *shell*. Dit geeft dan de kleinst mogelijke omschreven cirkel van de huidige oplossing.

Het is echter mogelijk om in constante tijd een omschreven cirkel te berekenen. Simpelweg door, telkens er een cirkel geplaatst wordt buiten de huidige omschreven cirkel, een nieuwe te berekenen die rond de oude omcirkel en deze nieuwe cirkel past. Dit kan zeer eenvoudig in constante tijd, maar geeft niet de kleinst-mogelijke omschreven cirkel voor de *packing*. Deze constante-tijd omschreven cirkel is mogelijk wel goed genoeg als benadering, en kan een grote snelheids-winst geven voor zeer grote *packings*.

7.2 Best-Fit

Het algoritme is gebaseerd op een *best-fit* principe. Met andere woorden in elke stap wordt de best-passende cirkel gekozen. Welke cirkel het best past is echter moeilijk te definiëren. In de huidige implementatie worden er daarom enkele keuzes gemaakt, gebaseerd op bestaande heuristieken en eigen intuïtie. Hier kunnen echter andere keuzes gemaakt worden, die mogelijk tot betere resultaten zullen leiden.

Eén voorbeeld van een alternatieve keuze is bij het plaatsen van een cirkel op de *shell*. Nu worden de nieuwe cirkels zo dicht mogelijk bij het huidige middelpunt geplaatst. Dit gebeurt omdat de omschreven cirkel dan zo weinig mogelijk uitgebreid moet worden. Maar wat als, wanneer we geen cirkel meer binnen de huidige omcirkel kunnen plaatsen, we de cirkel zo ver mogelijk van het middelpunt plaatsen (maar wel nog tegen twee cirkels op de *shell*)? Dit zal er voor zorgen dat de omschreven cirkel veel verder uitgebreid zal worden. Het zal er ook voor zorgen dat de omcirkel veel minder vaak uitgebreid moet worden, omdat hij in grotere stappen uitbreidt. Mogelijk geeft dit aanleiding tot betere

en misschien snellere *packings* omdat de omschreven cirkel minder vaak herberekend moet worden.

Hoofdstuk 8

Conclusie

In deze thesis heb ik een nieuwe constructieve heuristiek voorgesteld voor het oplossen van het *circle-packing* probleem. Deze is gebaseerd op het *best-fit* principe, waarbij in elke stap de beste cirkel gekozen wordt om aan de oplossing toe te voegen. Deze nieuwe heuristiek gebruikt twee nieuwe concepten: *holes* en de *shell*. Eerst worden de *holes* opgevuld, daarna worden de cirkels geplaatst aan de buitenkant van de tussen-oplossing, op de *shell*. Door enkele veronderstellingen te maken over het aantal cirkels die getest moeten worden voor overlap bij elke plaatsing, kan het algoritme zeer snel goede *packings* genereren. In deze thesis worden de resultaten voor een groot aantal cirkel-verdelingen getoond en vergeleken met gekende *packings* uit de literatuur. Deze tonen aan dat het algoritme goede *packings* maakt in vergelijking met bestaande algoritmen. Deze *packings* werden gegenereerd in een fractie van de tijd die andere, *state-of-the-art* algoritmen, nodig hebben (milliseconden tegenover uren). Ook is het mogelijk om dit zelfde algoritme, dat goed werkt voor kleine problemen, te gebruiken om problemen met veel meer cirkels op te lossen.

Daarom denk ik dat ik geslaagd ben in mijn opzet: het maken van een constructief algoritme dat in real-time goede *packings* kan maken voor het *cirkel-packing* probleem.

Bibliografie

- [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] P. Bollansée. Packomania tabellen. <https://github.com/circle-packing/docs/blob/master/thesis/packomania-tables.pdf>.
- [5] 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.
- [6] John A George, Jennifer M George, and Bruce W Lamar. Packing different-sized circles into a rectangular container. *European Journal of Operational Research*, 84(3):693–712, 1995.
- [7] Ronald L Graham and Boris D Lubachevsky. Repeated patterns of dense packings of equal disks in a square. *the electronic journal of combinatorics*, 3(1):R16, 1996.
- [8] Ronald L Graham, Boris D Lubachevsky, Kari J Nurmela, and Patric RJ Östergård. Dense packings of congruent circles in a circle. *Discrete Mathematics*, 181(1):139–154, 1998.
- [9] 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.
- [10] Mhand Hifi and Rym M'Hallah. Approximate algorithms for constrained circular cutting problems. *Computers & Operations Research*, 31(5):675–694, 2004.
- [11] Mhand Hifi, Vangelis Th Paschos, and Vassilis Zissimopoulos. A simulated annealing approach for the circular cutting problem. *European Journal of Operational Research*, 159(2):430–448, 2004.
- [12] Wen Qi Huang, Yu Li, Chu Min Li, and Ru Chu Xu. New heuristics for packing unequal circles into a circular container. *Computers & Operations Research*, 33(8):2125–2142, 2006.

- [13] WenQi Huang, ZhangHua Fu, and RuChu Xu. Tabu search algorithm combined with global perturbation for packing arbitrary sized circles into a circular container. *Science China Information Sciences*, 56(9):1–14, 2013.
- [14] CO Lopez and JE Beasley. Packing unequal circles using formulation space search. *Computers & Operations Research*, 40(5):1276–1288, 2013.
- [15] Boris D Lubachevsky and Ronald L Graham. Curved hexagonal packings of equal disks in a circle. *Discrete & Computational Geometry*, 18(2):179–194, 1997.
- [16] Rym M’Hallah, Abdulaziz Alkandari, and Nenad Mladenovic. Packing unit spheres into the smallest sphere using vns and nlp. *Computers & Operations Research*, 40(2):603–615, 2013.
- [17] E. Specht. Packomania. <http://www.packomania.com/>. Accessed: 2016-05-23.
- [18] Sunshine. Computing the smallest enclosing disk in 2d. <http://www.sunshine2k.de/coding/java/Welzl/Welzl.html>, 2008.
- [19] Huaiqing Wang, Wenqi Huang, Quan Zhang, and Dongming Xu. An improved algorithm for the packing of unequal circles within a larger containing circle. *European Journal of Operational Research*, 141(2):440–453, 2002.
- [20] Emo Welzl. *Smallest enclosing disks (balls and ellipsoids)*. Springer, 1991.
- [21] Tao Ye, Wenqi Huang, and Zhipeng Lu. Iterated tabu search algorithm for packing unequal circles in a circle. *arXiv preprint arXiv:1306.0694*, 2013.
- [22] Zhizhong Zeng, Xinguo Yu, Kun He, Wenqi Huang, and Zhanghua Fu. Iterated tabu search and variable neighborhood descent for packing unequal circles into a circular container. *European Journal of Operational Research*, 250(2):615–627, 2016.

Bijlage A

Packomania Benchmark Verdelingen

Tabel A.1: Packomania Benchmark Verdelingen

| Instantie | N | Radii |
|-----------|----|--|
| NR10-1 | 10 | 10, 12, 15, 20, 21, 30, 30, 30, 40, 50 |
| NR11-1 | 11 | 8.4, 11, 10, 10.5, 12, 14, 15, 20, 20, 25, 25 |
| NR12-1 | 12 | 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22 |
| NR14-1 | 14 | 11, 14, 15, 16, 17, 19, 23, 27, 31, 35, 36, 37, 38, 40 |
| NR15-1 | 15 | 3, 3, 4, 4, 4.5, 6, 7.5, 8, 9, 10, 11, 12, 13, 14, 15 |
| NR15-2 | 15 | 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 |
| NR16-1 | 16 | 13, 14, 15, 15, 17, 19, 23, 26, 27, 27, 32, 37, 38, 47, 57, 63 |
| NR16-2 | 16 | 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36 |
| NR17-1 | 17 | 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 10, 10, 10, 15, 15, 20, 25 |
| NR18-1 | 18 | 12, 14, 16, 23, 25, 26, 27, 28, 33, 35, 47, 49, 53, 53, 55, 60, 67, 71 |
| NR20-1 | 20 | 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42 |
| NR20-2 | 20 | 6, 8, 9, 12, 12, 15, 16, 18, 20, 21, 24, 24, 27, 28, 30, 32, 33, 36, 40, 44 |
| NR21-1 | 21 | 10, 15, 16, 17, 17, 18, 21, 22, 23, 25, 26, 31, 33, 34, 37, 37, 38, 39, 40, 42, 45 |
| NR23-1 | 23 | 14, 14, 16, 18, 18, 21, 22, 23, 26, 28, 28, 32, 34, 34, 36, 37, 39, 41, 45, 48, 49, 49, 51 |
| NR24-1 | 24 | 9, 10, 11, 13, 13, 16, 17, 17, 18, 19, 19, 20, 20, 20, 21, 22, 23, 23, 24, 25, 30, 31, 32, 82 |
| NR25-1 | 25 | 14, 17, 22, 26, 26, 27, 28, 29, 29, 30, 31, 32, 33, 34, 34, 34, 34, 35, 37, 37, 37, 47, 52, 53, 55 |
| NR26-1 | 26 | 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56 |
| NR26-2 | 26 | 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66 |
| NR27-1 | 27 | 17, 21, 25, 26, 26, 27, 27, 28, 29, 33, 33, 34, 35, 35, 35, 37, 40, 42, 43, 44, 45, 49, 53, 55, 55, 55, 63 |
| NR30-1 | 30 | 5, 8, 10, 10, 12, 14, 15, 16, 18, 20, 20, 20, 20, 20, 22, 24, 25, 26, 30, 30, 30, 30, 35, 40, 40, 45, 48, 50, 55, 60 |
| NR30-2 | 30 | 6, 8, 8, 10, 12, 13, 14, 16, 18, 18, 20, 22, 23, 24, 25, 27, 28, 29, 31, 33, 33, 35, 37, 38, 39, 41, 43, 43, 48, 53 |

Bijlage B

Packomania Vergelijking Tabellen

In deze tabellen wordt telkens het aantal cirkels in de kolom N gegeven, de best gekende radius van op Packomania in kolom *Besteradius* en de radius verkregen door mijn algoritme in kolom *Radius*. Ook wordt het absolute verschil van mijn oplossing ten opzichte van de best gekende radius in de kolom *Vergroting* gegeven, en de procentuele vergroting in kolom *Vergroting(%)*. Als laatste wordt de rekentijd die mijn algoritme nodig had om tot deze oplossing te komen gegeven in kolom *Tijd*.

B.1 Cirkels met gelijke grootte

Merk op dat voor het *circle-packing* probleem met cirkels van gelijke grootte enkel de oplossingen voor $N = 3$ tot $N = 100$ hier getoond worden. Dit is slechts een subset van alle gekende oplossingen. De volledige lijst van vergelijkingen voor dit probleem, meer dan 2500 vergelijkingen, kan u terug vinden op GitHub [4].

Tabel B.1: Packomania $r_i = 1$

| N | Beste radius | Radius | Vergroting | Vergroting (%) | Tijd |
|----|--------------|-------------|--------------|----------------|--------|
| 3 | 2.154700538 | 2.154700538 | 0.0000000000 | 0.00 | 0s:0ms |
| 4 | 2.414213562 | 2.732050808 | 0.3178372452 | 13.17 | 0s:0ms |
| 5 | 2.701301617 | 3 | 0.2986983833 | 11.06 | 0s:0ms |
| 6 | 3 | 3 | 0.0000000000 | 0.00 | 0s:0ms |
| 7 | 3 | 3 | 0.0000000000 | 0.00 | 0s:0ms |
| 8 | 3.304764871 | 3.694301256 | 0.3895363853 | 11.79 | 0s:0ms |
| 9 | 3.61312593 | 3.8 | 0.1868740702 | 5.17 | 0s:0ms |
| 10 | 3.813025631 | 4 | 0.1869743686 | 4.90 | 0s:0ms |
| 11 | 3.9238044 | 4.055050463 | 0.1312460631 | 3.34 | 0s:0ms |
| 12 | 4.02960193 | 4.055050463 | 0.0254485332 | 0.63 | 0s:0ms |
| 13 | 4.236067977 | 4.605551275 | 0.3694832980 | 8.72 | 0s:0ms |
| 14 | 4.328428555 | 4.605551275 | 0.2771227206 | 6.40 | 0s:0ms |
| 15 | 4.521356965 | 4.815756806 | 0.2943998410 | 6.51 | 0s:0ms |
| 16 | 4.615425595 | 4.815756806 | 0.2003312108 | 4.34 | 0s:0ms |
| 17 | 4.792033748 | 5 | 0.2079662517 | 4.34 | 0s:0ms |
| 18 | 4.863703305 | 5 | 0.1362966948 | 2.80 | 0s:0ms |

Tabel B.1: Packomania $r_i = 1$

| N | Beste radius | Radius | Vergroting | Vergroting (%) | Tijd |
|----|--------------|-------------|--------------|----------------|--------|
| 19 | 4.863703305 | 5 | 0.1362966948 | 2.80 | 0s:0ms |
| 20 | 5.122320737 | 5.613025038 | 0.4907043009 | 9.58 | 0s:0ms |
| 21 | 5.252317475 | 5.613025038 | 0.3607075629 | 6.87 | 0s:0ms |
| 22 | 5.439718959 | 5.694198862 | 0.2544799032 | 4.68 | 0s:0ms |
| 23 | 5.545204223 | 5.75 | 0.2047957774 | 3.69 | 0s:0ms |
| 24 | 5.651661092 | 5.891911267 | 0.2402501756 | 4.25 | 0s:0ms |
| 25 | 5.752824331 | 6.003702333 | 0.2508780021 | 4.36 | 0s:0ms |
| 26 | 5.828176537 | 6.196152423 | 0.3679758858 | 6.31 | 0s:0ms |
| 27 | 5.906397847 | 6.33487632 | 0.4284784726 | 7.25 | 0s:0ms |
| 28 | 6.014938097 | 6.428571429 | 0.4136333312 | 6.88 | 0s:0ms |
| 29 | 6.138597904 | 6.547689565 | 0.4090916605 | 6.66 | 0s:0ms |
| 30 | 6.197741071 | 6.604759882 | 0.4070188110 | 6.57 | 0s:0ms |
| 31 | 6.291502622 | 6.625462944 | 0.3339603218 | 5.31 | 0s:0ms |
| 32 | 6.429462971 | 6.636363636 | 0.2069006654 | 3.22 | 0s:1ms |
| 33 | 6.486703124 | 6.669933699 | 0.1832305752 | 2.82 | 0s:1ms |
| 34 | 6.61095709 | 6.766281297 | 0.1553242073 | 2.35 | 0s:1ms |
| 35 | 6.697171092 | 7.103417131 | 0.4062460396 | 6.07 | 0s:1ms |
| 36 | 6.746753793 | 7 | 0.2532462066 | 3.75 | 0s:1ms |
| 37 | 6.758770483 | 7 | 0.2412295169 | 3.57 | 0s:2ms |
| 38 | 6.961886965 | 7.293994497 | 0.3321075319 | 4.77 | 0s:1ms |
| 39 | 7.057884163 | 7.331139971 | 0.2732558085 | 3.87 | 0s:1ms |
| 40 | 7.123846436 | 7.584629125 | 0.4607826895 | 6.47 | 0s:1ms |
| 41 | 7.260012329 | 7.609084657 | 0.3490723281 | 4.81 | 0s:1ms |
| 42 | 7.346796407 | 7.429100507 | 0.0823041004 | 1.12 | 0s:1ms |
| 43 | 7.419944856 | 7.677768534 | 0.2578236776 | 3.47 | 0s:1ms |
| 44 | 7.498036683 | 7.677768534 | 0.1797318509 | 2.40 | 0s:1ms |
| 45 | 7.572912326 | 7.82657133 | 0.2536590037 | 3.35 | 0s:2ms |
| 46 | 7.650179915 | 8 | 0.3498200853 | 4.57 | 0s:2ms |
| 47 | 7.724170053 | 8 | 0.2758299474 | 3.57 | 0s:2ms |
| 48 | 7.791271431 | 8 | 0.2087285694 | 2.68 | 0s:2ms |
| 49 | 7.886870959 | 8.253232038 | 0.3663610794 | 4.65 | 0s:2ms |
| 50 | 7.947515275 | 8.359291376 | 0.4117761008 | 5.18 | 0s:2ms |
| 51 | 8.027506952 | 8.408103671 | 0.3805967186 | 4.74 | 0s:2ms |
| 52 | 8.084717191 | 8.472517892 | 0.3878007012 | 4.80 | 0s:2ms |
| 53 | 8.179582827 | 8.488877336 | 0.3092945096 | 3.78 | 0s:2ms |
| 54 | 8.203982383 | 8.538461538 | 0.3344791550 | 4.08 | 0s:2ms |
| 55 | 8.211102551 | 8.587147277 | 0.3760447262 | 4.58 | 0s:2ms |
| 56 | 8.383529923 | 8.6 | 0.2164700774 | 2.58 | 0s:2ms |
| 57 | 8.447184653 | 8.628073151 | 0.1808884978 | 2.14 | 0s:2ms |
| 58 | 8.52455377 | 8.741697766 | 0.2171439957 | 2.55 | 0s:3ms |
| 59 | 8.592499959 | 8.95896775 | 0.3664677911 | 4.26 | 0s:3ms |
| 60 | 8.646219845 | 8.967076656 | 0.3208568103 | 3.71 | 0s:3ms |
| 61 | 8.661297576 | 9.021337505 | 0.3600399294 | 4.16 | 0s:3ms |

Tabel B.1: Packomania $r_i = 1$

| N | Beste radius | Radius | Vergroting | Vergroting (%) | Tijd |
|-----|--------------|-------------|--------------|----------------|--------|
| 62 | 8.829765409 | 9.045787337 | 0.2160219284 | 2.45 | 0s:3ms |
| 63 | 8.892351538 | 9.185352772 | 0.2930012343 | 3.29 | 0s:3ms |
| 64 | 8.961971108 | 9.185352772 | 0.2233816634 | 2.49 | 0s:3ms |
| 65 | 9.017397323 | 9.275353858 | 0.2579565352 | 2.86 | 0s:3ms |
| 66 | 9.096279427 | 9.275353858 | 0.1790744315 | 1.97 | 0s:3ms |
| 67 | 9.168971882 | 9.275353858 | 0.1063819766 | 1.16 | 0s:3ms |
| 68 | 9.229773747 | 9.326663998 | 0.0968902511 | 1.05 | 0s:3ms |
| 69 | 9.269761267 | 9.326663998 | 0.0569027312 | 0.61 | 0s:3ms |
| 70 | 9.345653194 | 9.632545009 | 0.2868918150 | 3.07 | 0s:4ms |
| 71 | 9.415796897 | 9.632545009 | 0.2167481122 | 2.30 | 0s:4ms |
| 72 | 9.473890857 | 9.801090841 | 0.3271999848 | 3.45 | 0s:4ms |
| 73 | 9.540346152 | 9.801090841 | 0.2607446893 | 2.73 | 0s:4ms |
| 74 | 9.589232764 | 9.888925926 | 0.2996931615 | 3.13 | 0s:4ms |
| 75 | 9.672029632 | 9.909090909 | 0.2370612771 | 2.45 | 0s:4ms |
| 76 | 9.729596802 | 10 | 0.2704031978 | 2.78 | 0s:4ms |
| 77 | 9.798911925 | 10.21190226 | 0.4129903345 | 4.21 | 0s:4ms |
| 78 | 9.8577099 | 10.27521352 | 0.4175036178 | 4.24 | 0s:4ms |
| 79 | 9.905063468 | 10.27521352 | 0.3701500500 | 3.74 | 0s:4ms |
| 80 | 9.968151813 | 10.34126647 | 0.3731146566 | 3.74 | 0s:5ms |
| 81 | 10.01086424 | 10.38839772 | 0.3775334833 | 3.77 | 0s:5ms |
| 82 | 10.05082422 | 10.38839772 | 0.3375735011 | 3.36 | 0s:5ms |
| 83 | 10.11685788 | 10.40638081 | 0.2895229379 | 2.86 | 0s:5ms |
| 84 | 10.14953087 | 10.55252264 | 0.4029917684 | 3.97 | 0s:6ms |
| 85 | 10.16311147 | 10.64622206 | 0.4831105925 | 4.75 | 0s:6ms |
| 86 | 10.29870105 | 10.68161804 | 0.3829169837 | 3.72 | 0s:5ms |
| 87 | 10.36320851 | 10.71444133 | 0.3512328258 | 3.39 | 0s:5ms |
| 88 | 10.43233769 | 10.71444133 | 0.2821036381 | 2.70 | 0s:5ms |
| 89 | 10.50049181 | 10.71444133 | 0.2139495163 | 2.04 | 0s:5ms |
| 90 | 10.54606918 | 10.71444133 | 0.1683721529 | 1.60 | 0s:5ms |
| 91 | 10.56677223 | 10.8488578 | 0.2820855683 | 2.67 | 0s:5ms |
| 92 | 10.68464585 | 10.8488578 | 0.1642119539 | 1.54 | 0s:5ms |
| 93 | 10.7333526 | 10.97505539 | 0.2417027933 | 2.25 | 0s:6ms |
| 94 | 10.77803216 | 10.97505539 | 0.1970232333 | 1.83 | 0s:6ms |
| 95 | 10.84020502 | 11.12553963 | 0.2853346097 | 2.63 | 0s:6ms |
| 96 | 10.88320276 | 11.13086122 | 0.2476584589 | 2.28 | 0s:6ms |
| 97 | 10.93859011 | 11.15326389 | 0.2146737787 | 1.96 | 0s:6ms |
| 98 | 10.97938313 | 11.19403587 | 0.2146527432 | 1.96 | 0s:6ms |
| 99 | 11.03314115 | 11.22161834 | 0.1884771882 | 1.71 | 0s:7ms |
| 100 | 11.08214972 | 11.22161834 | 0.1394686153 | 1.26 | 0s:6ms |

B.2 Packomania Macht problemen

Tabel B.2: Packomania $r_i = i$

| N | Beste radius | Radius | Vergroting | Vergroting (%) | Tijd |
|----|--------------|-------------|---------------|----------------|--------|
| 5 | 9.001397746 | 9.001397746 | 0.0000000000 | 0.00 | 0s:0ms |
| 6 | 11.0570404 | 11.0570404 | 0.0000000000 | 0.00 | 0s:0ms |
| 7 | 13.46211068 | 13.63864452 | 0.1765338429 | 1.31 | 0s:0ms |
| 8 | 16.22174668 | 16.9084418 | 0.6866951196 | 4.23 | 0s:0ms |
| 9 | 19.23319391 | 19.96907573 | 0.7358818171 | 3.83 | 0s:0ms |
| 10 | 22.00019301 | 23.2877147 | 1.2875216823 | 5.85 | 0s:0ms |
| 11 | 24.96063429 | 26.65104175 | 1.6904074596 | 6.77 | 0s:0ms |
| 12 | 28.37138944 | 30.27869776 | 1.9073083282 | 6.72 | 0s:0ms |
| 13 | 31.54586702 | 32.93900634 | 1.3931393191 | 4.42 | 0s:0ms |
| 14 | 35.09564714 | 36.65714933 | 1.5615021892 | 4.45 | 0s:0ms |
| 15 | 38.83799551 | 40.46917515 | 1.6311796458 | 4.20 | 0s:0ms |
| 16 | 42.45811644 | 44.97670011 | 2.5185836705 | 5.93 | 0s:0ms |
| 17 | 46.29134212 | 49.17727631 | 2.8859341932 | 6.23 | 0s:0ms |
| 18 | 50.11976262 | 53.26425023 | 3.1444876017 | 6.27 | 0s:0ms |
| 19 | 54.24029359 | 58.0413688 | 3.8010752065 | 7.01 | 0s:0ms |
| 20 | 58.40056748 | 61.63712441 | 3.2365569305 | 5.54 | 0s:0ms |
| 21 | 62.55887709 | 66.19614909 | 3.6372719987 | 5.81 | 0s:0ms |
| 22 | 66.76028624 | 70.74575745 | 3.9854712097 | 5.97 | 0s:0ms |
| 23 | 71.19946161 | 75.28800296 | 4.0885413566 | 5.74 | 0s:0ms |
| 24 | 75.74914258 | 79.82437643 | 4.0752338484 | 5.38 | 0s:0ms |
| 25 | 80.28586444 | 85.20507608 | 4.9192116398 | 6.13 | 0s:0ms |
| 26 | 84.97819107 | 89.93692251 | 4.9587314446 | 5.84 | 0s:0ms |
| 27 | 89.75096268 | 94.88136225 | 5.1303995654 | 5.72 | 0s:0ms |
| 28 | 94.5258771 | 99.86768794 | 5.3418108395 | 5.65 | 0s:1ms |
| 29 | 99.48311156 | 106.2962349 | 6.8131233045 | 6.85 | 0s:1ms |
| 30 | 104.5403638 | 111.0148098 | 6.4744460321 | 6.19 | 0s:1ms |
| 31 | 109.6292407 | 117.0878782 | 7.4586375248 | 6.80 | 0s:1ms |
| 32 | 114.7998147 | 123.1802853 | 8.3804706408 | 7.30 | 0s:1ms |
| 33 | 120.0656596 | 128.215259 | 8.1495993452 | 6.79 | 0s:1ms |
| 34 | 125.3669392 | 133.247447 | 7.8805077664 | 6.29 | 0s:1ms |
| 35 | 130.8490788 | 138.2771663 | 7.4280874986 | 5.68 | 0s:1ms |
| 36 | 136.3079108 | 144.6463617 | 8.3384509125 | 6.12 | 0s:1ms |
| 37 | 141.7837334 | 152.4389507 | 10.6552172234 | 7.52 | 0s:1ms |
| 38 | 147.4521165 | 158.3732762 | 10.9211597336 | 7.41 | 0s:1ms |
| 39 | 153.1997434 | 163.7256474 | 10.5259040974 | 6.87 | 0s:1ms |
| 40 | 159.0215769 | 170.8117726 | 11.7901957151 | 7.41 | 0s:1ms |
| 41 | 164.8062304 | 177.004655 | 12.1984245382 | 7.40 | 0s:1ms |
| 42 | 170.697811 | 182.440372 | 11.7425609294 | 6.88 | 0s:1ms |
| 43 | 176.7326562 | 188.4172803 | 11.6846240215 | 6.61 | 0s:2ms |
| 44 | 182.7726194 | 196.6620902 | 13.8894708123 | 7.60 | 0s:2ms |
| 45 | 188.964969 | 202.5286416 | 13.5636726221 | 7.18 | 0s:2ms |
| 46 | 195.2039014 | 207.3758687 | 12.1719673837 | 6.24 | 0s:2ms |
| 47 | 201.4861933 | 212.7149266 | 11.2287332287 | 5.57 | 0s:2ms |

Tabel B.2: Packomania $r_i = i$

| N | Beste radius | Radius | Vergroting | Vergroting (%) | Tijd |
|----|--------------|-------------|---------------|----------------|--------|
| 48 | 207.8008434 | 220.8336262 | 13.0327828219 | 6.27 | 0s:2ms |
| 49 | 214.1809301 | 227.3484969 | 13.1675667664 | 6.15 | 0s:2ms |
| 50 | 220.5654003 | 234.0553916 | 13.4899913627 | 6.12 | 0s:2ms |
| 51 | 227.6803362 | 240.1831765 | 12.5028402517 | 5.49 | 0s:2ms |
| 52 | 234.4402714 | 247.2368978 | 12.7966263895 | 5.46 | 0s:2ms |
| 53 | 241.13286 | 255.7171621 | 14.5843020125 | 6.05 | 0s:2ms |
| 54 | 247.8032878 | 262.868371 | 15.0650832017 | 6.08 | 0s:2ms |
| 55 | 254.7210367 | 268.5141407 | 13.7931039905 | 5.41 | 0s:3ms |
| 56 | 261.640041 | 274.9108253 | 13.2707843507 | 5.07 | 0s:3ms |
| 57 | 268.4240815 | 282.8776937 | 14.4536122679 | 5.38 | 0s:3ms |
| 58 | 275.4715577 | 289.605714 | 14.1341563166 | 5.13 | 0s:3ms |
| 59 | 282.3339759 | 299.2757681 | 16.9417921475 | 6.00 | 0s:3ms |
| 60 | 289.3422659 | 304.6812371 | 15.3389711825 | 5.30 | 0s:3ms |
| 61 | 296.717773 | 313.9597079 | 17.2419348953 | 5.81 | 0s:3ms |
| 62 | 303.7777367 | 323.9177529 | 20.1400161641 | 6.63 | 0s:3ms |
| 63 | 310.9235322 | 332.2123812 | 21.2888489762 | 6.85 | 0s:3ms |
| 64 | 318.5409988 | 340.6053063 | 22.0643074867 | 6.93 | 0s:3ms |
| 65 | 325.5680022 | 347.0994671 | 21.5314649851 | 6.61 | 0s:3ms |
| 66 | 333.243233 | 354.4394593 | 21.1962262651 | 6.36 | 0s:3ms |
| 67 | 340.7251536 | 362.181251 | 21.4560974215 | 6.30 | 0s:3ms |
| 68 | 348.1910155 | 370.3151028 | 22.1240873386 | 6.35 | 0s:4ms |
| 69 | 355.6061123 | 376.9371949 | 21.3310825298 | 6.00 | 0s:4ms |
| 70 | 363.5378366 | 388.0662131 | 24.5283764994 | 6.75 | 0s:4ms |
| 71 | 371.3559914 | 392.3562249 | 21.0002335141 | 5.66 | 0s:4ms |
| 72 | 378.9536133 | 402.5022428 | 23.5486294950 | 6.21 | 0s:4ms |
| 73 | 386.7761998 | 411.6110859 | 24.8348860634 | 6.42 | 0s:4ms |
| 74 | 394.6660409 | 420.7679123 | 26.1018713787 | 6.61 | 0s:4ms |
| 75 | 402.3461644 | 424.7177231 | 22.3715586755 | 5.56 | 0s:4ms |
| 76 | 410.1895435 | 432.5052319 | 22.3156883095 | 5.44 | 0s:4ms |
| 77 | 418.5499612 | 443.6467399 | 25.0967787844 | 6.00 | 0s:4ms |
| 78 | 426.3941267 | 451.6199239 | 25.2257972078 | 5.92 | 0s:4ms |
| 79 | 434.7680082 | 461.4070427 | 26.6390345424 | 6.13 | 0s:4ms |
| 80 | 442.7191587 | 468.0681694 | 25.3490106831 | 5.73 | 0s:5ms |
| 81 | 450.9036642 | 475.5595161 | 24.6558519794 | 5.47 | 0s:4ms |
| 82 | 459.3903517 | 486.5415641 | 27.1512124091 | 5.91 | 0s:5ms |
| 83 | 467.6800865 | 494.9875395 | 27.3074530526 | 5.84 | 0s:5ms |
| 84 | 475.8025872 | 503.1575199 | 27.3549327168 | 5.75 | 0s:5ms |
| 85 | 484.4776292 | 512.2361165 | 27.7584873045 | 5.73 | 0s:5ms |
| 86 | 492.7211187 | 521.9387989 | 29.2176801460 | 5.93 | 0s:5ms |
| 87 | 501.3589996 | 530.6873361 | 29.3283365110 | 5.85 | 0s:6ms |
| 88 | 509.9171158 | 535.810191 | 25.8930752125 | 5.08 | 0s:5ms |
| 89 | 518.3668565 | 545.4204186 | 27.0535621118 | 5.22 | 0s:5ms |
| 90 | 526.9030101 | 554.9318685 | 28.0288584034 | 5.32 | 0s:6ms |

Tabel B.2: Packomania $r_i = i$

| N | Beste radius | Radius | Vergroting | Vergroting (%) | Tijd |
|-----|--------------|-------------|---------------|----------------|---------|
| 91 | 535.6241529 | 563.8084375 | 28.1842846818 | 5.26 | 0s:6ms |
| 92 | 544.4518726 | 572.9225881 | 28.4707155757 | 5.23 | 0s:5ms |
| 93 | 553.2844216 | 581.3293274 | 28.0449058225 | 5.07 | 0s:5ms |
| 94 | 562.2873009 | 590.033592 | 27.7462911062 | 4.93 | 0s:6ms |
| 95 | 570.9739657 | 599.2930087 | 28.3190430029 | 4.96 | 0s:6ms |
| 96 | 579.9547414 | 612.6535248 | 32.6987833884 | 5.64 | 0s:6ms |
| 97 | 589.1016036 | 618.6339788 | 29.5323752126 | 5.01 | 0s:6ms |
| 98 | 598.0873457 | 628.7495879 | 30.6622421306 | 5.13 | 0s:6ms |
| 99 | 607.2130456 | 640.7227437 | 33.5096980965 | 5.52 | 0s:6ms |
| 100 | 615.8676392 | 654.2342342 | 38.3665949658 | 6.23 | 0s:6ms |
| 101 | 625.4129166 | 654.7565755 | 29.3436589135 | 4.69 | 0s:6ms |
| 102 | 635.0014481 | 666.8855903 | 31.8841421988 | 5.02 | 0s:6ms |
| 103 | 644.0261452 | 679.660035 | 35.6338898842 | 5.53 | 0s:6ms |
| 104 | 653.2488265 | 687.260894 | 34.0120675068 | 5.21 | 0s:7ms |
| 105 | 662.757924 | 694.7227123 | 31.9647883149 | 4.82 | 0s:7ms |
| 106 | 672.3912463 | 705.6561613 | 33.2649149921 | 4.95 | 0s:7ms |
| 107 | 681.7774575 | 713.2447002 | 31.4672427496 | 4.62 | 0s:6ms |
| 108 | 691.021475 | 725.6010759 | 34.5796008747 | 5.00 | 0s:7ms |
| 109 | 700.2736046 | 735.262392 | 34.9887873584 | 5.00 | 0s:7ms |
| 110 | 710.0999889 | 745.7975534 | 35.6975644566 | 5.03 | 0s:7ms |
| 111 | 719.5987078 | 758.1779863 | 38.5792784127 | 5.36 | 0s:8ms |
| 112 | 728.9808997 | 769.5217717 | 40.5408719841 | 5.56 | 0s:8ms |
| 113 | 739.2681393 | 779.6794931 | 40.4113538763 | 5.47 | 0s:8ms |
| 114 | 748.9170311 | 790.0110369 | 41.0940057618 | 5.49 | 0s:8ms |
| 115 | 758.5403224 | 800.4993412 | 41.9590188761 | 5.53 | 0s:8ms |
| 116 | 768.0575864 | 811.9736432 | 43.9160567286 | 5.72 | 0s:9ms |
| 117 | 778.0802266 | 823.8439535 | 45.7637268808 | 5.88 | 0s:8ms |
| 118 | 787.955193 | 830.3459981 | 42.3908051008 | 5.38 | 0s:8ms |
| 119 | 797.8699489 | 842.9611167 | 45.0911678906 | 5.65 | 0s:8ms |
| 120 | 808.3093264 | 853.4123081 | 45.1029817409 | 5.58 | 0s:8ms |
| 121 | 818.2475492 | 859.0828754 | 40.8353261717 | 4.99 | 0s:8ms |
| 122 | 828.1642448 | 872.3640645 | 44.1998196689 | 5.34 | 0s:9ms |
| 123 | 837.9108474 | 883.3198174 | 45.4089699367 | 5.42 | 0s:9ms |
| 124 | 848.265692 | 888.5239112 | 40.2582192058 | 4.75 | 0s:8ms |
| 125 | 858.5698155 | 906.9182567 | 48.3484411811 | 5.63 | 0s:9ms |
| 126 | 868.7452097 | 919.1873186 | 50.4421088463 | 5.81 | 0s:9ms |
| 127 | 878.9113552 | 929.1894792 | 50.2781240664 | 5.72 | 0s:9ms |
| 128 | 889.4083279 | 940.6625452 | 51.2542173032 | 5.76 | 0s:9ms |
| 129 | 899.3794711 | 939.0402908 | 39.6608196933 | 4.41 | 0s:10ms |
| 130 | 910.0149851 | 956.4974894 | 46.4825042620 | 5.11 | 0s:9ms |
| 131 | 920.3565478 | 964.284437 | 43.9278891473 | 4.77 | 0s:9ms |
| 132 | 931.0037789 | 978.8995627 | 47.8957838592 | 5.14 | 0s:10ms |
| 133 | 941.572814 | 992.651323 | 51.0785089994 | 5.42 | 0s:10ms |

Tabel B.2: Packomania $r_i = i$

| N | Beste radius | Radius | Vergroting | Vergroting (%) | Tijd |
|-----|--------------|-------------|---------------|----------------|---------|
| 134 | 951.4867325 | 999.1692947 | 47.6825622415 | 5.01 | 0s:10ms |
| 135 | 962.6267783 | 1011.670356 | 49.0435779611 | 5.09 | 0s:10ms |
| 136 | 973.5501094 | 1023.408578 | 49.8584681961 | 5.12 | 0s:11ms |
| 137 | 983.6545581 | 1039.839059 | 56.1845006701 | 5.71 | 0s:11ms |
| 138 | 994.589987 | 1046.579775 | 51.9897882140 | 5.23 | 0s:11ms |
| 139 | 1005.134043 | 1060.799299 | 55.6652552216 | 5.54 | 0s:11ms |
| 140 | 1016.243955 | 1066.391406 | 50.1474504792 | 4.93 | 0s:11ms |
| 141 | 1027.012735 | 1075.826455 | 48.8137198003 | 4.75 | 0s:11ms |
| 142 | 1038.033343 | 1091.535835 | 53.5024911515 | 5.15 | 0s:12ms |
| 143 | 1048.086632 | 1105.062108 | 56.9754760099 | 5.44 | 0s:11ms |
| 144 | 1059.610594 | 1118.135792 | 58.5251983013 | 5.52 | 0s:11ms |
| 145 | 1070.299995 | 1128.198594 | 57.8985985688 | 5.41 | 0s:11ms |
| 146 | 1081.849804 | 1137.382853 | 55.5330488821 | 5.13 | 0s:12ms |
| 147 | 1092.224839 | 1146.180093 | 53.9552534597 | 4.94 | 0s:11ms |
| 148 | 1103.625998 | 1157.325822 | 53.6998239456 | 4.87 | 0s:12ms |
| 149 | 1114.562888 | 1170.783856 | 56.2209683647 | 5.04 | 0s:12ms |
| 150 | 1125.962247 | 1179.834746 | 53.8724990106 | 4.78 | 0s:12ms |
| 151 | 1136.450736 | 1188.206974 | 51.7562382973 | 4.55 | 0s:12ms |
| 152 | 1148.381201 | 1198.503306 | 50.1221048286 | 4.36 | 0s:12ms |
| 153 | 1159.288108 | 1211.469481 | 52.1813720335 | 4.50 | 0s:13ms |
| 154 | 1170.008519 | 1221.771848 | 51.7633292913 | 4.42 | 0s:12ms |
| 155 | 1182.306413 | 1232.750873 | 50.4444602860 | 4.27 | 0s:12ms |
| 156 | 1193.219558 | 1241.96315 | 48.7435918571 | 4.09 | 0s:13ms |
| 157 | 1205.242985 | 1257.673634 | 52.4306489161 | 4.35 | 0s:13ms |
| 158 | 1216.514329 | 1272.623712 | 56.1093833890 | 4.61 | 0s:13ms |
| 159 | 1227.02294 | 1287.046839 | 60.0238985772 | 4.89 | 0s:14ms |
| 160 | 1238.93349 | 1297.811321 | 58.8778314960 | 4.75 | 0s:13ms |
| 161 | 1251.042708 | 1309.194957 | 58.1522496450 | 4.65 | 0s:13ms |
| 162 | 1262.789645 | 1325.157536 | 62.3678910466 | 4.94 | 0s:14ms |
| 163 | 1273.875363 | 1340.254659 | 66.3792957824 | 5.21 | 0s:13ms |
| 164 | 1286.025828 | 1355.126621 | 69.1007934484 | 5.37 | 0s:14ms |
| 165 | 1297.507389 | 1360.843465 | 63.3360754563 | 4.88 | 0s:14ms |
| 166 | 1309.139789 | 1372.652535 | 63.5127460293 | 4.85 | 0s:14ms |
| 167 | 1321.053819 | 1380.367699 | 59.3138806314 | 4.49 | 0s:14ms |
| 168 | 1332.790901 | 1396.706975 | 63.9160738570 | 4.80 | 0s:14ms |
| 169 | 1344.525206 | 1408.899427 | 64.3742216180 | 4.79 | 0s:14ms |
| 170 | 1356.179054 | 1421.473252 | 65.2941986483 | 4.81 | 0s:17ms |
| 171 | 1368.338575 | 1436.819503 | 68.4809275612 | 5.00 | 0s:15ms |
| 172 | 1380.333616 | 1443.901427 | 63.5678107594 | 4.61 | 0s:15ms |
| 173 | 1391.879458 | 1456.325953 | 64.4464957907 | 4.63 | 0s:15ms |
| 174 | 1403.566254 | 1471.496936 | 67.9306813422 | 4.84 | 0s:15ms |
| 175 | 1415.989735 | 1485.609411 | 69.6196752979 | 4.92 | 0s:15ms |
| 176 | 1428.497598 | 1498.150763 | 69.6531648576 | 4.88 | 0s:15ms |

Tabel B.2: Packomania $r_i = i$

| N | Beste radius | Radius | Vergroting | Vergroting (%) | Tijd |
|-----|--------------|-------------|---------------|----------------|---------|
| 177 | 1440.624114 | 1520.601441 | 79.9773270606 | 5.55 | 0s:15ms |
| 178 | 1452.757272 | 1533.313102 | 80.5558307306 | 5.55 | 0s:16ms |
| 179 | 1464.812903 | 1544.279674 | 79.4667714894 | 5.43 | 0s:15ms |
| 180 | 1476.565762 | 1549.253422 | 72.6876600373 | 4.92 | 0s:16ms |
| 181 | 1489.017452 | 1567.397547 | 78.3800946877 | 5.26 | 0s:16ms |
| 182 | 1501.087384 | 1582.428192 | 81.3408078754 | 5.42 | 0s:17ms |
| 183 | 1514.01682 | 1586.179083 | 72.1622628565 | 4.77 | 0s:16ms |
| 184 | 1525.885179 | 1608.260257 | 82.3750775373 | 5.40 | 0s:16ms |
| 185 | 1537.376408 | 1619.244541 | 81.8681332884 | 5.33 | 0s:16ms |
| 186 | 1550.809164 | 1630.783013 | 79.9738492793 | 5.16 | 0s:16ms |
| 187 | 1563.028446 | 1641.222098 | 78.1936519827 | 5.00 | 0s:17ms |
| 188 | 1575.802178 | 1658.869995 | 83.0678169214 | 5.27 | 0s:18ms |
| 189 | 1588.346875 | 1670.718704 | 82.3718286801 | 5.19 | 0s:17ms |
| 190 | 1600.66992 | 1684.525388 | 83.8554676059 | 5.24 | 0s:17ms |
| 191 | 1613.286429 | 1691.729998 | 78.4435689224 | 4.86 | 0s:17ms |
| 192 | 1626.176015 | 1707.916237 | 81.7402226434 | 5.03 | 0s:18ms |
| 193 | 1638.978104 | 1722.336844 | 83.3587400976 | 5.09 | 0s:18ms |
| 194 | 1651.184614 | 1733.930788 | 82.7461736083 | 5.01 | 0s:18ms |
| 195 | 1663.511587 | 1742.176526 | 78.6649392823 | 4.73 | 0s:18ms |
| 196 | 1676.838498 | 1757.798572 | 80.9600734542 | 4.83 | 0s:17ms |
| 197 | 1688.921565 | 1770.904998 | 81.9834328647 | 4.85 | 0s:18ms |
| 198 | 1701.700662 | 1782.097388 | 80.3967258220 | 4.72 | 0s:18ms |
| 199 | 1714.696039 | 1797.645421 | 82.9493820303 | 4.84 | 0s:19ms |
| 200 | 1726.221913 | 1807.004915 | 80.7830024141 | 4.68 | 0s:20ms |

Tabel B.3: Packomania $r_i = i^{1/2}$

| N | Beste radius | Radius | Vergroting | Vergroting (%) | Tijd |
|----|--------------|-------------|--------------|----------------|--------|
| 5 | 4.521480277 | 4.645331059 | 0.1238507820 | 2.74 | 0s:0ms |
| 6 | 5.35096299 | 5.565200476 | 0.2142374858 | 4.00 | 0s:0ms |
| 7 | 6.049378486 | 6.313171861 | 0.2637933742 | 4.36 | 0s:0ms |
| 8 | 6.774266652 | 7.011200784 | 0.2369341324 | 3.50 | 0s:0ms |
| 9 | 7.559002377 | 7.645002451 | 0.0860000737 | 1.14 | 0s:0ms |
| 10 | 8.303468122 | 8.773595833 | 0.4701277110 | 5.66 | 0s:0ms |
| 11 | 9.072125879 | 9.564217949 | 0.4920920696 | 5.42 | 0s:0ms |
| 12 | 9.865320304 | 10.47459566 | 0.6092753519 | 6.18 | 0s:0ms |
| 13 | 10.58832629 | 11.28988685 | 0.7015605668 | 6.63 | 0s:0ms |
| 14 | 11.36497759 | 12.03662293 | 0.6716453426 | 5.91 | 0s:0ms |
| 15 | 12.0755266 | 12.73140124 | 0.6558746339 | 5.43 | 0s:0ms |
| 16 | 12.81931152 | 13.54360586 | 0.7242943423 | 5.65 | 0s:0ms |
| 17 | 13.58212394 | 14.56084288 | 0.9787189360 | 7.21 | 0s:0ms |
| 18 | 14.32496989 | 15.462612 | 1.1376421126 | 7.94 | 0s:0ms |
| 19 | 15.03535243 | 16.1728907 | 1.1375382697 | 7.57 | 0s:0ms |

Tabel B.3: Packomania $r_i = i^{1/2}$

| N | Beste radius | Radius | Vergroting | Vergroting (%) | Tijd |
|----|--------------|-------------|--------------|----------------|--------|
| 20 | 15.80237882 | 16.99625392 | 1.1938751032 | 7.56 | 0s:0ms |
| 21 | 16.55930116 | 17.78557896 | 1.2262778069 | 7.41 | 0s:0ms |
| 22 | 17.29396452 | 18.52909152 | 1.2351270012 | 7.14 | 0s:0ms |
| 23 | 18.0499852 | 19.23530642 | 1.1853212204 | 6.57 | 0s:0ms |
| 24 | 18.79875241 | 19.91971384 | 1.1209614357 | 5.96 | 0s:0ms |
| 25 | 19.5506756 | 20.66318875 | 1.1125131521 | 5.69 | 0s:0ms |
| 26 | 20.2902618 | 22.23042619 | 1.9401643933 | 9.56 | 0s:0ms |
| 27 | 21.06193173 | 22.9880056 | 1.9260738751 | 9.14 | 0s:1ms |
| 28 | 21.79712527 | 23.71343628 | 1.9163110120 | 8.79 | 0s:1ms |
| 29 | 22.54735751 | 24.41115787 | 1.8638003585 | 8.27 | 0s:1ms |
| 30 | 23.25868018 | 25.19760804 | 1.9389278525 | 8.34 | 0s:1ms |
| 31 | 24.03727193 | 25.81304012 | 1.7757681886 | 7.39 | 0s:1ms |
| 32 | 24.78200903 | 26.41001473 | 1.6280056998 | 6.57 | 0s:1ms |
| 33 | 25.54486052 | 27.14270567 | 1.5978451512 | 6.26 | 0s:1ms |
| 34 | 26.30459206 | 28.23676513 | 1.9321730714 | 7.35 | 0s:1ms |
| 35 | 27.03382245 | 29.1675068 | 2.1336843460 | 7.89 | 0s:1ms |
| 36 | 27.77995711 | 29.74228203 | 1.9623249292 | 7.06 | 0s:1ms |
| 37 | 28.52327223 | 30.55237025 | 2.0290980255 | 7.11 | 0s:1ms |
| 38 | 29.2801575 | 31.32911627 | 2.0489587647 | 7.00 | 0s:1ms |
| 39 | 30.02551081 | 32.07750685 | 2.0519960334 | 6.83 | 0s:1ms |
| 40 | 30.75824676 | 32.80128099 | 2.0430342306 | 6.64 | 0s:1ms |
| 41 | 31.5074417 | 33.67925509 | 2.1718133896 | 6.89 | 0s:1ms |
| 42 | 32.27253417 | 34.41887774 | 2.1463435677 | 6.65 | 0s:2ms |
| 43 | 33.01068077 | 35.03290206 | 2.0222212869 | 6.13 | 0s:2ms |
| 44 | 33.76125465 | 35.99501679 | 2.2337621450 | 6.62 | 0s:2ms |
| 45 | 34.52293182 | 36.88407979 | 2.3611479634 | 6.84 | 0s:2ms |
| 46 | 35.24429078 | 37.49808526 | 2.2537944845 | 6.39 | 0s:2ms |
| 47 | 35.99250776 | 38.21813911 | 2.2256313508 | 6.18 | 0s:2ms |
| 48 | 36.76013222 | 38.92008922 | 2.1599570026 | 5.88 | 0s:2ms |
| 49 | 37.46753983 | 39.73435907 | 2.2668192423 | 6.05 | 0s:2ms |
| 50 | 38.25556972 | 40.53178138 | 2.2762116663 | 5.95 | 0s:2ms |
| 51 | 38.98768932 | 42.15091723 | 3.1632279036 | 8.11 | 0s:2ms |
| 52 | 39.74306968 | 42.89795161 | 3.1548819271 | 7.94 | 0s:2ms |
| 53 | 40.48968483 | 43.52039716 | 3.0307123211 | 7.49 | 0s:2ms |
| 54 | 41.23581075 | 44.15285577 | 2.9170450216 | 7.07 | 0s:3ms |
| 55 | 41.98271338 | 45.00105715 | 3.0183437706 | 7.19 | 0s:3ms |
| 56 | 42.72931327 | 45.64420809 | 2.9148948170 | 6.82 | 0s:3ms |
| 57 | 43.50080653 | 46.45103577 | 2.9502292405 | 6.78 | 0s:3ms |
| 58 | 44.22194556 | 47.13057987 | 2.9086343035 | 6.58 | 0s:3ms |
| 59 | 44.9825593 | 48.18944065 | 3.2068813482 | 7.13 | 0s:3ms |
| 60 | 45.70598269 | 48.92807634 | 3.2220936565 | 7.05 | 0s:3ms |
| 61 | 46.47167529 | 50.12902499 | 3.6573496935 | 7.87 | 0s:3ms |
| 62 | 47.19034394 | 50.84643939 | 3.6560954536 | 7.75 | 0s:4ms |

Tabel B.3: Packomania $r_i = i^{1/2}$

| N | Beste radius | Radius | Vergroting | Vergroting (%) | Tijd |
|-----|--------------|-------------|--------------|----------------|--------|
| 63 | 47.96038659 | 51.56087869 | 3.6004921000 | 7.51 | 0s:4ms |
| 64 | 48.70530574 | 51.94948876 | 3.2441830170 | 6.66 | 0s:4ms |
| 65 | 49.42245241 | 52.65486666 | 3.2324142527 | 6.54 | 0s:4ms |
| 66 | 50.18742036 | 53.35552985 | 3.1681094893 | 6.31 | 0s:4ms |
| 67 | 50.90373739 | 54.0422335 | 3.1384961055 | 6.17 | 0s:4ms |
| 68 | 51.64886468 | 54.85582673 | 3.2069620523 | 6.21 | 0s:5ms |
| 69 | 52.44298308 | 55.77600995 | 3.3330268692 | 6.36 | 0s:5ms |
| 70 | 53.18043899 | 56.43020596 | 3.2497669710 | 6.11 | 0s:4ms |
| 71 | 53.93496914 | 57.13993526 | 3.2049661248 | 5.94 | 0s:5ms |
| 72 | 54.65674105 | 57.98591706 | 3.3291760153 | 6.09 | 0s:5ms |
| 73 | 55.42818864 | 58.92061484 | 3.4924262054 | 6.30 | 0s:5ms |
| 74 | 56.17979032 | 59.38454976 | 3.2047594461 | 5.70 | 0s:5ms |
| 75 | 56.88186606 | 60.37897148 | 3.4971054144 | 6.15 | 0s:6ms |
| 76 | 57.65288799 | 61.03664465 | 3.3837566627 | 5.87 | 0s:5ms |
| 77 | 58.42166044 | 61.73259783 | 3.3109373885 | 5.67 | 0s:5ms |
| 78 | 59.1476465 | 62.57629857 | 3.4286520715 | 5.80 | 0s:6ms |
| 79 | 59.86847145 | 63.39927675 | 3.5308053035 | 5.90 | 0s:6ms |
| 80 | 60.63356054 | 64.2854286 | 3.6518680539 | 6.02 | 0s:6ms |
| 81 | 61.38066647 | 65.02500355 | 3.6443370809 | 5.94 | 0s:6ms |
| 82 | 62.10506478 | 65.65078433 | 3.5457195519 | 5.71 | 0s:6ms |
| 83 | 62.88171517 | 66.23500237 | 3.3532871953 | 5.33 | 0s:6ms |
| 84 | 63.62031086 | 67.45460619 | 3.8342953230 | 6.03 | 0s:6ms |
| 85 | 64.37295794 | 68.32063478 | 3.9476768349 | 6.13 | 0s:6ms |
| 86 | 65.10351537 | 68.97796537 | 3.8744500057 | 5.95 | 0s:6ms |
| 87 | 65.85511151 | 69.71427598 | 3.8591644643 | 5.86 | 0s:6ms |
| 88 | 66.52181857 | 70.94485009 | 4.4230315140 | 6.65 | 0s:6ms |
| 89 | 67.3413989 | 71.79373861 | 4.4523397045 | 6.61 | 0s:7ms |
| 90 | 68.08292742 | 72.11066047 | 4.0277330422 | 5.92 | 0s:7ms |
| 91 | 68.83792287 | 72.81415128 | 3.9762284068 | 5.78 | 0s:7ms |
| 92 | 69.58671058 | 73.49111243 | 3.9044018523 | 5.61 | 0s:7ms |
| 93 | 70.31569892 | 74.27702903 | 3.9613301133 | 5.63 | 0s:7ms |
| 94 | 71.03023084 | 75.33180092 | 4.3015700839 | 6.06 | 0s:8ms |
| 95 | 71.82403089 | 76.50249183 | 4.6784609402 | 6.51 | 0s:8ms |
| 96 | 72.54565607 | 77.36198139 | 4.8163253140 | 6.64 | 0s:7ms |
| 97 | 73.30173645 | 78.05100651 | 4.7492700614 | 6.48 | 0s:7ms |
| 98 | 74.06394605 | 78.39015055 | 4.3262045063 | 5.84 | 0s:8ms |
| 99 | 74.80086926 | 79.06946294 | 4.2685936749 | 5.71 | 0s:8ms |
| 100 | 75.54699422 | 79.96233173 | 4.4153375112 | 5.84 | 0s:8ms |

Tabel B.4: Packomania $r_i = i^{-1/5}$

| N | Beste radius | Radius | Vergroting | Vergroting (%) | Tijd |
|----|--------------|-------------|--------------|----------------|--------|
| 5 | 2.244615846 | 2.417764166 | 0.1731483194 | 7.71 | 0s:0ms |
| 6 | 2.387986383 | 2.490641404 | 0.1026550210 | 4.30 | 0s:0ms |
| 7 | 2.422623343 | 2.541024002 | 0.1184006594 | 4.89 | 0s:0ms |
| 8 | 2.523820904 | 2.802493413 | 0.2786725086 | 11.04 | 0s:0ms |
| 9 | 2.630026572 | 2.893082814 | 0.2630562428 | 10.00 | 0s:0ms |
| 10 | 2.715784826 | 3.043778646 | 0.3279938201 | 12.08 | 0s:0ms |
| 11 | 2.767466306 | 3.06381508 | 0.2963487739 | 10.71 | 0s:0ms |
| 12 | 2.829018403 | 3.06381508 | 0.2347966767 | 8.30 | 0s:0ms |
| 13 | 2.92391778 | 3.146230401 | 0.2223126209 | 7.60 | 0s:0ms |
| 14 | 2.984783969 | 3.31986946 | 0.3350854907 | 11.23 | 0s:0ms |
| 15 | 3.040492052 | 3.31986946 | 0.2793774083 | 9.19 | 0s:0ms |
| 16 | 3.096414432 | 3.33097666 | 0.2345622289 | 7.58 | 0s:0ms |
| 17 | 3.15212915 | 3.438918964 | 0.2867898134 | 9.10 | 0s:0ms |
| 18 | 3.214823783 | 3.529068281 | 0.3142444981 | 9.77 | 0s:0ms |
| 19 | 3.259332781 | 3.574355458 | 0.3150226770 | 9.67 | 0s:0ms |
| 20 | 3.311831017 | 3.595528088 | 0.2836970703 | 8.57 | 0s:0ms |
| 21 | 3.36159381 | 3.813698731 | 0.4521049215 | 13.45 | 0s:0ms |
| 22 | 3.411451256 | 3.813698731 | 0.4022474751 | 11.79 | 0s:0ms |
| 23 | 3.450137359 | 3.826191897 | 0.3760545382 | 10.90 | 0s:0ms |
| 24 | 3.502510278 | 3.849500792 | 0.3469905142 | 9.91 | 0s:0ms |
| 25 | 3.545245578 | 3.849500792 | 0.3042552139 | 8.58 | 0s:0ms |
| 26 | 3.591133783 | 3.969766564 | 0.3786327815 | 10.54 | 0s:1ms |
| 27 | 3.632687191 | 4.057652417 | 0.4249652256 | 11.70 | 0s:1ms |
| 28 | 3.673947341 | 4.057652417 | 0.3837050759 | 10.44 | 0s:1ms |
| 29 | 3.709614743 | 4.102992738 | 0.3933779955 | 10.60 | 0s:1ms |
| 30 | 3.753020534 | 4.102992738 | 0.3499722044 | 9.33 | 0s:1ms |
| 31 | 3.794636899 | 4.100467051 | 0.3058301526 | 8.06 | 0s:1ms |
| 32 | 3.826360682 | 4.161242705 | 0.3348820228 | 8.75 | 0s:1ms |
| 33 | 3.866024486 | 4.161242705 | 0.2952182183 | 7.64 | 0s:1ms |
| 34 | 3.901727549 | 4.27545313 | 0.3737255803 | 9.58 | 0s:1ms |
| 35 | 3.936688768 | 4.291727438 | 0.3550386705 | 9.02 | 0s:1ms |
| 36 | 3.969717962 | 4.308499631 | 0.3387816690 | 8.53 | 0s:1ms |
| 37 | 4.006158224 | 4.360397929 | 0.3542397051 | 8.84 | 0s:1ms |
| 38 | 4.038554895 | 4.363613142 | 0.3250582469 | 8.05 | 0s:1ms |
| 39 | 4.074618568 | 4.457940853 | 0.3833222845 | 9.41 | 0s:1ms |
| 40 | 4.105672784 | 4.500185046 | 0.3945122618 | 9.61 | 0s:2ms |
| 41 | 4.129990611 | 4.500185046 | 0.3701944350 | 8.96 | 0s:2ms |
| 42 | 4.170056305 | 4.516843613 | 0.3467873080 | 8.32 | 0s:2ms |
| 43 | 4.196545217 | 4.556691854 | 0.3601466376 | 8.58 | 0s:2ms |
| 44 | 4.230014441 | 4.556691854 | 0.3266774127 | 7.72 | 0s:2ms |
| 45 | 4.258819887 | 4.559478152 | 0.3006582655 | 7.06 | 0s:2ms |
| 46 | 4.279963899 | 4.578989471 | 0.2990255718 | 6.99 | 0s:2ms |
| 47 | 4.31934031 | 4.684566353 | 0.3652260424 | 8.46 | 0s:2ms |

Tabel B.4: Packomania $r_i = i^{-1/5}$

| N | Beste radius | Radius | Vergroting | Vergroting (%) | Tijd |
|----|--------------|-------------|--------------|----------------|--------|
| 48 | 4.340298255 | 4.718254714 | 0.3779564595 | 8.71 | 0s:2ms |
| 49 | 4.369922887 | 4.718254714 | 0.3483318277 | 7.97 | 0s:2ms |
| 50 | 4.398199789 | 4.718254714 | 0.3200549252 | 7.28 | 0s:2ms |
| 51 | 4.421005635 | 4.789771253 | 0.3687656172 | 8.34 | 0s:3ms |
| 52 | 4.457103013 | 4.79446383 | 0.3373608175 | 7.57 | 0s:3ms |
| 53 | 4.481095278 | 4.88401277 | 0.4029174922 | 8.99 | 0s:3ms |
| 54 | 4.507233924 | 4.94871256 | 0.4414786359 | 9.79 | 0s:3ms |
| 55 | 4.533359692 | 4.94871256 | 0.4153528680 | 9.16 | 0s:3ms |
| 56 | 4.552402168 | 4.94871256 | 0.3963103920 | 8.71 | 0s:3ms |
| 57 | 4.583331939 | 4.960637843 | 0.3773059034 | 8.23 | 0s:3ms |
| 58 | 4.604279546 | 5.006194354 | 0.4019148082 | 8.73 | 0s:3ms |
| 59 | 4.633931987 | 5.015140166 | 0.3812081781 | 8.23 | 0s:3ms |
| 60 | 4.661004393 | 5.039668431 | 0.3786640380 | 8.12 | 0s:3ms |
| 61 | 4.67975982 | 5.039668431 | 0.3599086110 | 7.69 | 0s:3ms |
| 62 | 4.703936871 | 5.08559604 | 0.3816591685 | 8.11 | 0s:4ms |
| 63 | 4.734693491 | 5.08559604 | 0.3509025491 | 7.41 | 0s:4ms |
| 64 | 4.754788576 | 5.09503129 | 0.3402427132 | 7.16 | 0s:4ms |

Tabel B.5: Packomania $r_i = i^{-1/2}$

| N | Beste radius | Radius | Vergroting | Vergroting (%) | Tijd |
|----|--------------|-------------|--------------|----------------|--------|
| 5 | 1.751552455 | 1.802472656 | 0.0509202011 | 2.91 | 0s:0ms |
| 6 | 1.810076939 | 1.873533983 | 0.0634570432 | 3.51 | 0s:0ms |
| 7 | 1.838724068 | 1.963882951 | 0.1251588836 | 6.81 | 0s:0ms |
| 8 | 1.858400955 | 1.963882951 | 0.1054819968 | 5.68 | 0s:0ms |
| 9 | 1.878812755 | 2.003759856 | 0.1249471003 | 6.65 | 0s:0ms |
| 10 | 1.913435515 | 2.054440116 | 0.1410046004 | 7.37 | 0s:0ms |
| 11 | 1.929187751 | 2.055014716 | 0.1258269651 | 6.52 | 0s:0ms |
| 12 | 1.949823437 | 2.0660242 | 0.1162007622 | 5.96 | 0s:0ms |
| 13 | 1.965236819 | 2.113388316 | 0.1481514969 | 7.54 | 0s:0ms |
| 14 | 1.980248748 | 2.14077957 | 0.1605308220 | 8.11 | 0s:0ms |
| 15 | 1.992709274 | 2.14077957 | 0.1480702960 | 7.43 | 0s:0ms |
| 16 | 2.004585778 | 2.140794139 | 0.1362083606 | 6.79 | 0s:0ms |
| 17 | 2.015257783 | 2.140794139 | 0.1255363550 | 6.23 | 0s:0ms |
| 18 | 2.028148582 | 2.161557565 | 0.1334089823 | 6.58 | 0s:0ms |
| 19 | 2.041997315 | 2.189486219 | 0.1474889039 | 7.22 | 0s:0ms |
| 20 | 2.051442268 | 2.195567792 | 0.1441255240 | 7.03 | 0s:0ms |
| 21 | 2.062331108 | 2.195567792 | 0.1332366847 | 6.46 | 0s:0ms |
| 22 | 2.06796317 | 2.195567792 | 0.1276046224 | 6.17 | 0s:0ms |
| 23 | 2.079772368 | 2.195567792 | 0.1157954243 | 5.57 | 0s:0ms |
| 24 | 2.090383164 | 2.212125838 | 0.1217426733 | 5.82 | 0s:0ms |
| 25 | 2.097527836 | 2.212125838 | 0.1145980019 | 5.46 | 0s:0ms |
| 26 | 2.107630171 | 2.212125838 | 0.1044956670 | 4.96 | 0s:0ms |

Tabel B.5: Packomania $r_i = i^{-1/2}$

| N | Beste radius | Radius | Vergroting | Vergroting (%) | Tijd |
|----|--------------|-------------|--------------|----------------|--------|
| 27 | 2.115890492 | 2.212125838 | 0.0962353458 | 4.55 | 0s:1ms |
| 28 | 2.122849176 | 2.25561005 | 0.1327608737 | 6.25 | 0s:1ms |
| 29 | 2.13197913 | 2.26989985 | 0.1379207195 | 6.47 | 0s:1ms |
| 30 | 2.137483926 | 2.26989985 | 0.1324159233 | 6.19 | 0s:1ms |
| 31 | 2.148983307 | 2.272391577 | 0.1234082702 | 5.74 | 0s:1ms |
| 32 | 2.155354151 | 2.272391577 | 0.1170374256 | 5.43 | 0s:1ms |
| 33 | 2.165803682 | 2.300760835 | 0.1349571532 | 6.23 | 0s:1ms |
| 34 | 2.174364112 | 2.300760835 | 0.1263967230 | 5.81 | 0s:1ms |
| 35 | 2.171162102 | 2.303126269 | 0.1319641666 | 6.08 | 0s:1ms |
| 36 | 2.184151152 | 2.31172537 | 0.1275742188 | 5.84 | 0s:1ms |
| 37 | 2.190324008 | 2.31172537 | 0.1214013622 | 5.54 | 0s:1ms |
| 38 | 2.200283193 | 2.327362681 | 0.1270794878 | 5.78 | 0s:1ms |
| 39 | 2.207438262 | 2.327362681 | 0.1199244191 | 5.43 | 0s:2ms |
| 40 | 2.215788047 | 2.336852047 | 0.1210639995 | 5.46 | 0s:2ms |
| 41 | 2.221496388 | 2.336852047 | 0.1153556592 | 5.19 | 0s:2ms |
| 42 | 2.225063517 | 2.348630674 | 0.1235671577 | 5.55 | 0s:2ms |
| 43 | 2.225967367 | 2.348630674 | 0.1226633073 | 5.51 | 0s:2ms |
| 44 | 2.22976905 | 2.348630674 | 0.1188616248 | 5.33 | 0s:2ms |
| 45 | 2.235330122 | 2.348630674 | 0.1133005522 | 5.07 | 0s:2ms |
| 46 | 2.24069414 | 2.348630674 | 0.1079365346 | 4.82 | 0s:2ms |
| 47 | 2.251179399 | 2.348630674 | 0.0974512759 | 4.33 | 0s:2ms |
| 48 | 2.2546717 | 2.348630674 | 0.0939589745 | 4.17 | 0s:3ms |
| 49 | 2.259576657 | 2.348630674 | 0.0890540172 | 3.94 | 0s:3ms |
| 50 | 2.263343706 | 2.348630674 | 0.0852869681 | 3.77 | 0s:3ms |
| 51 | 2.270868233 | 2.361262988 | 0.0903947555 | 3.98 | 0s:3ms |
| 52 | 2.279267038 | 2.367760412 | 0.0884933744 | 3.88 | 0s:3ms |
| 53 | 2.281766324 | 2.367760412 | 0.0859940880 | 3.77 | 0s:3ms |
| 54 | 2.284519164 | 2.381915573 | 0.0973964091 | 4.26 | 0s:7ms |
| 55 | 2.285947292 | 2.383596802 | 0.0976495093 | 4.27 | 0s:4ms |
| 56 | 2.285979541 | 2.383596802 | 0.0976172611 | 4.27 | 0s:4ms |
| 57 | 2.288534442 | 2.384328728 | 0.0957942864 | 4.19 | 0s:4ms |
| 58 | 2.292664714 | 2.385305153 | 0.0926404388 | 4.04 | 0s:4ms |
| 59 | 2.296297312 | 2.38689011 | 0.0905927983 | 3.95 | 0s:4ms |
| 60 | 2.306501919 | 2.407776328 | 0.1012744095 | 4.39 | 0s:4ms |
| 61 | 2.311319628 | 2.409449022 | 0.0981293940 | 4.25 | 0s:4ms |
| 62 | 2.317481916 | 2.409449022 | 0.0919671063 | 3.97 | 0s:4ms |
| 63 | 2.318797322 | 2.409449022 | 0.0906517004 | 3.91 | 0s:4ms |
| 64 | 2.320980355 | 2.409449022 | 0.0884686673 | 3.81 | 0s:5ms |
| 65 | 2.321469818 | 2.413031572 | 0.0915617534 | 3.94 | 0s:5ms |
| 66 | 2.324362904 | 2.416403418 | 0.0920405139 | 3.96 | 0s:5ms |
| 67 | 2.332217263 | 2.416403418 | 0.0841861545 | 3.61 | 0s:5ms |
| 68 | 2.334282511 | 2.416403418 | 0.0821209072 | 3.52 | 0s:5ms |
| 69 | 2.341094038 | 2.416403418 | 0.0753093795 | 3.22 | 0s:5ms |

Tabel B.5: Packomania $r_i = i^{-1/2}$

| N | Beste radius | Radius | Vergroting | Vergroting (%) | Tijd |
|-----|--------------|-------------|--------------|----------------|---------|
| 70 | 2.344495311 | 2.41743353 | 0.0729382190 | 3.11 | 0s:5ms |
| 71 | 2.348186028 | 2.434594573 | 0.0864085445 | 3.68 | 0s:6ms |
| 72 | 2.348822083 | 2.434594573 | 0.0857724897 | 3.65 | 0s:6ms |
| 73 | 2.347237572 | 2.434594573 | 0.0873570010 | 3.72 | 0s:7ms |
| 74 | 2.35004496 | 2.439743433 | 0.0896984733 | 3.82 | 0s:6ms |
| 75 | 2.352202465 | 2.439743433 | 0.0875409681 | 3.72 | 0s:7ms |
| 76 | 2.352948629 | 2.446211033 | 0.0932624035 | 3.96 | 0s:7ms |
| 77 | 2.357614973 | 2.458852324 | 0.1012373513 | 4.29 | 0s:8ms |
| 78 | 2.354416639 | 2.458852324 | 0.1044356856 | 4.44 | 0s:7ms |
| 79 | 2.355232406 | 2.458852324 | 0.1036199180 | 4.40 | 0s:7ms |
| 80 | 2.35721057 | 2.458852324 | 0.1016417546 | 4.31 | 0s:7ms |
| 81 | 2.358864905 | 2.462287263 | 0.1034223576 | 4.38 | 0s:9ms |
| 82 | 2.362221053 | 2.462287263 | 0.1000662097 | 4.24 | 0s:9ms |
| 83 | 2.364586515 | 2.469562772 | 0.1049762569 | 4.44 | 0s:8ms |
| 84 | 2.367639497 | 2.470848575 | 0.1032090780 | 4.36 | 0s:8ms |
| 85 | 2.369100792 | 2.477635185 | 0.1085343927 | 4.58 | 0s:9ms |
| 86 | 2.374745075 | 2.477635185 | 0.1028901101 | 4.33 | 0s:9ms |
| 87 | 2.37769523 | 2.477635185 | 0.0999399553 | 4.20 | 0s:8ms |
| 88 | 2.378456936 | 2.477635185 | 0.0991782491 | 4.17 | 0s:10ms |
| 89 | 2.382644831 | 2.477635185 | 0.0949903541 | 3.99 | 0s:14ms |
| 90 | 2.388107932 | 2.477635185 | 0.0895272530 | 3.75 | 0s:9ms |
| 91 | 2.40120373 | 2.477635185 | 0.0764314548 | 3.18 | 0s:10ms |
| 92 | 2.40172684 | 2.477635185 | 0.0759083455 | 3.16 | 0s:10ms |
| 93 | 2.404023803 | 2.477635185 | 0.0736113818 | 3.06 | 0s:9ms |
| 94 | 2.404197382 | 2.477635185 | 0.0734378028 | 3.05 | 0s:9ms |
| 95 | 2.410782192 | 2.48978687 | 0.0790046782 | 3.28 | 0s:10ms |
| 96 | 2.413190402 | 2.490964995 | 0.0777745929 | 3.22 | 0s:10ms |
| 97 | 2.412677724 | 2.490964995 | 0.0782872702 | 3.24 | 0s:10ms |
| 98 | 2.414386149 | 2.497843145 | 0.0834569954 | 3.46 | 0s:10ms |
| 99 | 2.416192906 | 2.497843145 | 0.0816502382 | 3.38 | 0s:11ms |
| 100 | 2.426033793 | 2.495438263 | 0.0694044704 | 2.86 | 0s:11ms |

Tabel B.6: Packomania $r_i = i^{-2/3}$

| N | Beste radius | Radius | Vergroting | Vergroting (%) | Tijd |
|----|--------------|-------------|--------------|----------------|--------|
| 5 | 1.629960525 | 1.629960525 | 0.0000000000 | 0.00 | 0s:0ms |
| 6 | 1.629960525 | 1.629960525 | 0.0000000000 | 0.00 | 0s:0ms |
| 7 | 1.62997278 | 1.653697845 | 0.0237250648 | 1.46 | 0s:0ms |
| 8 | 1.631484075 | 1.654338063 | 0.0228539885 | 1.40 | 0s:0ms |
| 9 | 1.637863996 | 1.679866987 | 0.0420029910 | 2.56 | 0s:0ms |
| 10 | 1.646957238 | 1.679866987 | 0.0329097486 | 2.00 | 0s:0ms |
| 11 | 1.650313825 | 1.682192134 | 0.0318783092 | 1.93 | 0s:0ms |
| 12 | 1.656760259 | 1.683078469 | 0.0263182105 | 1.59 | 0s:0ms |

Tabel B.6: Packomania $r_i = i^{-2/3}$

| N | Beste radius | Radius | Vergroting | Vergroting (%) | Tijd |
|----|--------------|-------------|--------------|----------------|--------|
| 13 | 1.662778033 | 1.714281175 | 0.0515031423 | 3.10 | 0s:0ms |
| 14 | 1.670183528 | 1.729067653 | 0.0588841247 | 3.53 | 0s:0ms |
| 15 | 1.673001055 | 1.729067653 | 0.0560665977 | 3.35 | 0s:0ms |
| 16 | 1.679631815 | 1.73789466 | 0.0582628453 | 3.47 | 0s:0ms |
| 17 | 1.683840973 | 1.73789466 | 0.0540536868 | 3.21 | 0s:0ms |
| 18 | 1.686025242 | 1.738271955 | 0.0522467129 | 3.10 | 0s:0ms |
| 19 | 1.689532602 | 1.770039695 | 0.0805070930 | 4.77 | 0s:0ms |
| 20 | 1.696447725 | 1.770039695 | 0.0735919695 | 4.34 | 0s:0ms |
| 21 | 1.698943028 | 1.770039695 | 0.0710966671 | 4.18 | 0s:0ms |
| 22 | 1.700668034 | 1.770757259 | 0.0700892256 | 4.12 | 0s:0ms |
| 23 | 1.703933782 | 1.770757259 | 0.0668234774 | 3.92 | 0s:0ms |
| 24 | 1.705387705 | 1.770757259 | 0.0653695549 | 3.83 | 0s:0ms |
| 25 | 1.712330424 | 1.770757259 | 0.0584268354 | 3.41 | 0s:0ms |
| 26 | 1.714593298 | 1.771802524 | 0.0572092258 | 3.34 | 0s:0ms |
| 27 | 1.718985638 | 1.785339316 | 0.0663536785 | 3.86 | 0s:1ms |
| 28 | 1.721909048 | 1.785339316 | 0.0634302683 | 3.68 | 0s:1ms |
| 29 | 1.722427772 | 1.785339316 | 0.0629115448 | 3.65 | 0s:1ms |
| 30 | 1.725040754 | 1.785339316 | 0.0602985628 | 3.50 | 0s:1ms |
| 31 | 1.72537599 | 1.785339316 | 0.0599633265 | 3.48 | 0s:1ms |
| 32 | 1.727084673 | 1.785339316 | 0.0582546429 | 3.37 | 0s:1ms |
| 33 | 1.731890808 | 1.785339316 | 0.0534485086 | 3.09 | 0s:1ms |
| 34 | 1.732984603 | 1.785339316 | 0.0523547133 | 3.02 | 0s:1ms |
| 35 | 1.733788081 | 1.785339316 | 0.0515512354 | 2.97 | 0s:1ms |
| 36 | 1.733942525 | 1.785339316 | 0.0513967910 | 2.96 | 0s:1ms |
| 37 | 1.73433911 | 1.785480739 | 0.0511416288 | 2.95 | 0s:1ms |
| 38 | 1.736647727 | 1.785480739 | 0.0488330114 | 2.81 | 0s:1ms |
| 39 | 1.740163339 | 1.793653228 | 0.0534898894 | 3.07 | 0s:1ms |
| 40 | 1.740440323 | 1.793653228 | 0.0532129052 | 3.06 | 0s:1ms |
| 41 | 1.743049384 | 1.793653228 | 0.0506038436 | 2.90 | 0s:1ms |
| 42 | 1.74463862 | 1.793653228 | 0.0490146083 | 2.81 | 0s:1ms |
| 43 | 1.746050874 | 1.793653228 | 0.0476023541 | 2.73 | 0s:2ms |
| 44 | 1.747310651 | 1.793653228 | 0.0463425767 | 2.65 | 0s:2ms |
| 45 | 1.74702235 | 1.793653228 | 0.0466308777 | 2.67 | 0s:2ms |
| 46 | 1.747830527 | 1.793653228 | 0.0458227015 | 2.62 | 0s:2ms |
| 47 | 1.750876036 | 1.793653228 | 0.0427771925 | 2.44 | 0s:2ms |
| 48 | 1.752745314 | 1.79723811 | 0.0444927958 | 2.54 | 0s:2ms |
| 49 | 1.753835619 | 1.79723811 | 0.0434024905 | 2.47 | 0s:2ms |
| 50 | 1.754493929 | 1.79723811 | 0.0427441809 | 2.44 | 0s:2ms |
| 51 | 1.754510736 | 1.79723811 | 0.0427273733 | 2.44 | 0s:2ms |
| 52 | 1.754548663 | 1.79723811 | 0.0426894470 | 2.43 | 0s:2ms |
| 53 | 1.757284994 | 1.797834156 | 0.0405491620 | 2.31 | 0s:3ms |
| 54 | 1.76113362 | 1.801421901 | 0.0402882810 | 2.29 | 0s:3ms |
| 55 | 1.768231856 | 1.801421901 | 0.0331900447 | 1.88 | 0s:3ms |

Tabel B.6: Packomania $r_i = i^{-2/3}$

| N | Beste radius | Radius | Vergroting | Vergroting (%) | Tijd |
|----|--------------|-------------|--------------|----------------|--------|
| 56 | Missing | 1.803122244 | | | 0s:3ms |
| 57 | 1.772557462 | 1.803122244 | 0.0305647820 | 1.72 | 0s:3ms |
| 58 | 1.772711083 | 1.803122244 | 0.0304111611 | 1.72 | 0s:3ms |
| 59 | 1.774022224 | 1.803122244 | 0.0291000203 | 1.64 | 0s:3ms |
| 60 | 1.772831554 | 1.806012683 | 0.0331811293 | 1.87 | 0s:3ms |

Opmerking: Een oplossing voor 56 cirkels, met raddii $r_i = i^{-2/3}$ miste op de Packomania website op het moment van het schrijven van deze thesis.

B.3 Packomania Benchmark problemen

Tabel B.7: Packomania Benchmark Instances

| Instantie | N | Beste radius | Radius | Vergroting | Vergroting (%) | Tijd |
|-----------|----|--------------|-------------|---------------|----------------|--------|
| NR10-1 | 10 | 99.88507689 | 105.2785764 | 5.3934994850 | 5.40 | 0s:0ms |
| NR11-1 | 11 | 60.70996139 | 64.21790153 | 3.5079401370 | 5.78 | 0s:1ms |
| NR12-1 | 12 | 65.02442247 | 69.5660027 | 4.5415802283 | 6.98 | 0s:0ms |
| NR14-1 | 14 | 113.5587629 | 119.9892774 | 6.4305144700 | 5.66 | 0s:1ms |
| NR15-1 | 15 | 38.91138667 | 40.83623531 | 1.9248486471 | 4.95 | 0s:1ms |
| NR15-2 | 15 | 38.83799551 | 40.46917515 | 1.6311796458 | 4.20 | 0s:0ms |
| NR16-1 | 16 | 143.3797811 | 153.1680967 | 9.7883156069 | 6.83 | 0s:0ms |
| NR16-2 | 16 | 127.6978254 | 138.852915 | 11.1550895980 | 8.74 | 0s:0ms |
| NR17-1 | 17 | 49.18730653 | 50.88962734 | 1.7023208042 | 3.46 | 0s:0ms |
| NR18-1 | 18 | 196.982624 | 207.315911 | 10.3332869685 | 5.25 | 0s:0ms |
| NR20-1 | 20 | 125.1177542 | 132.3922982 | 7.2745439975 | 5.81 | 0s:0ms |
| NR20-2 | 20 | 121.7887166 | 131.2593927 | 9.4706761087 | 7.78 | 0s:0ms |
| NR21-1 | 21 | 148.0967879 | 156.9650187 | 8.8682307593 | 5.99 | 0s:0ms |
| NR23-1 | 23 | 174.3425422 | 184.1686439 | 9.8261016920 | 5.64 | 0s:0ms |
| NR24-1 | 24 | 137.7590521 | 142.434406 | 4.6753539539 | 3.39 | 0s:1ms |
| NR25-1 | 25 | 188.7187899 | 199.1131651 | 10.3943751609 | 5.51 | 0s:1ms |
| NR26-1 | 26 | 244.5742803 | 264.8485458 | 20.2742655209 | 8.29 | 0s:1ms |
| NR26-2 | 26 | 300.2630794 | 331.9292039 | 31.6661245429 | 10.55 | 0s:1ms |
| NR27-1 | 27 | 220.659606 | 233.3093883 | 12.6497823783 | 5.73 | 0s:1ms |
| NR30-1 | 30 | 177.2586681 | 184.4651984 | 7.2065303184 | 4.07 | 0s:1ms |
| NR30-2 | 30 | 172.6501848 | 182.7352882 | 10.0851033839 | 5.84 | 0s:1ms |
| NR40-1 | 40 | 352.4026268 | 382.7128028 | 30.3101760009 | 8.60 | 0s:2ms |
| NR50-1 | 50 | 376.8063801 | 394.6690622 | 17.8626821409 | 4.74 | 0s:3ms |
| NR60-1 | 60 | 514.8363192 | 547.7523987 | 32.9160794969 | 6.39 | 0s:4ms |
| IN9-1 | 9 | 24.14213562 | 27.32050808 | 3.1783724520 | 13.17 | 0s:0ms |
| IN10-1 | 10 | 98.8354317 | 102.6159684 | 3.7805367434 | 3.83 | 0s:0ms |
| IN10-2 | 10 | 99.88507689 | 105.2785764 | 5.3934994850 | 5.40 | 0s:0ms |
| IN11-1 | 11 | 56.98440037 | 61.42216631 | 4.4377659420 | 7.79 | 0s:0ms |

Tabel B.7: Packomania Benchmark Instances

| Instantie | N | Beste radius | Radius | Vergroting | Vergroting (%) | Tijd |
|-----------|-----|--------------|-------------|---------------|----------------|---------|
| IN11-2 | 11 | 60.70996139 | 64.21790153 | 3.5079401370 | 5.78 | 0s:0ms |
| IN12-1 | 12 | 215.4700538 | 215.4700538 | 0.0000000000 | 0.00 | 0s:0ms |
| IN14-1 | 14 | 113.5587629 | 119.9892774 | 6.4305144700 | 5.66 | 0s:0ms |
| IN15-1 | 15 | 38.83799551 | 40.46917515 | 1.6311796458 | 4.20 | 0s:0ms |
| IN16-1 | 16 | 127.6978254 | 138.852915 | 11.1550895980 | 8.74 | 0s:0ms |
| IN17-1 | 17 | 49.18730653 | 50.88962734 | 1.7023208042 | 3.46 | 0s:0ms |
| IN17-2 | 17 | 241.4213562 | 273.2050808 | 31.7837245196 | 13.17 | 0s:0ms |
| IN25-1 | 25 | 21.54700538 | 22.53477869 | 0.9877733061 | 4.58 | 0s:0ms |
| IN28-1 | 28 | 21.54700538 | 21.54700538 | 0.0000000000 | 0.00 | 0s:1ms |
| IN162-1 | 162 | 11.40165346 | 11.92099073 | 0.5193372717 | 4.55 | 0s:27ms |

B.4 Grotere aantallen cirkels

Enkel de resultaten voor meer dan 1000 cirkels worden hier getoond. De volledige lijst van grotere problemen, van 100 cirkels tot 14000 cirkels, kan je terug vinden op GitHub [4].

Tabel B.8: Grotere aantallen gelijke cirkels

| circlearcount | Radius | Overlap (<i>eenheden</i> ²) | Tijd |
|---------------|-------------|--|-----------|
| 1000 | 34.19787581 | 1.94E-19 | 0s:252ms |
| 1100 | 35.8477868 | 2.41E-19 | 0s:295ms |
| 1200 | 37.28489633 | 2.94E-19 | 0s:336ms |
| 1300 | 38.89323424 | 3.45E-19 | 0s:391ms |
| 1400 | 40.26528552 | 3.81E-19 | 0s:425ms |
| 1500 | 41.68391561 | 4.29E-19 | 0s:472ms |
| 1600 | 42.93386435 | 4.76E-19 | 0s:530ms |
| 1700 | 44.30874729 | 6.18E-19 | 0s:581ms |
| 1800 | 45.52705198 | 7.04E-19 | 0s:644ms |
| 1900 | 46.78978667 | 6.85E-19 | 0s:698ms |
| 2000 | 47.98140222 | 7.66E-19 | 0s:747ms |
| 3000 | 58.49108183 | 1.76E-18 | 1s:447ms |
| 4000 | 67.45994578 | 3.07E-18 | 2s:325ms |
| 5000 | 75.2832421 | 4.80E-18 | 3s:237ms |
| 6000 | 82.39743073 | 7.21E-18 | 4s:300ms |
| 7000 | 88.91507648 | 1.03E-17 | 5s:544ms |
| 8000 | 95.01109884 | 1.43E-17 | 6s:842ms |
| 9000 | 100.6438734 | 1.89E-17 | 8s:315ms |
| 10000 | 106.0253597 | 2.32E-17 | 9s:848ms |
| 11000 | 111.1346661 | 3.14E-17 | 11s:309ms |
| 12000 | 116.0794088 | 3.76E-17 | 12s:915ms |
| 13000 | 120.7501442 | 4.43E-17 | 14s:766ms |
| 14000 | 125.3002024 | 5.07E-17 | 16s:606ms |

Tabel B.9: Grotere aantallen cirkels: $r_i = i^{1/2}$

| circlecount | Radius | Overlap (<i>eenheden</i> ²) | Tijd |
|-------------|-------------|--|-----------|
| 1000 | 774.0131872 | 2.72E-17 | 0s:339ms |
| 1100 | 849.6728853 | 3.23E-17 | 0s:385ms |
| 1200 | 926.1796164 | 4.35E-17 | 0s:448ms |
| 1300 | 1001.604293 | 5.46E-17 | 0s:509ms |
| 1400 | 1079.798785 | 6.69E-17 | 0s:562ms |
| 1500 | 1156.055812 | 8.82E-17 | 0s:631ms |
| 1600 | 1233.213828 | 9.67E-17 | 0s:705ms |
| 1700 | 1309.750398 | 1.20E-16 | 0s:768ms |
| 1800 | 1386.13319 | 1.34E-16 | 0s:824ms |
| 1900 | 1462.807238 | 1.54E-16 | 0s:888ms |
| 2000 | 1539.951867 | 1.79E-16 | 1s:15ms |
| 3000 | 2303.207787 | 4.22E-16 | 1s:810ms |
| 4000 | 3066.965929 | 1.11E-15 | 2s:816ms |
| 5000 | 3830.949569 | 1.90E-15 | 3s:968ms |
| 6000 | 4595.507795 | 336.1716726 | 5s:210ms |
| 7000 | 5358.723848 | 585.3556305 | 6s:511ms |
| 8000 | 6116.413934 | 7.83E-15 | 7s:726ms |
| 9000 | 6874.532534 | 0.1235373296 | 9s:550ms |
| 10000 | 7646.122416 | 1.454465798 | 11s:589ms |
| 11000 | 8398.61771 | 1.64E-14 | 13s:231ms |
| 12000 | 9164.900313 | 2267.23912 | 14s:473ms |
| 13000 | 9928.581657 | 2.543904356 | 16s:154ms |
| 14000 | 10694.5632 | 3.48E-14 | 18s:90ms |

Tabel B.10: Grotere aantallen cirkels: $r_i = i^{-1/2}$

| circlecount | Radius | Overlap (<i>eenheden</i> ²) | Tijd |
|-------------|-------------|--|----------|
| 1000 | 2.893283783 | 4.51E-22 | 0s:538ms |
| 1100 | 2.908350125 | 5.10E-22 | 0s:617ms |
| 1200 | 2.923408029 | 5.19E-22 | 0s:716ms |
| 1300 | 2.936039285 | 6.21E-22 | 0s:833ms |
| 1400 | 2.943533134 | 6.57E-22 | 0s:946ms |
| 1500 | 2.953979594 | 6.30E-22 | 1s:78ms |
| 1600 | 2.966069401 | 6.77E-22 | 1s:154ms |
| 1700 | 2.976648708 | 7.33E-22 | 1s:304ms |
| 1800 | 2.985482798 | 7.81E-22 | 1s:429ms |
| 1900 | 2.995659982 | 8.14E-22 | 1s:572ms |
| 2000 | 3.000845426 | 8.10E-22 | 1s:698ms |
| 3000 | 3.065430097 | 1.07E-21 | 3s:288ms |
| 4000 | 3.111816155 | 1.39E-21 | 5s:248ms |
| 5000 | 3.145865417 | 1.62E-21 | 7s:509ms |
| 6000 | 3.173380049 | 1.90E-21 | 10s:94ms |

Tabel B.10: Grotere aantallen cirkels: $r_i = i^{-1/2}$

| circlecount | Radius | Overlap (eenheden ²) | Tijd |
|-------------|-------------|----------------------------------|-----------|
| 7000 | 3.19748614 | 2.11E-21 | 12s:814ms |
| 8000 | 3.217527155 | 2.37E-21 | 15s:977ms |
| 9000 | 3.234875939 | 2.63E-21 | 19s:132ms |
| 10000 | 3.251215521 | 6.92E-06 | 22s:468ms |
| 11000 | 3.265715891 | 3.19E-21 | 26s:522ms |
| 12000 | 3.278766989 | 5.51E-17 | 30s:235ms |
| 13000 | 3.291515829 | 3.49E-21 | 34s:392ms |
| 14000 | 3.301772515 | 3.86E-21 | 39s:261ms |

Tabel B.11: Grotere aantallen cirkels: $r_i = i^{-2/3}$

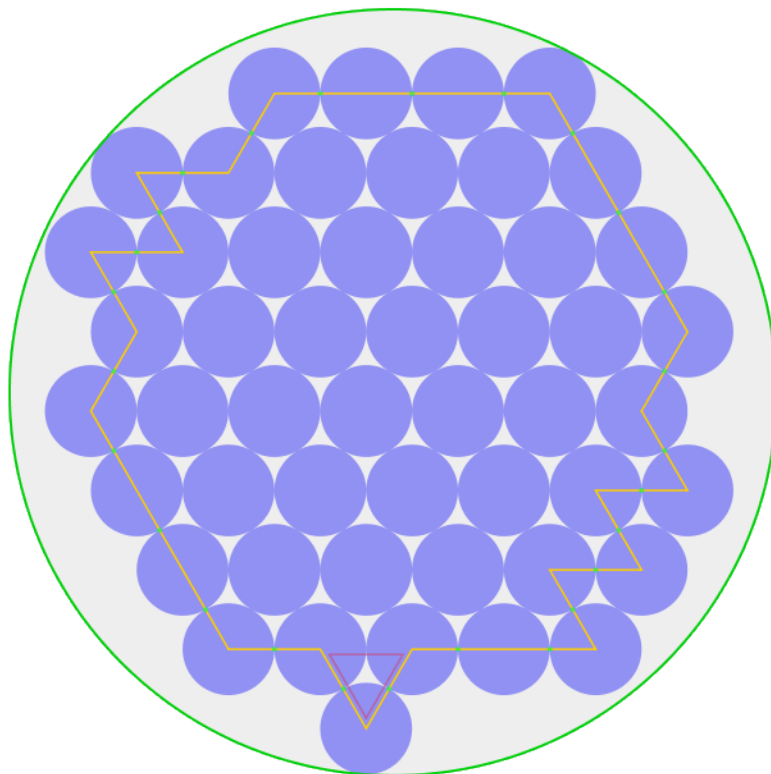
| circlecount | Radius | Overlap (eenheden ²) | Tijd |
|-------------|-------------|----------------------------------|-----------|
| 1000 | 1.856466117 | 5.37E-22 | 0s:539ms |
| 1100 | 1.858198724 | 4.73E-22 | 0s:635ms |
| 1200 | 1.859175544 | 4.16E-22 | 0s:748ms |
| 1300 | 1.859678713 | 1.11E-21 | 0s:848ms |
| 1400 | 1.860597082 | 8.41E-22 | 0s:942ms |
| 1500 | 1.86111262 | 9.23E-22 | 1s:88ms |
| 1600 | 1.861617779 | 7.82E-22 | 1s:229ms |
| 1700 | 1.862434075 | 1.02E-21 | 1s:364ms |
| 1800 | 1.862989167 | 9.70E-22 | 1s:544ms |
| 1900 | 1.863786306 | 1.29E-21 | 1s:668ms |
| 2000 | 1.864143827 | 8.76E-22 | 1s:833ms |
| 3000 | 1.868063495 | 1.29E-21 | 3s:677ms |
| 4000 | 1.870522828 | 2.51E-21 | 6s:232ms |
| 5000 | 1.872601012 | 3.12E-21 | 9s:237ms |
| 6000 | 1.873700703 | 3.46E-21 | 12s:617ms |
| 7000 | 1.874811381 | 4.58E-21 | 16s:713ms |
| 8000 | 1.875973551 | 4.03E-21 | 21s:538ms |
| 9000 | 1.87660983 | 6.46E-21 | 25s:726ms |
| 10000 | 1.87406634 | 0.03735698644 | 31s:536ms |
| 11000 | 1.875967596 | 0.03611959323 | 37s:846ms |
| 12000 | 1.876456183 | 0.03611959323 | 44s:796ms |
| 13000 | 1.87685631 | 0.03611959323 | 50s:342ms |
| 14000 | 1.877341355 | 0.03611959323 | 57s:698ms |

Tabel B.12: Grotere aantallen cirkels: $r_i = i^{-1/5}$

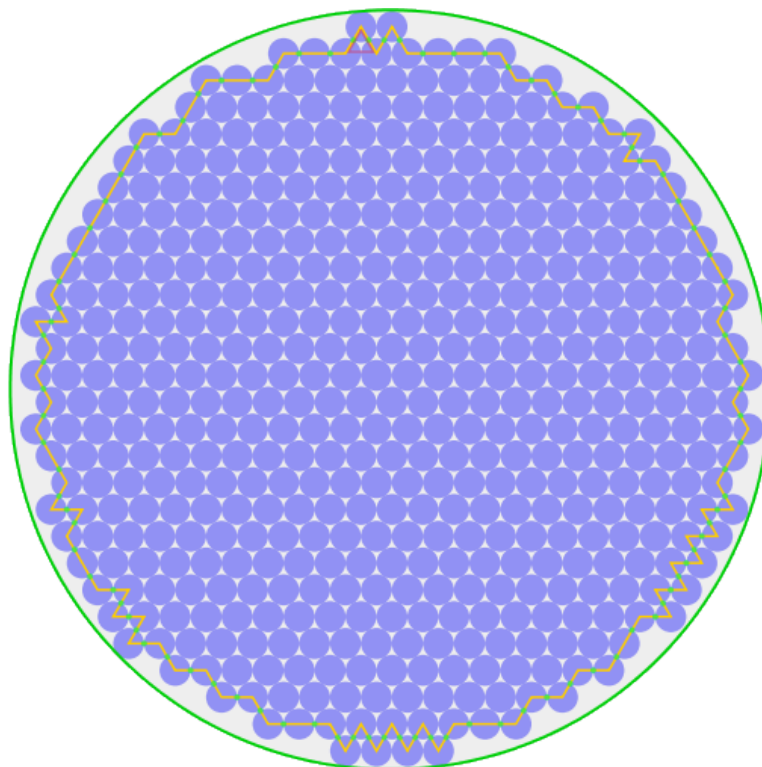
| circlecount | Radius | Overlap (eenheden ²) | Tijd |
|-------------|-------------|----------------------------------|-----------|
| 1000 | 11.42927809 | 5.26E-21 | 0s:324ms |
| 1100 | 11.73702799 | 7.34E-21 | 0s:383ms |
| 1200 | 12.04875608 | 7.00E-21 | 0s:438ms |
| 1300 | 12.34201715 | 8.30E-21 | 0s:496ms |
| 1400 | 12.59566073 | 8.52E-21 | 0s:561ms |
| 1500 | 12.86483556 | 9.24E-21 | 0s:614ms |
| 1600 | 13.10781613 | 1.04E-20 | 0s:701ms |
| 1700 | 13.37232588 | 1.09E-20 | 0s:759ms |
| 1800 | 13.56759994 | 1.23E-20 | 0s:809ms |
| 1900 | 13.79318079 | 1.31E-20 | 0s:862ms |
| 2000 | 14.00920357 | 1.33E-20 | 0s:926ms |
| 3000 | 15.7834951 | 2.15E-20 | 1s:788ms |
| 4000 | 17.19442986 | 3.29E-20 | 2s:758ms |
| 5000 | 18.38386809 | 4.42E-20 | 3s:887ms |
| 6000 | 19.39991246 | 5.92E-20 | 5s:150ms |
| 7000 | 20.29329017 | 7.32E-20 | 6s:536ms |
| 8000 | 21.11324047 | 8.98E-20 | 7s:953ms |
| 9000 | 21.8771281 | 1.09E-19 | 9s:610ms |
| 10000 | 22.55910849 | 1.22E-19 | 11s:336ms |
| 11000 | 23.23957158 | 1.37E-19 | 13s:33ms |
| 12000 | 23.8376261 | 1.55E-19 | 14s:991ms |
| 13000 | 24.40974434 | 1.73E-19 | 16s:856ms |
| 14000 | 24.95175949 | 1.90E-19 | 18s:923ms |

Bijlage C

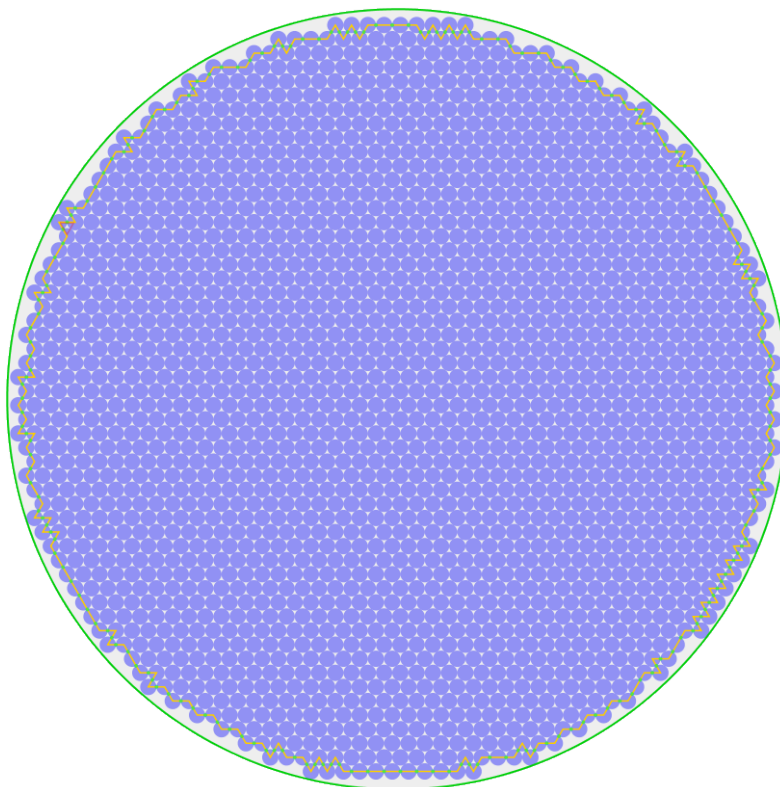
Extra *packing* figuren



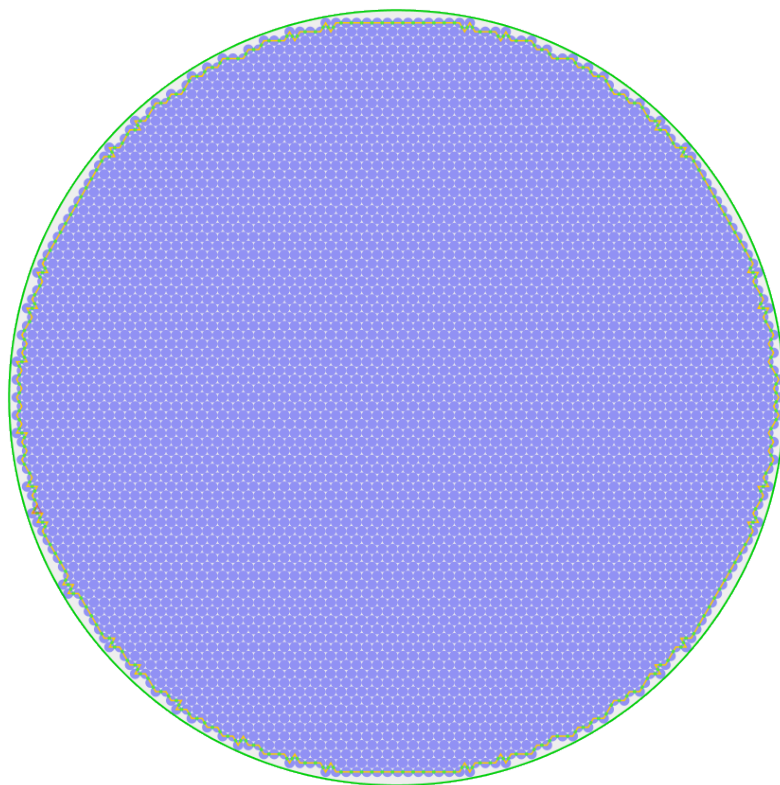
Figuur C.1: *Packing* voor 50 cirkels met gelijke grootte



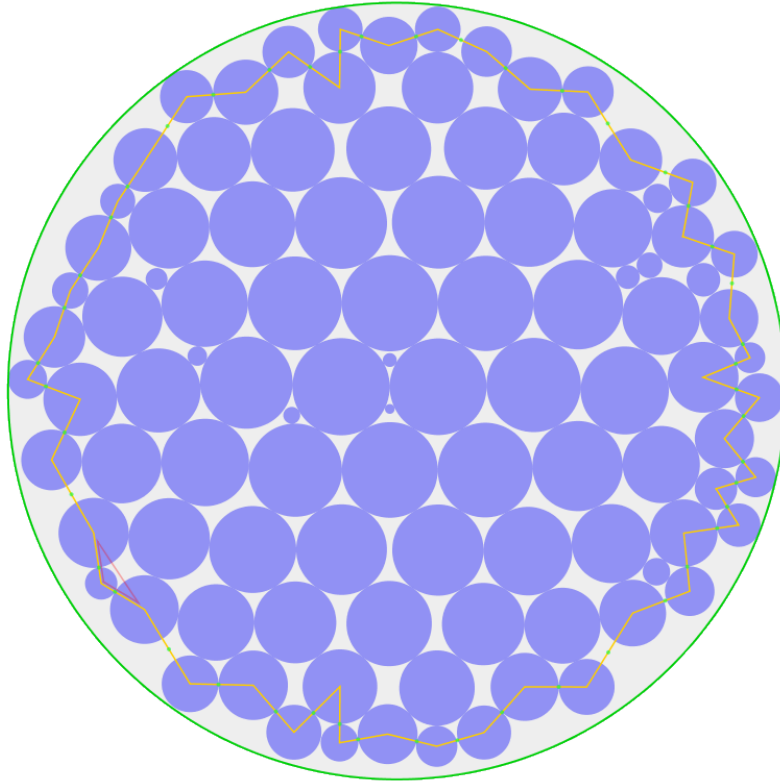
Figuur C.2: *Packing* voor 500 cirkels met gelijke grootte



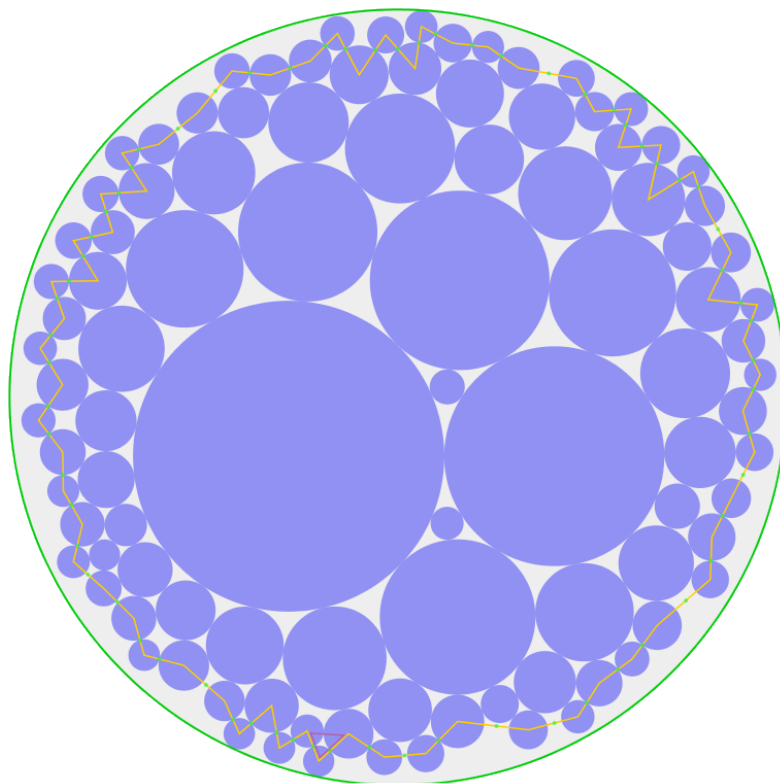
Figuur C.3: *Packing* voor 2000 cirkels met gelijke grootte



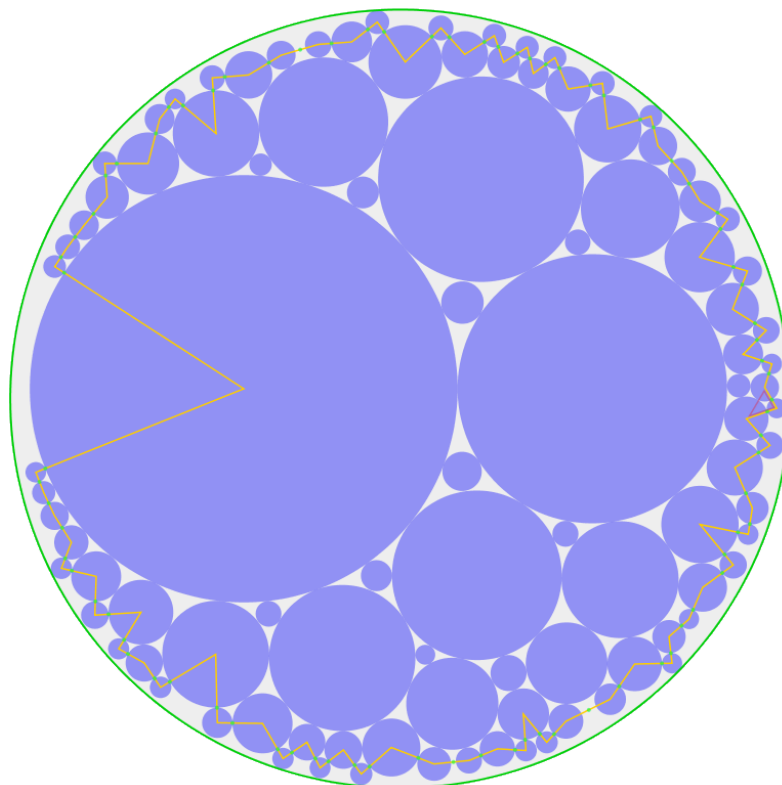
Figuur C.4: *Packing* voor 5000 cirkels met gelijke grootte



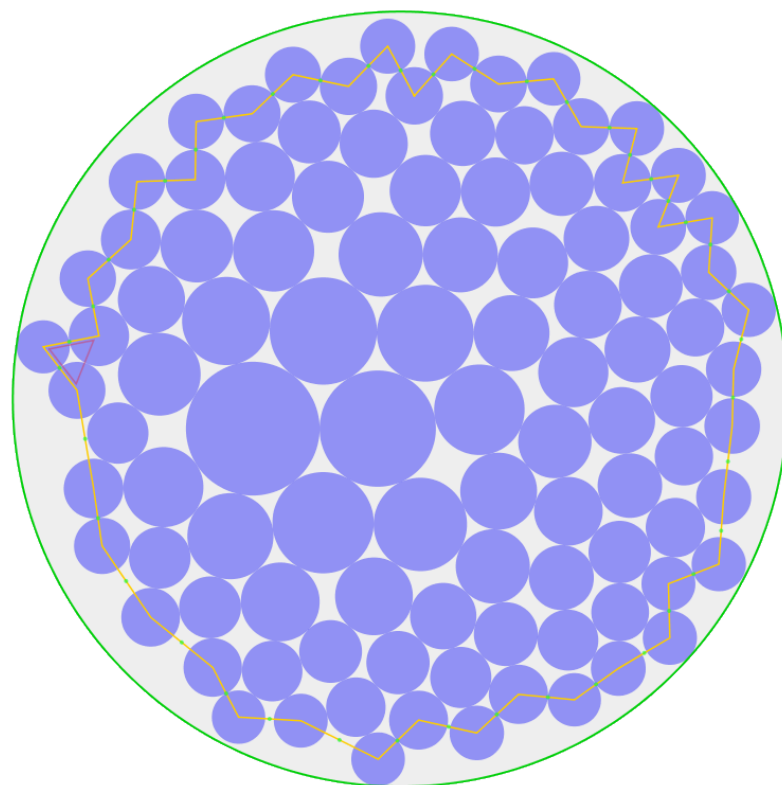
Figuur C.5: *Packing* voor 100 cirkels met verdeling $r_i = i^{1/2}$



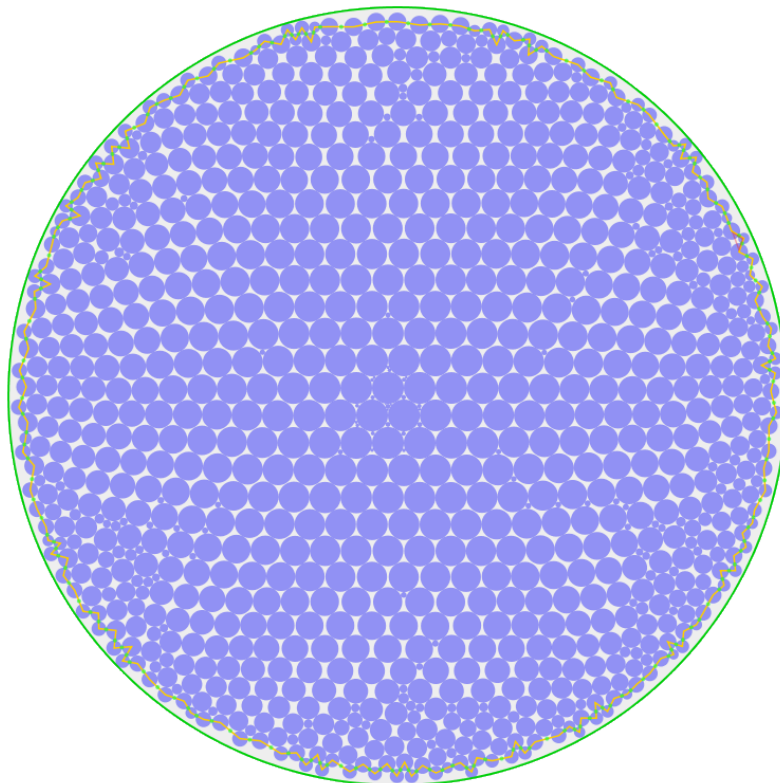
Figuur C.6: *Packing* voor 100 cirkels met verdeling $r_i = i^{-1/2}$



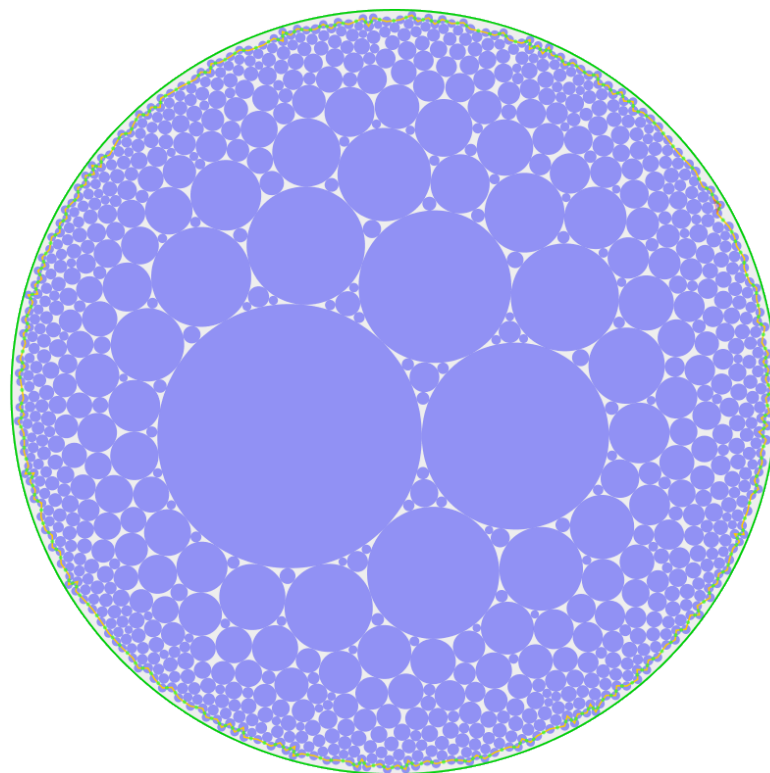
Figuur C.7: *Packing* voor 100 cirkels met verdeling $r_i = i^{-2/3}$



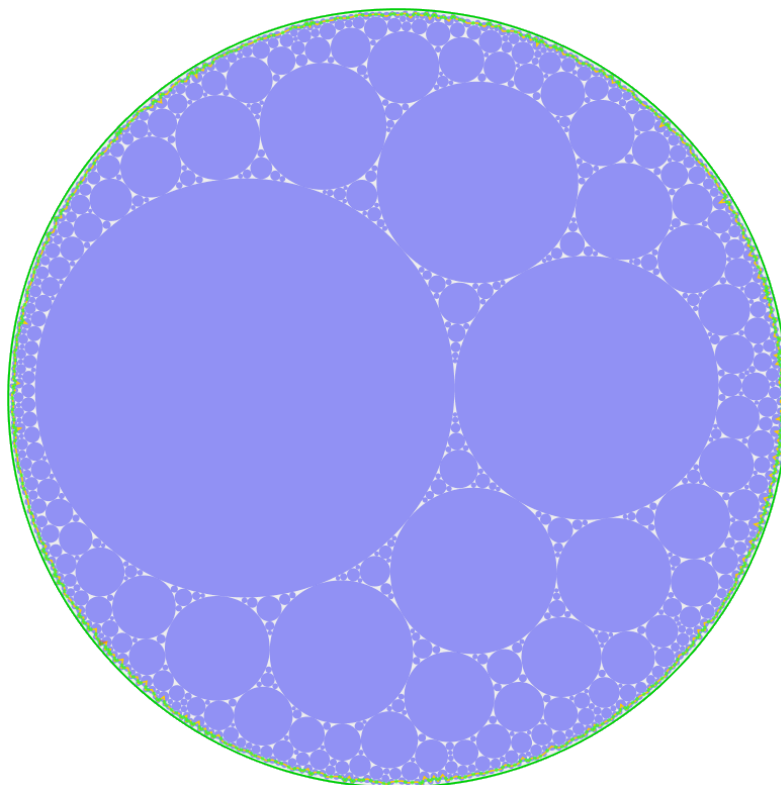
Figuur C.8: *Packing* voor 100 cirkels met verdeling $r_i = i^{-1/5}$



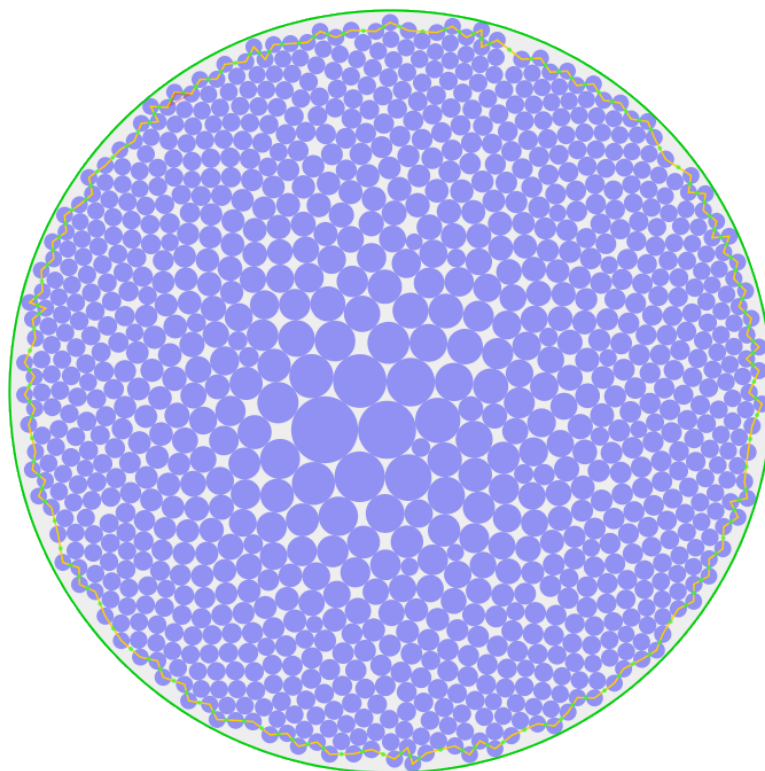
Figuur C.9: *Packing* voor 1000 cirkels met verdeling $r_i = i^{1/2}$



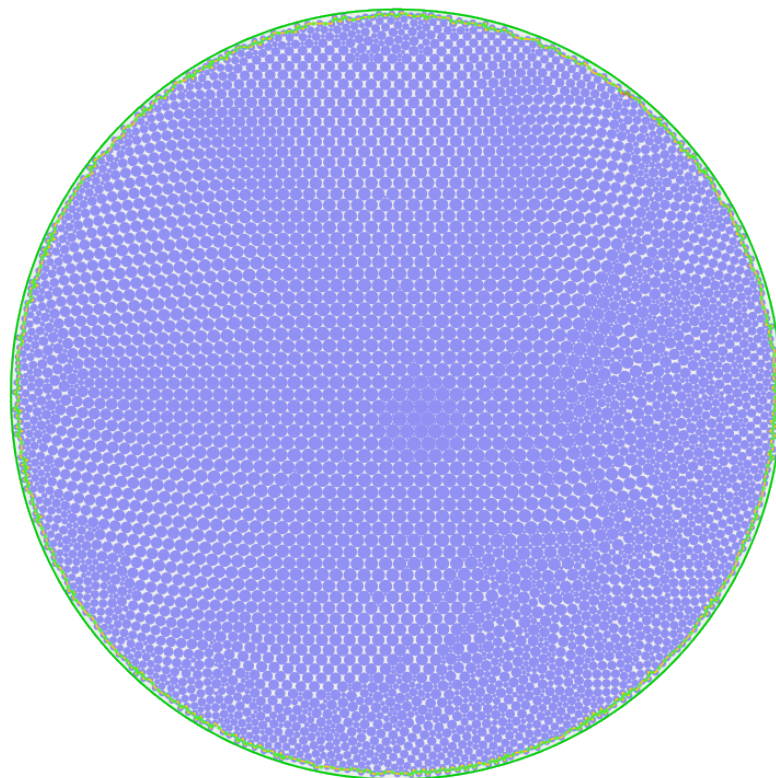
Figuur C.10: *Packing* voor 1000 cirkels met verdeling $r_i = i^{-1/2}$



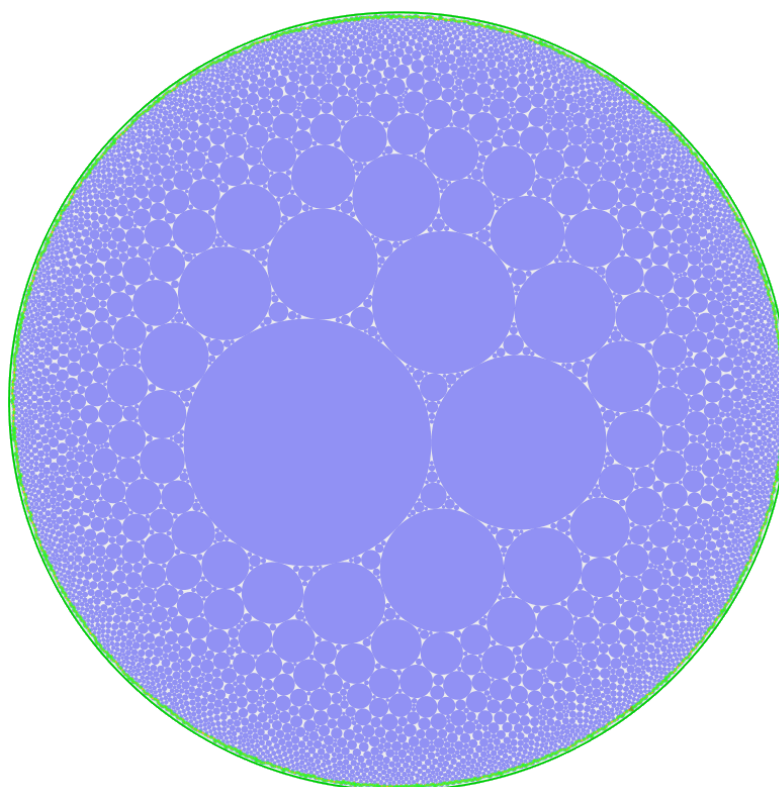
Figuur C.11: *Packing* voor 1000 cirkels met verdeling $r_i = i^{-2/3}$



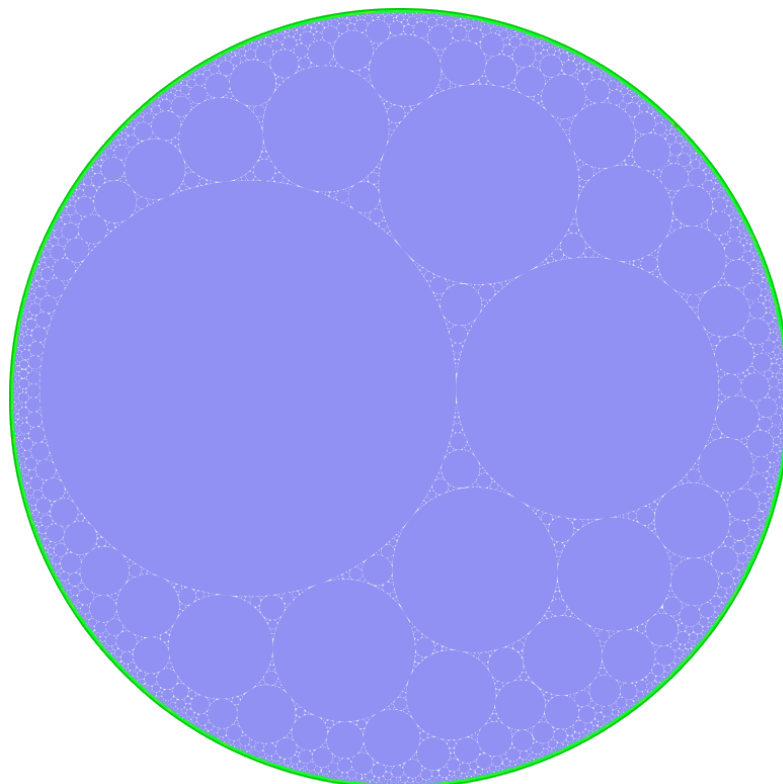
Figuur C.12: *Packing* voor 1000 cirkels met verdeling $r_i = i^{-1/5}$



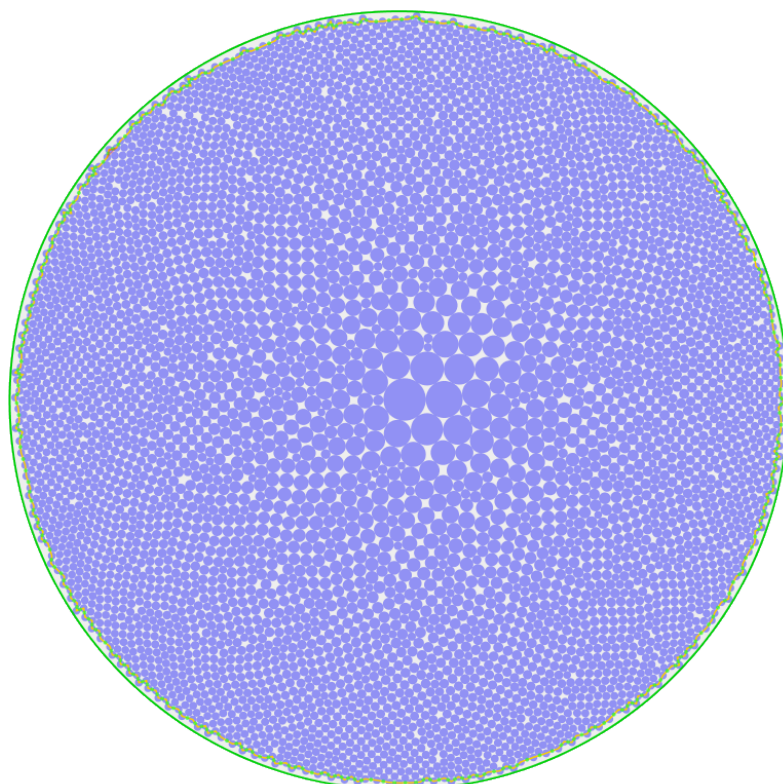
Figuur C.13: *Packing* voor 5000 cirkels met verdeling $r_i = i^{1/2}$



Figuur C.14: *Packing* voor 5000 cirkels met verdeling $r_i = i^{-1/2}$



Figuur C.15: *Packing* voor 5000 cirkels met verdeling $r_i = i^{-2/3}$



Figuur C.16: *Packing* voor 5000 cirkels met verdeling $r_i = i^{-1/5}$

COMPUTERWETENSCHAPPEN
Celestijnenlaan 200A - bus 2402
3001 HEVERLEE, BELGIË
tel. + 32 16 32 77 00
fax + 32 16 32 79 96
www.kuleuven.be

