Glasmeesters Case

Milou Bisseling, Belle Bruinsma en Joeri Fresen

Abstract

In dit verslag wordt het two-dimensional bin packing problem behandeld in de glasindustrie. Aan de hand van verschillende bestellingen worden algoritmes met elkaar vergeleken. Het Guillotine Algoritme blijkt het best presterende algoritme en is ook nog eens uitvoerbaar in de praktijk.

Keywords: Two-dimensional bin packing, Optimization, Heuristic algorithm, Glass cutting

1. Inleiding

In het *two-dimensional bin packing problem* (2BP) wordt gezocht naar een de optimale manier een bepaald aantal rechthoeken uit een minimum aantal identieke grote rechthoeken te halen zonder overlapping (Lodi, 1999). Het 2BP is in verschillende industrien toepasbaar waaronder bij het snijden van orders in de glas- en staalindustrie. Eerdere theoretische bijdragen aan het 2BP hebben bewezen dat het probleem *NP-hard* is. Problemen die als *NP-hard* geclassificeerd worden kunnen (deels) benaderd worden door een schatting of een stapsgewijze oplossing, in plaats van door het vinden van een efficint exact algoritme (Chou 2016).

In deze situatie wordt het 2BP toegepast in de glasindustrie waarbij gezocht wordt naar snijpatronen. Het aantal glasplaten en de resten moeten hierbij worden geminimaliseerd daarnaast is het mogelijk om de rechthoeken te roteren. Na eigen onderzoek binnen de glasindustrie is gebleken dat glasplaten alleen in rechte lijnen gesneden kunnen worden, maar in deze case is deze beperking geen vereiste. Daarom valt ons probleem onder het probleem waarin de rechthoeken gedraaid mogen worden en er zonder beperking gesneden mag worden. De laatste vereiste die toegepast wordt in dit verslag is dat de rechthoeken orthogonaal geplaatst worden, hetgeen inhoudt dat ze geplaatst worden parallel aan de grote rechthoek. Lodi categoriseert dit probleem als *2BP Rotate Free* (2BP—R—F).

Vijf verschillende algoritmen zijn toegepast op deze case, waarvan sommigen ook nog *guillotineable* zijn. Dit houdt in dat de snedes in de glasplaten alleen recht door de hele glasplaat gemaakt kunnen worden, wat tot op heden volgens Puchinger et al. de praktijk is in de glasindustrie. De eerste drie algoritmes die vergeleken worden zijn gebaseerd op het artikel van Jylnki (2010): de Guillotine, de MaxRects en de Skyline. Het vierde algoritme wat vergeleken wordt, de Shelf, wordt hierna gentroduceerd, waarbij de glasplaat ingedeeld wordt in lagen. Als laatste wordt het de DFBF beschreven, hetgeen een combinatie is van een depte-first, breadth-first en een constructief algoritme. Van de vergeleken algoritmes zijn de Guillotine en de Shelf *guillotineable*.

Aan de hand een aantal bestellingen van ruiten van verschillende groottes worden verschillende algoritmes vergeleken, hiervan wordt een uitgebreide beschrijving gegeven in Hoofdstuk 2. Deze algoritmen hebben verschillende scores opgeleverd waarvan de Guillotine de beste oplossing is van de algoritmes die wij vergeleken hebben, deze scores worden beschreven in Hoofdstuk 3. Waarna in Hoofdstuk 4 samengevat zal worden wat de uitkomsten zijn maar ook wat de beperkingen zijn van de toegepaste methodes.

2. Materialen en Methode

2.1. Materialen

Alle algoritmen in deze case zijn geschreven in Python 3. Daarnaast is er gebruik gemaakt van de in python ingebouwde library: rectpack. In rectpack zit een verzameling van heuristische algoritmen voor het oplossen van het 2BP. Rectpack implementeert drie algoritmen: Skyline, Maxrects en Guillotine. Daarnaast geeft het de mogelijkheid tot sorteren van de orderlijst op verschillende gronden: oppvervlakte, omtrek, langste zijde, ratio en random. De andere libraries die gebruikt zijn in de algoritmes die gentroduceerd worden zijn numpy, itertools en matplotlib. Verder is gebruik gemaakt van de Macbook Pro 2016 met 2 GHz Intel Core i5 processor en 8 GB 1867 MHz LPDDR3 geheugen.

2.2. Methoden

2.2.1. Skyline Algoritme

Het Skyline algoritme houdt een lijst bij over de breedte van de glasplaat, waarbij de grens loopt over de bovenkanten van de geplaatste rechthoeken.

Deze lijst groeit lineair met het aantal geplaatste rechthoeken. Daarbij wordt de *Bottom-left heuristic* gemplementeerd wat inhoudt dat de rechthoeken geplaatst worden van linksonder naar rechtsboven. Wanneer de onderste skyline vol is, e.g. er past geen rechthoek meer naast, wordt er een nieuwe skyline boven gecreerd. Het nadeel van de Skyline is dat het algoritme niet goed bijhoud welke gebieden in de glasplaat nog vrij zijn waardoor je grote gaten kunt krijgen zie Figuur 4 en 5

2.2.2. Guillotine Algoritme

Bij de Guillotine wordt de eerste rechthoek in een vrije hoek van de glasplaat wordt gezet waarna de resterende L-vormige ruimte wordt verdeeld in twee losse rechthoeken. Het is een populair algoritme aangezien het de overige ruimte bijhoudt, anders dan bij de Skyline waar deze niet bijgehouden wordt. In het algoritme wordt een lijst van rechthoeken bijgehouden die de vrije ruimte van de glasplaat vertegenwoordigen. Bij elke plaatsing wordt een lege rechthoek uitgekozen om de volgende order in te verpakken. De rechthoek wordt links onderin de hoek geplaatst, waarna door het gebruik van de Guillotine splitsing twee kleinere lege rechthoeken worden gecreerd zie Figuur 1. De volgende rechthoek wordt zo geplaatst dat de overgebleven ruimte in de vrije rechthoek geminimaliseerd wordt, dit wordt de *Guillotine Best Short Side Fit* genoemd. Dit proces gaat door tot dat er geen rechthoek van de orderlijst meer past in de lege rechthoeken die over zijn, in dit geval wordt er een nieuwe glasplaat aangemaakt. Een nadeel van dit algoritme is dat het alleen rechthoeken plaatst binnen de grenzen van de gesplitste rechthoeken, waardoor er geen rechthoek geplaatst kan worden op een splitsing.

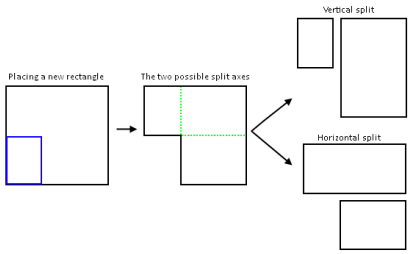

Figure 1: Guillotine split placement process (Jylnki, 2010)

2.2.3. Maxrects Algoritme

Zoals eerder genoemd kan bij de Guillotine geen rechthoek op de splitsing geplaatst worden. Een oplossing voor dit probleem is het Maximal Rectangle algorithm (MaxRects). Net als bij de Guillotine slaat MaxRects een lijst op van vrije rechthoeken die het vrije gebied van de glasplaat vertegenwoordigen. In tegenstelling tot de Guillotine dat een van de twee splitsingen kiest, neemt MaxRects beide splitsingen op hetzelfde moment op (zie figuur 2). Uit onderzoek van Jylnki (2010) is gebleken dat MaxRects beter presteert dan Skyline en de Guillotine.

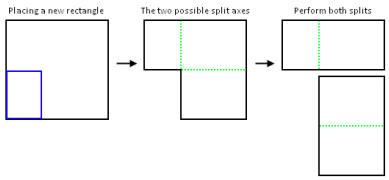

Figure 2: Maximal Rectangle split placement process (Jylanki, 2010)

2.2.4. Shelf Algoritme

Bij het Shelf algoritme wordt de glasplaat opgedeeld in verschillende planken. De orders worden gesneden van links naar rechts en van onder naar boven. Zodra een order niet meer in een plank past wordt er een nieuwe plank aangemaakt met daarin de order die niet meer in de plank daarvoor paste. Vervolgens wordt de orderlijst afgewerkt om te kijken of er in de overgebleven ruimte kleinere orders passen. Zodra er niks meer bij past In deze case worden de orders op vijf verschillende manieren gesorteerd: op lengte, breedte, omtrek en ratio.

2.2.5. DFBFC Algoritme

Dit algoritme bestaat uit twee fases. In de eerste fase wordt op een Depth First manier gezocht naar een combinatie van te snijden glasplaten die, gebaseerd op oppervlakte, goed in een grote plaat zouden passen. Dit gebeurt door verder te gaan in de boom totdat er geen item meer toegevoegd kan worden zonder de threshold te overschrijden.

Dit is hoe deze fase zou werken in het geval van een orderlijst met oppervlakten 4, 3, 2 en 1 met een threshold van 9. De eerste combinatie is dan dus 4, 3 en 2, want als 1 wordt toegevoegd, is de threshold van 9 overschreden.

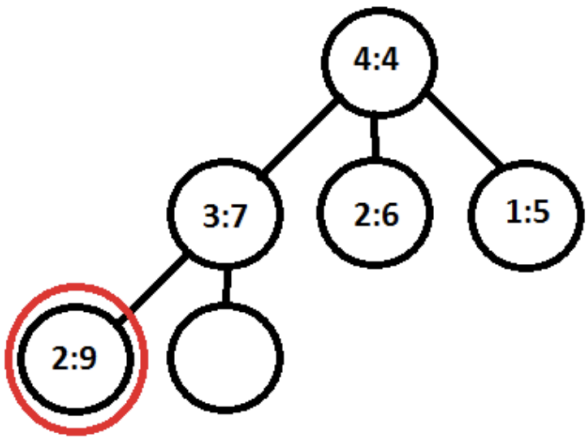

Figure 3: Depth-first Breadth-First Algoritme

In de tweede fase wordt gekeken of er een mogelijkheid is om de gevonden combinatie op een manier en 1 glasplaat de passen. Lukt dit niet, wordt in de Depth First tree gebacktrackt en naar een nieuwe combinatie gezocht. Dit zoeken van een manier om in 1 plaat te passen gaat op een Most Left-Most Bottom manier. Dit betekent dat er alleen iets geplaatst kan worden op een plek waar de linker onderhoek aan de linker- en onderzijde grenzen aan de buitenkant van de glasplaat of een ander geplaatst object.

Het toevoegen gaat altijd in een volgorde van groot naar klein oppervlakte. Dit zou dus 1 mogelijkheid per combinatie geven. Echter, omdat platen zowel liggend als staand kunnen worden toegevoegd, worden alle mogelijke lig-sta combinaties afgezocht door middel van een Breadth-First search. Voor elke gevonden oppervlakte-combinatie worden 2n (waarin n het aantal toe te voegen items is) opties geprobeerd.

3. Resultaten

In dit hoofdstuk worden de resultaten van de algoritmes per bestelling besproken en vergeleken.

3.1. Opdracht A

Allereerst was het de opdracht om een order van een Rotterdamse woningcorporatie te verwerken. Ze willen in een keer alle kapotte ruiten van

verschillende huizen, galerijwoningen en appartementen vervangen. De order bestond uit 22 rechthoeken van maximaal 200 cm breed cm en 340 cm lang. De glasplaten ze uit gesneden worden zijn 500 bij 600 cm. In figuur 4 is te zien dat elk algoritme drie glasplaten nodig heeft voor de order van de Rotterdamse woningcorporatie. Tevens is te zien dat het Guillotine algoritme de meeste aaneengesloten waste overhoudt. Ook de Shelf en DFBF komen op een score van 3 glasplaten maar scoren niet beter op het gebied van het minimaliseren van de waste.

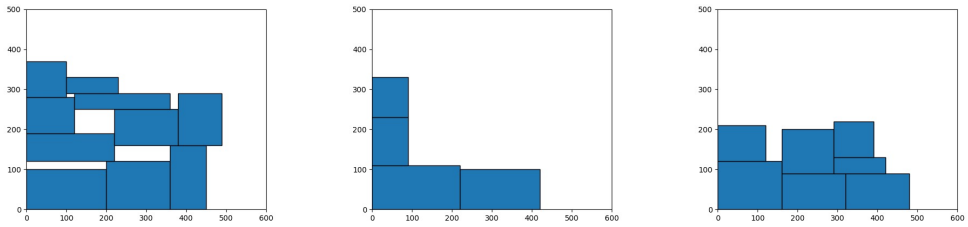

Figure 4: V.l.n.r. Skyline, Guillotine en MaxRects (in cm)

3.2. Opdracht B

In navolging van Rotterdam hebben drie Brabantse corporaties ook orders geplaatst. Deze orders zijn aan elkaar geplakt en in een keer verwerkt. In de figuur 5 is te zien dat elk algoritme zes glasplaten nodig heeft om de orders te verwerken. Wederom heeft het Guillotine algoritme de meeste aaneengesloten waste over.Ook de Shelf en DFBF komen op een score van 6 glasplaten maar scoren niet beter op het gebied van het minimaliseren van de waste. De Guillotine blijft het best presteren van de vergeleken algoritmes.

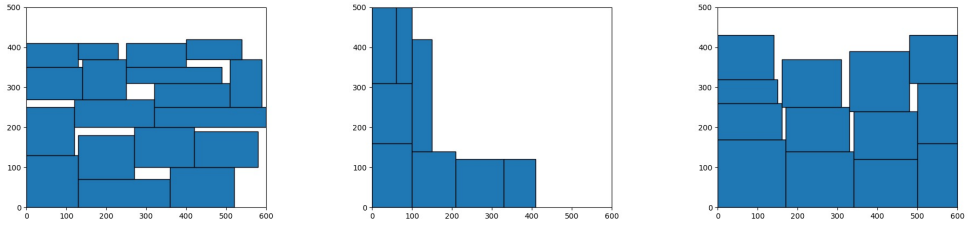

Figure 5: V.l.n.r. Skyline, Guillotine en MaxRects (in cm)

3.3. Opdracht C

In opdracht C bestaat de order uit drie types glas met verschillende standaard afmetingen. Een glasplaat van type I betekent dat de betreffende ruit tenminste uit type I moet bestaan, maar type II mag ook. Voor indicaties van type II mag je ook type III gebruiken, maar geen type I. Voor type III mag je alleen type III gebruiken. Ook hier blijft de Guillotine het best presteren van de vergeleken algoritmes.

3.4. Opdracht D

In deze opdracht wordt een bestelling gedaan van een bepaald aantal driehoeken, dit betreft in plaats van een 2BP een triangle packing problem (TBP). Chou heeft bewijst dat ook het TBP NP-hard is, ze concludeert daaruit dat het TBP benaderd moet worden met een schatting of heuristische algoritmes in plaats van exacte algoritmes. Ons idee was om driehoeken te combineren tot rechthoeken en ze op deze manier uit te snijden. Bij het tekenen van een aantal drie hoeken zien we dat het niet gaat om rechthoekige driehoeken en valt deze optie dan ook af. Een tweede optie bedacht is is om de rechthoeken te sorteren op de langste zijden, er dan recht hoeken van maken en deze in onze vorige algoritmes vergelijken. Een nadeel van deze methode is dat er er veel onbruikbare resten over zullen blijven.

4. Discussie

Door vergelijking van bovengenoemde algoritmes kan geconcludeerd worden dat het DFBF algoritme het best presterende algoritme is van de algoritmes die wij vergeleken hebben. Voor de glas industrie zou dit dan ook de het beste algoritme zijn wanneer zij de mogelijkheid hebben om te snijden in hoeken in plaats van alleen in rechte lijnen. Vooralsnog is het volgens Puchinger et al. niet mogelijk om in andere vormen dan rechte lijnen glas te snijden. Ook na klein onderzoek bij bedrijven in Nederland kan geconcludeerd worden dan de bedrijven alleen de mogelijkheid hebben om de glasplaat volledig door te snijden. Wanneer het praktisch nut meegenomen wordt in de vergelijking zou de Guillotine het enige algoritme zijn die bruikbaar is in de glasindustrie.

Wel is het mogelijk het algoritme te verbeteren, dit zou in de glasindustrie nog steeds niet te gebruiken zijn, maar het 2D bin parking probleem is ook in vele andere industriegebieden bruikbaar. In de door ons gebruikte versie moet men nog handmatig zoeken naar de meest optimale combinatie

van parameters (threshold en minimum waarde percentage). Dit zou potentieel kunnen worden geautomatiseerd, dit is niet alleen prettiger, maar geeft ook garanties voor de beste combinatie. Ook de volgorde waarin de Depth-First wordt afgezocht heeft invloed op de combinaties die gevonden kunnen worden. Tot dusver is alleen gebruik gemaakt van de gegeven volgorde in de orderlijsten en het sorteren op hoogte. Voor mogelijk betere resultaten kunnen eerst andere manieren van sorteren geprobeerd worden. Ook is het nog een eventuele mogelijkheid om dit op een willekeurige manier te doen. Dit geeft nog een grote hoeveelheid nieuwe combinaties en dus ook potentieel betere resultaten. Echter, het willekeurig proberen heeft nog weinig met heuristiek te maken en is praktisch een Brute Force manier van oplossen. Als toekomstig onderzoek kan geprobeerd worden de meest optimale manier van sorteren te vinden voor het Depth-First afzoeken.

5. Referenties

Chou, A. (2016). NP-hard triangle packing problems. Research Science Institute summer program for highschool students.

Jylnki, J. (2010). A thousand ways to pack the bin-a practical approach to two-dimensional rectangle bin packing.

Leung, J. Y., Tam, T. W., Wong, C. S., Young, G. H., and Chin, F. Y. (1990). Packing squares into a square. Journal of Parallel and Distributed Computing, 10(3), 271-275.

Lodi, A., Martello, S., and Vigo, D. (1999). Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. INFORMS Journal on Computing, 11(4), 345-357.

Puchinger, J., Raidl, G. R., and Koller, G. (2004, April). Solving a real-world glass cutting problem. In European Conference on Evolutionary Computation in Combinatorial Optimization (pp. 165-176). Springer Berlin Heidelberg.