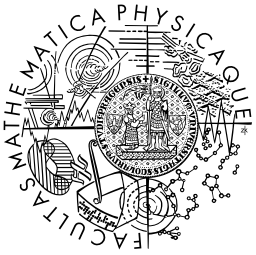


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Damian Wałoszek

Reálná aplikace uspořádávání mnohoúhelníků

Informatický ústav Univerzity Karlovy

Vedoucí bakalářské práce: RNDr. Petr Čermák, Ph.D.
Studijní program: Informatika
Studijní obor: Obecná informatika

Praha 2021

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Rád bych vyjádřil své upřímné poděkování především svému vedoucímu práce Petru Čermákovi za výborné vedení a rodině a Aničce za podporu v tomto náročném období.

Název práce: Reálná aplikace uspořádávání mnohoúhelníků

Autor: Damian Wałoszek

Katedra: Informatický ústav Univerzity Karlovy

Vedoucí bakalářské práce: RNDr. Petr Čermák, Ph.D., Katedra fyziky kondenzovaných látek

Abstrakt: V 2D irregular bin packing problem je za cíl umístit co nejvíce 2D předmětů do specifikované nádoby. Přicházíme s unikátním praktickým využitím, pro který je nutné použít online algoritmus. Vyvineme a implementujeme state-of-the-art algoritmus založený na vlastní tzv. picking policy, který lze využít i v offline algoritmech.

Klíčová slova: bin packing problem optimalizace online algoritmus MaMBA projekt 2DIBPP

Title: Real application of online 2D irregullar bin packing

Author: Damian Wałoszek

Department: Computer Science Institute of Charles University

Supervisor: RNDr. Petr Čermák, Ph.D., Department of Condensed Matter Physics

Abstract: In the 2D irregular bin packing problem, the goal is to place as many 2D items as possible to the specified bins. Our application has unique restriction of online algorithm. We develop and implement a state-of-the-art online algorithm based on our own picking policy that can also be used in offline algorithms.

Keywords: bin packing problem optimization online algorithm MaMBA projekt 2DIBPP

Obsah

Seznam použitých zkratek

MaMBA	Magnetoelastic Materials beyond Born-Oppenheimer Approximation
ALSA	Automatic Laue Sample Aligner
BPP	Bin packing problem
NFP	No-fit polygon
IFP	Inner-fit polygon
CFR	Collision-free region
CR	Competitive ratio

1. Úvod

V rámci fyzikálního projektu MaMBA jsme narazili na jeden praktický problém: v jedné fázi potřebujeme pomocí robotické ruky umístit velké množství (stovky) malých, plochých krystalků na kulatou destičku tak, aby se jich tam vešlo co nejvíc.

Na tento problém budeme nahlížet jako na online bin packing problem a budeme se snažit přijít na co nejlepší řešení, naimplementovat ho a vyhodnotit jeho účinnost na několika datasetech a porovnat s ostatními existujícími online i offline algoritmy.

1.1 Projekt MaMBA

Před téměř 100 lety zjednodušila Bornova–Oppenheimerova aproximace výpočty ve fyzice kondenzovaných látek oddělením pohybu jader atomů či molekul a řádově lehčích elektronů. Její porušení vede často k exotickým jevům, jako například multiferoické chování, polární uspořádání či supravodivost. Donedávna se mělo za to, že se jedná o anomální případy.

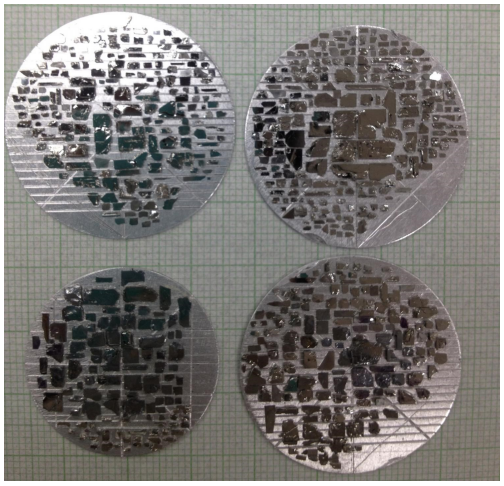
V nedávné studii (Čermák a kol., 2019) se však přišlo na množství nových hybridních magnetoelastických stavů v běžném intermetalickém krystalu, což naznačuje, že propojení jader a elektronů je mnohem důležitější, než jak na něj fyzikální svět pohlížel.

Projekt Magnetoelastic Materials beyond Born-Oppenheimer Approximation (MaMBA) je pětiletý (2021-2025) výzkumný projekt, jehož cílem je ukázat, že magnetoelastické jevy jsou obecnou vlastností fyziky pevných látek a že je potřeba důkladně zrevidovat předpoklad Borna a Oppenheimera. Věříme, že skryté magnetoelastické jevy jsou zodpovědné za spoustu nevyřešených problémů, například v těžkých fermionových sloučeninách, kde byla objevena nekonvenční supravodivost, nebo v železných pnictidech, kde porušení symetrie souvisí s magnetoelastickými efekty.

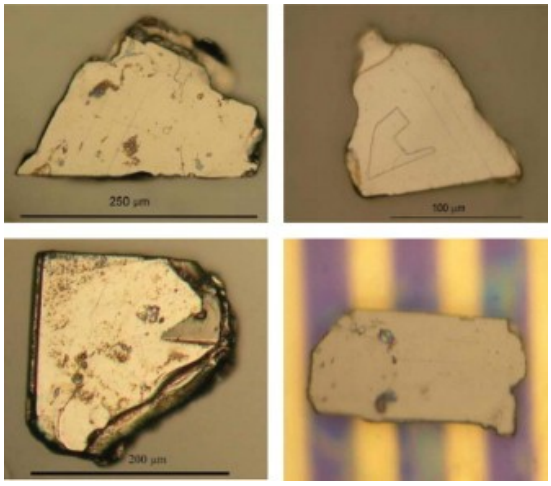
MaMBA hodlá experimentálně popsat magnetoelastické vlastnosti těchto materiálů pomocí nepružného neutronového rozptylu, který jako jediný dokáže tyto jevy identifikovat.

Nepružný neutronový rozptyl se však dá použít na vzorky o hmotnosti alespoň 1g. Bohužel, krystalky z materiálů potřebných pro tyto experimenty bývají velmi malé a dosahují mnohem nižších hmotností, a tudíž je potřeba vzít a umístit velmi blízko vedle sebe až stovky malých krystalků s podmínkou stejné krystalografické orientace. Krystalky bývají plochého tvaru a jejich orientace mívá několik os symetrie.

Ruční příprava takového vzorku jako na obrázku 1.1 je velmi časově náročná – zabírá měsíce práce. Navíc z některých sloučenin lze vytvořit tak malé krystalky (o rozměrech v řádu stovek mikrometrů, viz obrázek 1.2), že je jejich uspořádání



Obrázek 1.1: Stovky člověkem uspořádaných krystalků $CeCoIn_5$ na hliníkových destičkách. V pozadí milimetrový papír pro měřítko (Song a kol., 2016).



Obrázek 1.2: Krystalky $SmFeAsO_{1-x}F_x$, jejichž rozměry nepřesahují 250 μm (Karpinski a kol., 2009).

člověkem prakticky nereálné. Proto do hry vstupuje automatizace pomocí zařízení ALSA (Automatic Laue Sample Aligner), viz obrázky 1.3 a 1.4. Nejdříve se krystalky vyfotí pro účely zjištění jejich poloh. Potom robotická ruka s velmi vysokou přesností Mecademic Meca500 zvedne krystalek, umístí do rentgenového paprsku nejmodernějšího Laue difraktometru, který určí krystalografickou orientaci krystalku, nechá krystalek vyfotit makro kamerou, která s využitím knihovny OpenCV zjistí jeho přesný tvar, a nakonec ho umístí na hliníkovou destičku polepenou flouridovým lepidlem na místo určené naším vyvinutým bin packing algoritmem.

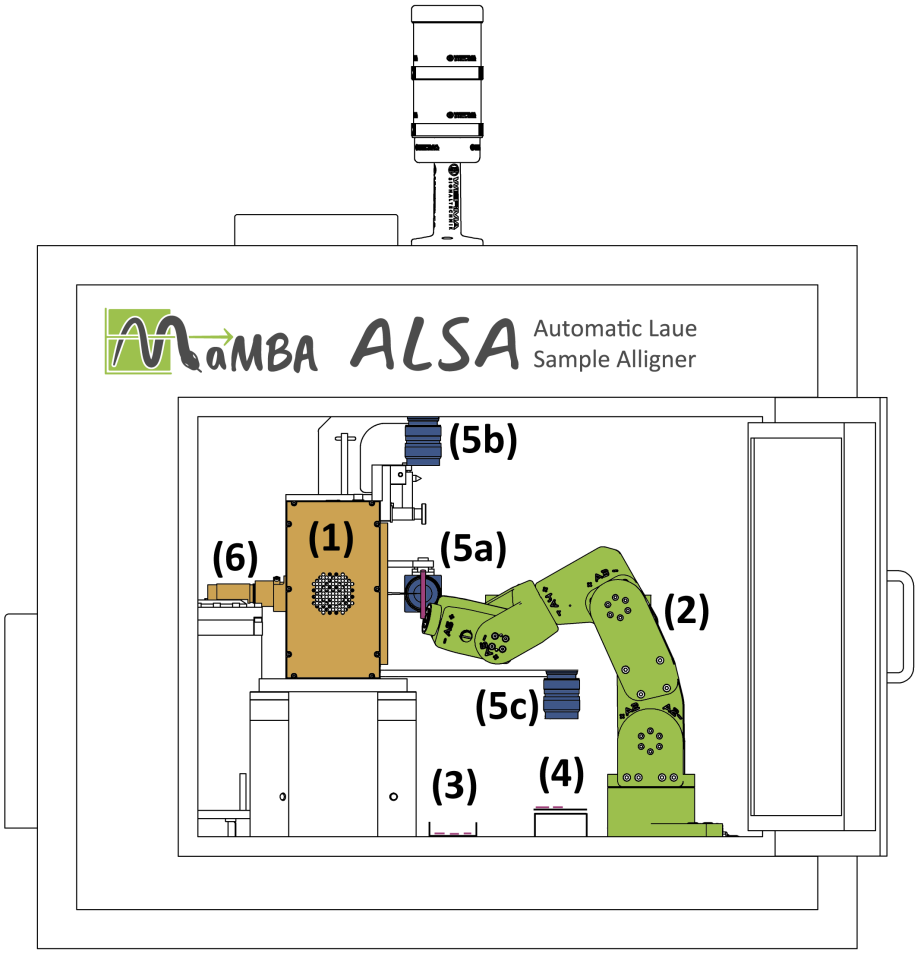
Více informací viz Čermák (2021).

1.2 Využití bin packingu v praxi

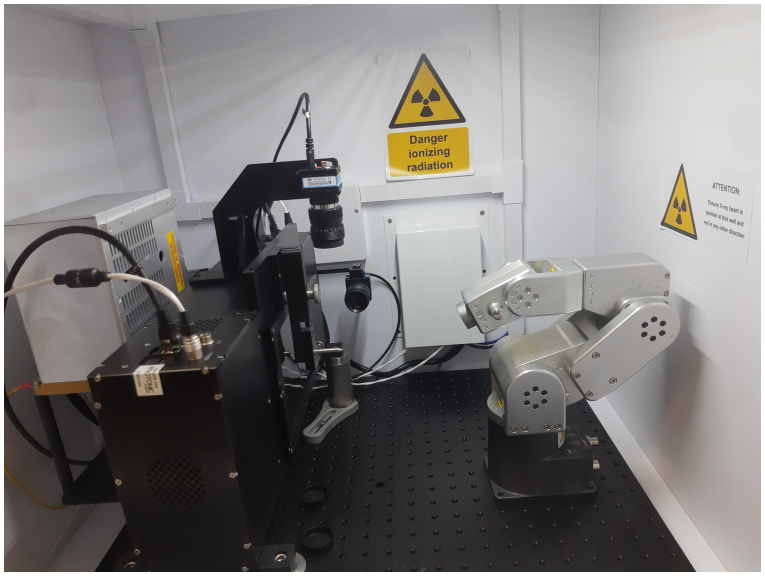
V praxi se setkáme se spoustou praktických využití řešení problému, jak na daný prostor umístit co nejvíce předmětů, aniž by se překrývaly, případně jak umístit dané předměty na co nejmenší prostor a minimalizovat nevyužité místo.

2D bin packing s libovolně tvarovanými předměty je hojně využívaný v průmyslech, kde se řežou materiály, například v oděvním průmyslu, keramickém průmyslu (Martinez-Sykora a kol., 2017), při stavbě lodí, při laserovém či vodním obrábění (Švanda, 2020; Qiao a kol., 2019) nebo třeba při 3D tisku.

Všechny zmíněné aplikace však znají celý vstup (všechny řezané části) od začátku, proto můžou používat offline algoritmus. Naše aplikace je v tomto ohledu unikátní, protože získává informace o umisťovaných předmětech v průběhu pokládání a vyžaduje tudíž online algoritmus.



Obrázek 1.3: Bokorys zařízení ALSA. (1) CCD detektor rentgenového záření, (2) Šestiosá robotická ruka Mecademic Meca500, (3) Krystalky čekající na uspořádání, (4) Hliníková destička s uspořádanými krystalky, (5a, b, c) kamery pro zjišťování a umísťování krystalků, (6) Zdroj rentgenového záření.



Obrázek 1.4: Fotografie ALSA zařízení ve výstavbě.

2. Definice a formulace problémů

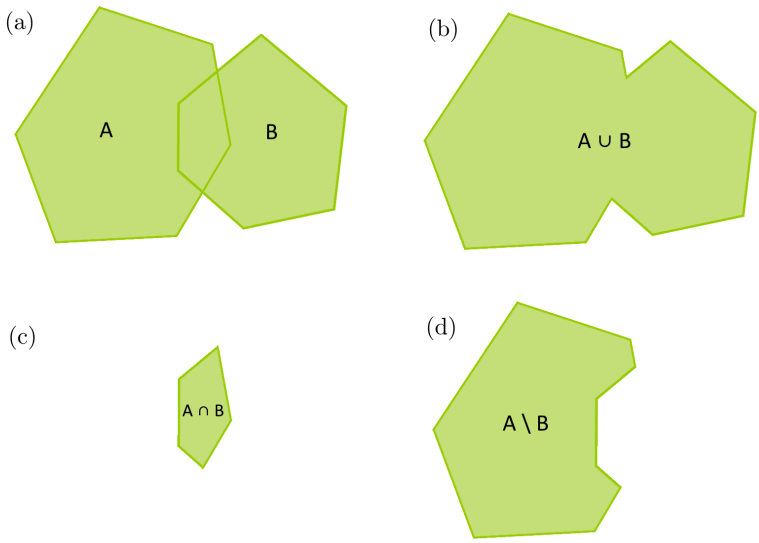
2.1 Geometrické operace

Nejprve si představíme několik geometrických operací, které budeme využívat.

Sjednocení dvou geometrických objektů A a B je geometrický objekt tvořený všemi body, které leží v A nebo v B. Formálně $A \cup B = \{x \mid x \in A \vee x \in B\}$. Viz obrázek 2.1b.

Průnik dvou geometrických objektů A a B je geometrický objekt tvořený všemi body, které leží v A a zároveň v B. Formálně $A \cap B = \{x \mid x \in A \wedge x \in B\}$. Viz obrázek 2.1c.

Rozdíl geometrického objektu A a geometrického objektu B je geometrický objekt tvořený všemi body, které leží v A a zároveň neleží v B. Formálně $A \setminus B = \{x \mid x \in A \wedge x \notin B\}$. Viz obrázek 2.1d).

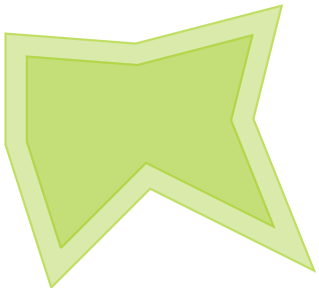


Obrázek 2.1: (a) Geometrické objekty A a B, (b) Jejich sjednocení, (c) Jejich průnik, (d) Rozdíl A a B.

Buffer (zvětšení) geometrického objektu A o vzdálenost x je operace vracející geometrický objekt, který je přibližnou reprezentací všech bodů vzdálených od A nejvýše o x (včetně bodů ležících v A). Běžné zvětšení by mělo kruhové výseče poblíž rohů A, ale my budeme používat pro naše účely postačující a zároveň produkující jednodušší geometrii zvětšení se stylem *mitre* (viz. dokumentace *Shapely*), jak můžeme vidět na obrázku 2.2.

2.2 Bin packing problem

Bin packing problem (BPP) je skupina problémů, ve kterých obecně máme množinu předmětů, které máme za úkol rozmístit do co nejmenšího počtu homo-



Obrázek 2.2: Mnohoúhelník a jeho zvětšení (o 0,5 jednotek délky) se stylem *mitre*.

genních kontejnerů (binů) tak, aby se nepřekrývaly. Původně se BPP týkal pouze 1-dimenzionálních přemětů.

Bin packing problem je definován následovně:
Vstup: Kapacita binu c, množina předmětů P, $\forall p \in P : s(p) \in (0, c]$, kde $s(p)$ je velikost (size) předmětu.
Výstup: Umístění všech předmětů $p \in P$ do m homogenních binů $B_1, B_2, \ldots, B_m$ tak, že součet velikostí předmětů v každém z binů nepřesáhne kapacitu c, tj. $\forall i \in \{1, \ldots, m\} : \sum_{p \in B_i} s(p) \leq 1$.
Cíl: Minimalizace počtu binů m.

Tento problém lze rozšířit a zobecnit do více dimenzí, kde už nebudeme řešit velikost objektu jakožto 1-dimenzionální hodnotu, ale zda se samotné předměty, které můžou nabývat libovolného tvaru, vejdou do binů, aniž by se překrývaly.

Obecný *D*-dimenzionální bin packing problem je definován takto:
Vstup: Množina d-dimenzionálních předmětů P, rozměry či jiná definice d-dimenzionálního binu jakožto geometrického objektu.
Výstup: Umístění všech předmětů $p \in P$ do m homogenních binů $B_1, B_2, \ldots, B_m$ takové, že:

- jednotlivé předměty se nemůžou překrývat, ale můžou se dotýkat, čili jejich průnik je nejvýše $(d-1)$-dimenzionální.
- každý předmět se musí celý vejít do daného binu – rozdíl každého z předmětů a binu je prázdná množina.
- předměty nelze dělit na části.

Cíl: Minimalizace počtu binů m.

Existuje množství variant a dělení BPP, například:

- **Online a offline algoritmus:** offline algoritmus zná dopředu celý vstup, zatímco online algoritmus dostává předměty postupně, tj. dostane následující předmět až po tom, co umístí aktuální.
- **Množství binů:** množství binů buď není omezené a cílem je minimalizovat počet binů, nebo je omezené a cílem je umístit podmnožinu předmětů P tak, aby suma obsahů (v případě 2D, jinak jeho d-dimenzionální analogie) všech umístěných předmětů byla co nejvyšší. V případě omezení se o problému někdy mluví i jako o (multiple) Knapsack problem – problému (více) batohu(ů).
- **Povolené rotace:** při umisťování předmětů s nimi nejde otáčet, jde otáčet pouze omezeně (v případě 2D např. rotace o 180° – 2 možné polohy, o 90°

– 4 možné polohy, nebo třeba 72° – 5 možných poloh), nebo jde otáčet neomezeně.
- **Tvar předmětů:** může být restrikce například na obdélníkové (případně d-dimenzionální analogie) předměty, na konvexní předměty, nebo bez restrikce povolující i nekonvexní tvary – označuje se to jako irregular BPP. Z výpočetního hlediska se všechny tvary nejčastěji aproximují pomocí mnohoúhelníků (polygonů) či jejich d-dimenzionálních analogií. V textu budeme o 2D předmětech mluvit i jako o polygonech.
- **Strip packing:** Máme jednu 2D nádobu s pevně danou šířkou a chceme minimalizovat výšku nádoby takové, že se do ní vejdou všechny předměty.
- **Rectangular BPP:** nádoba i předměty jsou ve tvaru obdélníku či jeho více-dimenzionální variantě, jejich hrany jsou rovnoběžné s osami souřadnic, takže jsou povolené rotace pouze o 90°.

2.3 Formulace našeho problému a cíle

V našem konkrétním případě potřebujeme vyřešit následující problém:
Máme krystalky (malé monokrystaly pevný látek), které potřebujeme položit a nalepit na kruhovou destičku (bin). Krystalky sice nemají úplně náhodné tvary, ale bývají nekonvexní. Krystalky se musí umístit tak, aby byly stejně krystalograficky orientované, přičemž díky několika osám symetrie krystalografické orientace bude umožněno několik rotací. Vzhledem k tomu, že při lepení je potřeba destičku nejdříve nahřát, tak nedává smysl mít částečně zaplněných několik binů a mezi nimi se střídat – nejdřív tedy zaplníme jednu destičku, ke které se pak už nebudeme vracet, potom druhou atd., čili budeme opakovaně řešit problém s počtem binů omezeným na 1. Vzhledem k tomu, že robotická ruka vezme krystalek, nechá zjistit jeho orientaci a přesný tvar a pak hned, aniž by ho pustila, umístí na destičku, budeme uvažovat online variantu algoritmu.

Chceme tedy řešit online 2-dimenzionální irregular BPP s 1 binem a omezenými rotacemi.

Cílem bakalářské práce je přijít s co nejlepším řešením tohoto problému a jeho implementací, přičemž implementace by měla být kvůli použití v praxi rozumně rychlá, což v našem případě znamená průměrně maximálně zhruba 10 vteřin na umístění jednoho předmětu pro instance problému o vysokých desítkách až nižších stovkách předmětů.

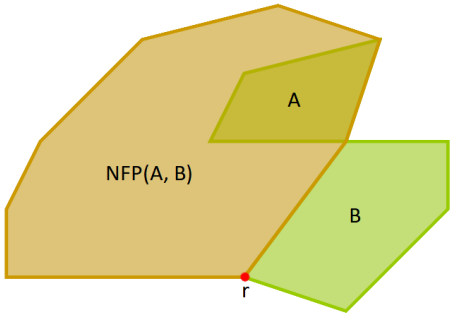
Zajímalo by nás také, o kolik lepší řešení bychom dostali, kdybychom místo online algoritmu použili offline algoritmus, který by v praxi znamenal mnohem více práce – každý krystalek by se musel pro zjištění tvaru a krystalografické orientace měřit 2x – poprvé, aby se získala data pro vstup offline algoritmu, a podruhé, aby naměřily přesné hodnoty pro účely položení na samotnou destičku, protože předpokládáme, že se krystalek mírně pootočí nebo posune během odkládání zpátky po prvním měření a následném zvedání.

Dalším cílem bakalářské práce je tedy pokusit se o odhad rozdílu efektivity řešení nejlepšího nalezeného online algoritmu a nejlepších dostupných offline algoritmů.

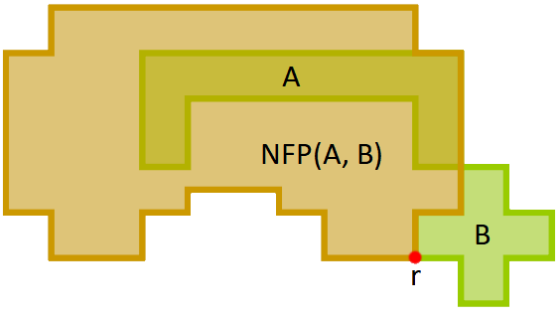
2.4 No-fit polygon, Inner-fit polygon a Collision-free region

V této sekci si představíme několik geometrických struktur používaných u 2D BPP.

No-fit polygon: Mějme 2 polygony A a B v nějaké počáteční pozici a referenční bod r polygonu B. No-fit polygon (NFP) je množina bodů takových, že pokud přesuneme r (spolu s celým polygonem B) do nějakého bodů této množiny, tak $A \cap B$ bude neprázdný. Příklady NFP jsou na obrázcích 2.3 a 2.4.



Obrázek 2.3: Polygony A a B, referenční bod r a jejich $NFP(A, B)$.



Obrázek 2.4: NFP s nekonvexními tvary.

Pokud přesuneme r na hranici NFP, tak se A a B budou pouze dotýkat, pokud r přesuneme dovnitř, tak se budou A a B překrývat, pokud mimo, tak A a B budou ležet vně sebe (budou mít prázdný průnik).

Všimněme si, že pokud změníme referenční bod r posunutím o nějaký vektor $(v_1, v_2) \in \mathbb{R}^2$, tak výsledný NFP se také posune o stejný vektor (v_1, v_2). Díky tomu je jedno, který bod vybereme jako r - může to být například střed B, vrchol B nebo třeba levý dolní roh bounding boxu (nejmenšího obdélníku, do kterého se polygon vejde) B.

Jak uvádí Nguyen Huu (2015), relace NFP pro nějaký pevně daný referenční bod r je antisymetrická - translace o vektor (v_1, v_2) bodu r (jako referenčního bodu B) vzhledem k A odpovídá translaci $-(v_1, v_2)$ bodu r (jako referenčního bodu A) vzhledem k B. Pro polygony A, B v iniciální pozici a společný referenčního bodu r platí:

$$NFP(A, B) = -NFP(B, A)$$

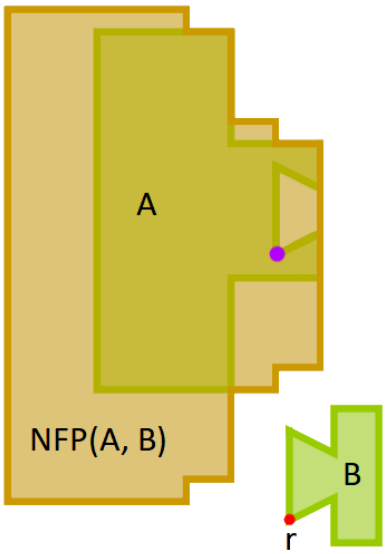
Tato skutečnost se využívá zejména v offline algoritmech, kde se počítá vzájemný NFP všech vstupních polygonů.

Co se týče rotací polygonů, platí, že rotace $NFP(A, B)$ o úhel θ odpovídá NFP s oběma polygony A, B zrotovanými o úhel θ. NFP s jenom jedním z polygonů A, B zrotovaným je obecně tvarově odlišný od $NFP(A, B)$.

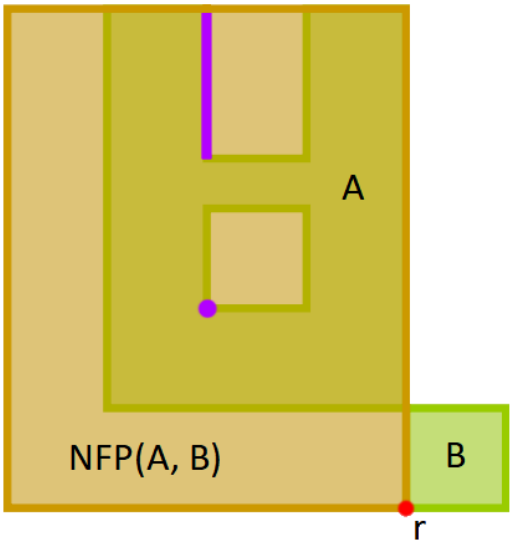
Čím je NFP z geometrického hlediska?
Nejdříve si upřesněme pojem polygon. Polygon je definován jako rovinný útvar, který je popsán jako oblast roviny ohraničené uzavřeným řetězem konečně mnoha úseček. Každý polygon, o kterém se budeme bavit, je jednoduchý (simple), čili

jeho strany se neprotínají, a má kladný obsah. Říkejme takovému polygonu běžný polygon.
(Obecný) polygon či polygon s dírami (polygon with holes) je běžný polygon, který v sobě může obsahovat díry tvořené obecnými polygony, které ale nesmí narušit spojitost tvaru.
Pro běžné konvexní polygony A, B je $NFP(A, B)$ také běžný konvexní polygon. Pro obecné polygony to už je těžší. Bereme-li v úvahu hranice NFP, tak ty většinou tvoří množina hranic běžných polygonů, ale lze narazit i na tzv. degenerované případy:

- pokud A obsahuje díru tvořenou stejným tvarem jako je B, tak hranice NFP bude obsahovat bod - místo, na které když přesuneme r (spolu s B), tak se B vejde přesně do té díry a bude se polygonu A pouze dotýkat, viz obrázek 2.6. Hranice NFP může obsahovat bod i v případě, kdy je část vnější hranice A tvarovaná přesně jako část polygonu B, viz obrázek 2.5.
- A může obsahovat i díry nebo tak tvarovanou část vnější hranice, že B tam „přesně zapadne“ a může se pohybovat pouze po úsečce, aby se nepřekrýval s A. V tomhle případě bude hranice NFP obsahovat úsečku, která alespoň na jednom konci není na nic napojená, viz obrázek 2.6



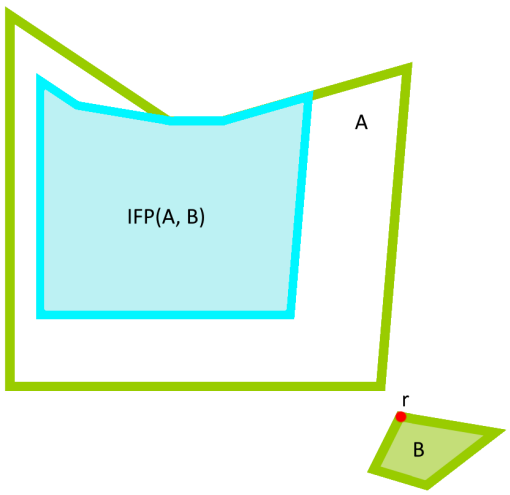
Obrázek 2.5: NFP s degenerovaným, fialově zvýrazněným, bodem. A má tak specifickou hranici, že se B dotýká A v místě fialového bodu, ale ne v jejím okolí.



Obrázek 2.6: A má díru přesně odpovídající tvaru B (tvoří bod v hranici NFP), a navíc nekonvexní hranici odpovídající velikosti B, že v hranici NFP vytvoří degenerovanou úsečku.

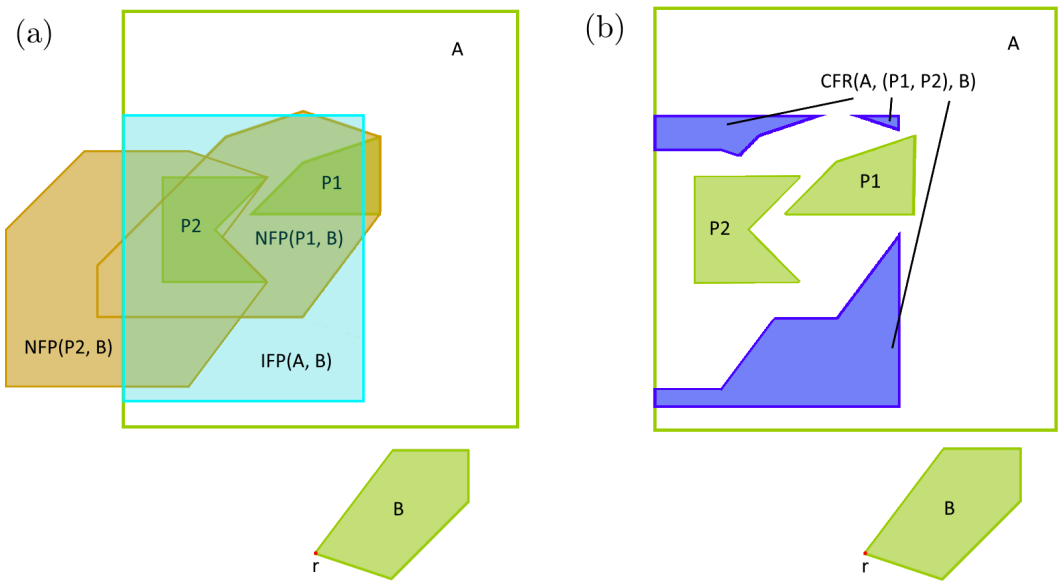
Inner-fit polygon: Inner-fit polygon (IFP) je v jistém smyslu opak NFP. Zatímco NFP je množina míst, ve kterých se se B dotýká nebo překrývá s A, tak IFP je množina míst, ve kterých je B celý uvnitř, čili $A \cup B = A$. Hranice IFP jsou body, ve kterých se B dotýká hranice A, ale zevnitř, viz obrázek 2.7. Polygon A se v tomto kontextu označuje jako nádoba (container).

Máme-li konvexní nádobu bez děr, je výsledný IFP také běžný konvexní polygon.



Obrázek 2.7: Příklad IFP.

Collision-free region: Mějme nádobu A, množinu již umístěných polygonů $P_1, ..., P_k$ a polygon B s referenčním bodem r. Collision-free region (CFR) je množina bodů, že pokud přesuneme r (spolu s B), tak B bude ležet celé uvnitř A a zároveň se B s žádným z polygonů $P_1, ..., P_k$ nepřekrývá. Čili $CFR(A, (P_1, ..., P_k), B)$ je množina takových bodů, pro které platí, že B leží celé v $IPF(A, B)$ a zároveň pro všechna $i = 1..k$ leží B vně nebo se pouze dotýká $NFP(P_i, B)$, resp. leží vně nebo se pouze dotýká $\cup_{i=1}^{k} NFP(P_i, B)$. Alternativně $CFR(A, (P_1, ..., P_k), B) = IFP(A \setminus (\cup_{i=1}^{k} P_i), B)$. Viz obrázek 2.8.



Obrázek 2.8: Příklad výpočtu CFR. (a) Polygon B a jeho referenční bod r (červeně) a jejich NFP (oranžově) s polygony P_1 a P_2 a IFP (azurově) s nádobou A. (b) Výsledný CFR je tvořen 3 fialově vyznačenými polygony.

2.5 Competitive ratio

V této sekci si představíme competitive ratio – vyjádření, o kolik minimálně je daný online algoritmus horší než optimální offline algoritmus.

Mějme algoritmus A a optimální offline algoritmus OPT pro nějaký BPP. Počet binů použitých algoritmem A, resp. OPT při vstupu V označme jako $A(V)$, resp. $OPT(V)$.
Tzv. competitive ratio (CR) pro vstup V je:

$$\frac{A(V)}{OPT(V)}.$$

Absolute competitive ratio je supremum competitive ratia pro všechny vstupy:

$$\sup_V \Big\{\frac{A(V)}{OPT(V)}\Big\}.$$

Asymptotic competitive ratio je supremum competitive ratia pro vstupy, jejichž optimální počet binů se limitně blíží nekonečnu:

$$\lim_{n\to\infty} \sup_V \Big\{\frac{A(V)}{OPT(V)} \;\Big|\; OPT(V) \geq n\Big\} = \limsup_{n\to\infty} \max \Big\{\frac{A(V)}{OPT(V)} \;\Big|\; OPT(V) = n\Big\}.$$

Asymptotic CR je vždy menší nebo rovno absolute CR.

Absolute a asymptotic CR se někdy zapisují jako hodnota závislá na optimálním počtu binů OPT, čili např. $a \cdot OPT + b$.

3. Rešerše oboru

V této kapitole se budeme zabývat teoretickou rovinou, výzkumem a výsledky i praktickými metodami řešení BPP.

BPP je velmi známý NP-těžký problém (Garey a Johnson, 1978). Dle průzkumu trhu založeném na sledování internetového provozu (Skiena, 1999) je ze všech NP-těžkých problémů jeden z 5 nejpopulárnějších a jeho implementace dokonce nejžádanější.

Literatura ohledně BPP by se dala rozdělit do 2 skupin – na tu, která na problém nahlíží spíše z teoretického hlediska a zkoumá a dokazuje absolute či asymptotic CR daných algoritmů či skupin algoritmů, a na tu, která ho řeší více z praktického hlediska a vyvíjí různé metody a heuristiky, jejichž efektivitu porovnává s jinými algoritmy pomocí výsledků dosažených na sadách konkrétních vstupů.

3.1 1D BPP

V oblasti 1D BPP existuje několik jednoduchých hladových online algoritmů – Next-Fit, First-Fit, Best-Fit, Worst-Fit. Všechny se snaží umístit předmět do nějakého otevřeného binu podle svého kritéria, pokud žádný otevřený bin nevyhovuje, tak otevřou nový a umístí předmět tam.
Next-Fit se podívá, jestli je možné předmět umístit do posledního otevřeného binu, pokud ano, tak ho tam umístí.
First-Fit umisťuje předmět do prvního binu, který ho dokáže pojmout.
Best-Fit umisťuje předmět do nejvíce zaplněného binu, který ho pojme.
Worst-Fit naopak umisťuje předmět do nejméně zaplněného binu, který ho pojme.

U všech zmíněných algoritmů se můžeme setkat i s offline „Decreasing“ variantou, která si na začátku setřídí předměty na vstupu od největšího po nejmenší.

Všechny zmíněné algoritmy lze implementovat v $\mathcal{O}(|P| \cdot log(|P|))$ (Johnson, 1973), Next-Fit navíc zjevně v $\mathcal{O}(|P|)$.

Dósa a Sgall dokázali o First-Fit a Best-Fit, že jejich absolute CR je přesně $\lfloor 1.7 \cdot OPT \rfloor$ (Dósa a Sgall (2013), Dósa a Sgall (2014)).

Aktuálně jsou online algoritmy s nejlepším asymptotic CR více či méně založené na algoritmu Harmonic-k.
Harmonic-k: každému předmětu je přiřazena jedna ze tříd $(0, \frac{1}{k}]$,$(\frac{1}{k}, \frac{1}{k-1}]$, …, $(\frac{1}{3}, \frac{1}{2}]$, $(\frac{1}{2}, 1]$ odpovídající jeho velikosti. Pro každou třídu existuje separátní množina binů, do kterých se vkládají pouze předměty dané třídy, a to pomocí Next-Fit.

Algoritmus v každém okamžiku uvažuje (má otevřených) maximálně k binů. Asymptotic CR je pro $k \to \infty \approx 1.691$ (Lee a Lee, 1985).

Algoritmus s aktuálně nejlepším asymptotic CR 1.5783 je Advanced Harmonic od Balogha a kol. (Balogh a kol., 2017).

Tentýž tým představil nejlepší dolní odhad asymptotic CR 1.54278 (Balogh a kol., 2018).

Podobný tým přišel i s nejlepším a zároveň optimálním absolute CR $\frac{5}{3}$ (Balogh a kol., 2019).

U offline polynomiálních algoritmů k aktuálně nejlepšímu výsledku přišli Hoberg a Rothvoß (2015) s algoritmem založeným na teorii diskrepance a 2-fázového balení používajícím $OPT + \mathcal{O}(\log OPT)$ binů.

3.2 2D rectangular BPP

U 2D rectangular BPP také existuje několik druhů jednodušších algoritmů, jejich přehled uvádí Kysela (2014).

Policové (shelf) algoritmy jsou založené na tom, že se v binech otevírají police, což jsou horizontální pruhy s výškou určenou prvním vloženým předmětem do dané police.

Předmět je uložen „na výšku“, pokud je jeho delší strana uložena rovnoběžně s levou stranou binu, „na šířku“, pokud je jeho delší strana rovnoběžně s dolní stranou binu.
Když se ukládá předmět do dané police, tak, pokud je prvním předmětem v dané polici, se uloží na šířku kvůli minimalizaci výšky police. Jinak se ukládá na výšku, vejde-li se, v opačném případě na šířku.

Policové algoritmy vlastně převádějí 2D problém na téměř 1D problém pomocí dělení binu na police, proto se používají velmi podobné algoritmy jako u 1D BPP – **Shelf Next Fit** ukládá do poslední otevřené police, **Shelf First Fit** do první police, do které se vejde, **Shelf Best Width Fit** do police, do které se vejde a zároveň jí zbývá nejméně horizontálního místa, **Shelf Worst Width Fit** naopak tam, kde zbývá nejvíce horizontálního místa.
Potom existují další méně podobné algoritmy – **Shelf Best Height Fit** vybírá polici, do které se vejde a zároveň rozdíl výšky předmětu a výšky police bude minimální, **Shelf Best Area Fit** v případě shodného rozdílu výšek předmětů vybere umístění na výšku. **Shelf Floor-Ceiling** si seřadí předměty sestupně podle rozměru delší strany, uloží je na dno police zleva doprava, a potom na strop téže police zprava doleva.

Gilotinové algoritmy fungují na rozdělování zbylého volného místa na obdélníky. První předmět uloží do levého dolního rohu binu. Potom zbývající místo rozdělí na dva obdélníky pomocí přímky danou pravou nebo horní stranou předmětu. Následující předmět umístí do jednoho z těchto dvou volných obdélníků atd. až narazí na předmět, který se nevejde do ani jednoho volného obdélníku.

Existuje několik algoritmů vybírajících obdélník, do kterého vložit předmět – **Guillotine Best Area Fit** vybírá nejmenší možný obdélník, do kterého se předmět vejde, **Guillotine Best Short Side Fit** vybírá obdélník, u kterého je

nejmenší minimum z rozdílu délek stran obdélníku a předmětu a rozdílu jejich šířek, **Guillotine Best Long Side Fit** naopak minimalizuje maximum z rozdílu délek a rozdílu šířek stran.
Guillotine Worst Fit se snaží o opak oproti výše zmíněným algoritmům, čili maximalizaci daných parametrů.
U těchto algoritmů se ještě využívá **Rectangle Merge Improvement**, které po umístění každého předmětu hledá, zda je možné nějaké 2 sousedící volné obdélníky, jejichž sjednocení je také obdélník, a případně je sloučí.

Maximal Rectangles algoritmy jsou modifikací gilotinových algoritmů, a fungují na principu, že provedou dělení jak pomocí přímky dané pravou stranou, tak přímky dané levou stranou, a z každého z těchto dvou dělení si vyberou ten větší obdélník, akorát musí řešit to, že se tyto volné obdélníky překrývají.
Zajímavým algoritmem z této skupiny je **Maximal Rectangles Contact Point**, který vybírá volný obdélník podle toho, jak dlouhá je délka doteku předmětu s binem či již uloženými předměty.

Velkým zrychlením oproti Maximal Rectangles algoritmům je použití datové struktury, která udržuje pouze množinu horizontálních panoramatických hran, tvořenou hranami či jejich úseky, nad kterými není umístěný žádný předmět. Objevují se zde algoritmy **Skyline Bottom-Left**, který vybere umístění na nejnižší možné panoramatické úrovni zarovnané co nejvíce nalevo, a **Skyline Best Fit**, který vybere pozici, u které po uložení ztratíme nejméně místa v panoramatické datové struktuře, v případě více pozic se použije Bottom-Left. Místo se ztrácí, kdy je část předmětu uložena na nějakém již umístěném předmětu, ale pod zbylou částí je volné místo.

Problémy ztraceného volného místa se dají omezit použitím **Waste Map** – gilotinovou strukturou, která si pamatuje ono ztracené volné místo a umožňuje ho danému algoritmu použít.

Co se týče algoritmů s nejlepším asymptotic CR, tak nejlepších offline výsledků dosáhli Bansal a Khan (2014) s asymptotic CR $ln(1{,}5) + 1 \approx 1{,}405$ pro varianty s rotacemi o 90° i bez rotací. Hlavní myšlenka algoritmu je založena na použití tzv. Round & approx frameworku (Bansal, Caprara a Sviridenko, 2009) na výsledcích od Jansena a Prädela (Jansen a Prädel, 2013).

Nejvyšší dokázaný dolní odhad asymptotic CR pro polynomiální offline algoritmy je však $1 + 1/3792$ (s rotacemi o 90°) a $1 + 1/2196$ (bez rotací), pokud $\mathsf{P} \neq \mathsf{NP}$ (Chlebík a Chlebíková, 2006).

V online variantě má nejlepší asymptotic CR 2,25 algoritmus Epsteinové a van Steeho (Epstein a van Stee, 2006) a nejvyšší dokázaný dolní odhad asymptotic CR je 1.9100449 (Epstein, 2019).

Pro online variantu s restrikcí na konstantou omezený počet aktivních binů (space bounded) přišla Epsteinová s algoritmem s asymptotic CR 2,54679 a dolním odhadem asymptotic CR 2,53536 (Epstein, 2010).

Spoustu dalších výsledků, včetně těch pro více-dimenzionální rectangular BPP, lze najít v průzkumu od Christensena a kol. (Christensen a kol., 2017).

3.3 Obecný 2D BPP

Zatímco v 1D BPP a 2D rectangular BPP se objevuje spousta algoritmů, u kterých jsou dokázané teoretické vlastnosti jako CR, u 2D BPP se složitejšími obecnými tvary – ať už konvexními, nebo i nekonvexními – jsme na žádnou práci, která by dokazovala CR, nenarazili. 2D irregular BPP je zjevně mnohem těžší problém z hlediska analýzy CR a není ani v rámci našich možností a znalostí se o ni pokoušet. Budeme tedy porovnávat efektivitu našeho algoritmu pomocí dosažených výsledků na sadách vstupů.

Navzdory absenci teoretické analýzy je o 2D irregular BPP obrovský zájem z praktického hlediska zejména díky důležitému využití v průmyslu, kde dochází k řezání materiálu. I díky finanční motivaci zminimalizovat odpad při řezání materiálu vzniklo velké množství přístupů a algoritmů.

U 2D irregular BPP je výběr binů spíše minoritní problém – většinou se používá First-Fit nebo Next-Fit, případně některé algoritmy pracují pouze s jedním binem, což se dá pojmout jako Next-Fit pro případ potřeby více binů.

Mnohem důležitější je způsob, jak umístit předměty dovnitř binu. Takový jednoduchý online algoritmus může být „tetris“ algoritmus – začneme s umístěním předmětu v pravém horním rohu nádoby a střídavě posouváme předmět směrem dolů a doleva tak daleko, dokud nenarazíme na jiný již umístěný předmět či stranu nádoby.
Dalším spíše základním online algoritmem je vypočíst si CFR a vybrat jeden z jeho vrcholů pomocí tzv. picking policy **Bottom Left** – vybrat vrchol (kandidáta), který je nejníže (má nejnižší souřadnici y), v případě více takových kandidátů vybrat toho, který je nejvíce nalevo (má nejnižší souřadnici x).

Offline algoritmy nezřídka používají nějaké počáteční konstruktivní (online) řešení, a potom aplikují offline heuristiky. Počáteční řešení se většinou konstruuje na vstupu setříděném od největší plochy po nejmenší.

Práce zaměřené hlavně na zlepšení počátečního řešení většinou představují nějakou picking policy.
Dalalah, Khrais a Bataineh (2014) představili picking policy založenou na minimalizaci konvexního obalu již uložených polygonů a toho právě pokládaného.
Torres a kol. (2020) navrhl picking policy pro konvexní polygony, která vybírá kandidáta, po jehož umístění vznikne nejmenší uzavřená oblast volného prostoru vytvořená umístěním kandidáta.
Liu a He (2006) uvedli picking policy založenou na nejníže položeném těžišti.
Oliveira a kol. (2000) prezentoval picking policy založené na minimalizaci bounding boxu již uložených polygonů + kandidáta, na největším překrytí bounding boxů kandidáta s bounding boxy již uložených polygonů a na nejmenší vzdálenosti středu bounding boxu kandidáta a středu bounding boxu všech již uložených polygonů.

Potom je zde množství offline heuristik:
Vidal (1993) aplikoval na 2D irregular BPP techniku **Simulated Annealing**, což je pravděpodobnostní optimalizační metoda, ve které se prohledává stavový prostor řízeným způsobem pomocí ovládacího parametru, který je analogie tep-

loty – čím větší teplota, tím větší změny.
Błażewicz, Hawryluk a Walkowiak (1993) jako první aplikovali tzv. **tabu search** – metodu, která využívá lokálního prohledávání s využitím seznamu tabu se zakázanými, již navštívenými řešeními.
Shalaby a Kashkoush (2013) navrhli aplikaci Particle Swarm Optimization.
Spousta heuristik je zaměřená na použití genetických algoritmů založených na generacích populace, kde z nejlepších jedinců (v našem případě daná uložení) a jejich mutací se tvoří generace následující. Využívá toho například Jakobs (1996), Hifi a M'Hallah (2003) nebo jíž zmínění Liu a He (2006).

Zajímavá je aplikace Djang and Finch heuristiky, původně používané pro 1D BPP (Lopez a kol., 2013). Djang and Finch heuristika plní biny tak, že nejdříve zaplní alespoň třetinu binu předměty seřazenými od největšího a poté se snaží najít kombinaci jednoho, dvou nebo tří předmětů, po jejichž umístění by byl bin přesně zaplněný. Pokud žádná taková kombinace neexistuje, tak se snaží najít kombinaci na zaplnění alespoň kapacity binu zmenšené o konstantu, o 2násobek konstanty atd.

Další skupinou jsou algoritmy zaměřené na 2D irregular BPP s libovolnými rotacemi – s algoritmem na výpočet nejlepší rotace přišel Nye (2000), aplikaci Jostle heuristiky na problém s libovolnými rotacemi představili Abeysooriya, Bennell a Martinez-Sykora (2018) a Martinez-Sykora a kol. (2017) vlastní heuristiku s využitím lineárního programování.

Pozoruhodnou skupinou jsou i hyper-heuristiky – heuristiky založené na výběru konkrétních heuristik pro daný problém. Hyper-heuristiky založené na genetických algoritmech představili Terashima-Marín a kol. (2010) a Sosa a kol. (2016).

Problémem však je, že i když nějaké množství z výše zmíněných prací obsahuje informace o naměřených výsledcích na veřejně známých datasetech, tak prakticky žádná z nich neposkytuje zdrojový kód, což značně komplikuje jejich vzájemné porovnání na nově vytvořených datasetech. Výjimkou jsou v tomto ohledu například Wang Zilu a Yang Shan, kteří publikují výsledky svého výzkumu, například využití reinforcement learningu na 2D irregular BPP (Zilu a Shan, 2020), včetně zdrojového kódu na platformě Github. Za zmínku stojí rovněž open-source implementace jedné či více prací jako SVGnest (Qiao a kol., 2019) nebo Dalsoo-Bin-Packing (Hua, 2020).
Snaze určit nejlepší algoritmus nepomáhá ani to, že některé komerční proprietární programy jsou velmi efektivní, možná dokonce nejefektivnější vůbec, ale nevíme u nich, jaký se použil algoritmus, ani nakolik je použitý algoritmus založen na již existujících publikovaných heuristikách.

3.4 No-Fit Polygon

V této sekci zmíníme několik způsobů, jak počítat NFP. V praxi se používají v zásadě 3 přístupy: s použitím Minkowského sumy, pomocí tzv. orbital sliding approach a pomocí konvexní dekompozice.

Minkowského suma dvou množin polohových vektorů (v našem případě vrcholů polygonů) A a B je tvořena sečtením každého vektoru z A s každým vektorem z B:

$$A \oplus B = \{a + b \mid a \in A, b \in B\}.$$

NFP dvou konvexních polygonů A a B, oba se stejnou orientací vrcholů (běžně proti směru hodinových ručiček) je ekvivalentní $A \oplus -B$, kde $-B$ je polygon s opačnou orientací vrcholů (viz Bennell, Dowsland a Dowsland, 2001). NFP dvou konvexních polygonů lze vypočíst v $\mathcal{O}(m + n)$, kde m, n jsou počty vrcholů polygonů A, B (Kaul, 1991).
V případě nekonvexních polygonů je tento problém mnohem těžší a je potřeba správně zjistit, které hrany vytvořené $A \oplus -B$ patří do NFP a jaká je jejich posloupnost. Robustní algoritmus navrhli Bennell a Song (2008). Uvádějí časovou složitost $\mathcal{O}(m^2n^2log(m^2n^2))$.

Orbital sliding approach je přístup založený na fyzickém posouvání jednoho polygonu kolem druhého. Je založen na výpočtu dotýkajících se vrcholů a hran, stanovení vektoru posunu a výpočtu délky posunu. Algoritmus navrhl například Burke a kol. (2007).

Konvexní dekompozice je přístup, kdy se konkávní polygony nejdřív rozloží na několik konvexních polygonů, pomocí nich se spočítají příslušné částí NFP a potom se rekombinují dohromady. Tento přístup volil Rocha (2019).

4. Vlastní algoritmus

4.1 Naivní vlastní přístup

Vývoj algoritmu pro 2D irregular BPP začal již v rámci Studentského fakultního grantu, ve kterém bylo za cíl vyvinout vlastní implementaci pro ukládání polygonů do kulaté nádoby. Tehdy vznikl náš naivní algoritmus, vytvořený bez znalosti existence NFP.

Již tehdy se zdálo jako vhodný přístup si nejdřív nějakým způsobem vypočíst vhodné kandidáty, a potom z nich pomocí picking policy vybrat nejlepšího.

Po celou dobu běhu si algoritmus udržoval datovou strukturu tvořenou množinou polygonů reprezentující neobsazené místo v nádobě (kruh aproximovaný pravidelným n-úhelníkem).

Pro každý takový volný polygon, který byl menší než právě pokládaný polygon (tvar), a každou dovolenou rotaci tvaru se spočítali kandidáti, kteří se volného polygonu alespoň 2 svými body.
Ti se spočítali tak, že pro každou úsečku volného polygonu a každou kombinaci bod-úsečka tvaru se spočítal kosodélník vzniklý pohybem úsečky tvaru při posouvání tvaru po úsečce volného polygonu, přičemž se úsečky volného polygonu dotýkal daným bodem. S využitím rozdílu kosodélníku a volného polygonu se vypočítalo místo, ve kterém se úsečka tvaru dotýkala volného polygonu. Pokud v takovém místě nepřesahoval tvar hranice volného polygonu, tak se takové místo přidalo do seznamu kandidátů.

Potom přišel na řadu výběr nejlepšího kandidáta. Po odstranění stejných kandidátů se nejdřív vybrala množina kandidátů s nejmenším vzniklým waste – pro kandidáty, kteří by daný volný polygon svým umístěním rozdělili na více částí, se vzal rozdíl plochy volného polygonu a odečetla se od něj plocha tvaru a největší vzniklé části, pro ostatní se vzala 0. Potom se vybrala množina kandidátů, jejichž umístění by nejméně zvýšilo obvod již položených polygonů a nádoby. Nakonec se vybral kandidát, po jehož umístění by vznikl nejmenší konvexní obal sjednocení položených polygonů.

Tento algoritmus sice neprodukoval zas tak dobré výsledky, ale největší jeho slabinou byla rychlost – pro instance, kde by položil více než 150 polygonů, běžel v řádu hodin, což bylo nepoužitelné. Zdaleka nejvíce času trávil ve výpočtu kandidátů, což bylo primárně potřeba zrychlit.

4.2 Výpočet kandidátů pomocí CFR

Výpočet kandidátů se podařilo velmi významně urychlit pomocí výpočtu CFR (rozdíl IFP pokládaného tvaru a sjednocení NFP tvaru s jednotlivými již položenými polygony) – pro každou povolenou rotaci tvaru se vypočte CFR a kandidáti jsou tvary umístěné ve vrcholech CFR. Pozice ve vrcholech CFR jsou obecně

považovány za dobré – spadají tam všechny pozice, kdy se umisťovaný tvar při pomyslném posouvání kolem již položených tvarů (či hranice nádoby) dostal „do rohu“, že musí změnit směr svého posouvání.

Implementovali jsme ale i 2 modifikace:

1) Sato, Martins a Tsuzuki (2012) přišli s tím, aby se ignorovaly konkávní vrcholy CFR, s argumentem, že konkávní vrcholy mají původ v konvexních vrcholech sjednocení NFP, což jsou vrcholy, kde se umisťovaný tvar dotýká již položených polygonů pouze v jednom bodě. V naší implementaci jsem pro každý polygon v CFR vybrali body ležící v jeho konvexním obalu.

2) Nguyen Huu (2015) navrhuje kromě vrcholů CFR i rovnoměrně vybírat body na stranách CFR. Rozhodli jsme se (kromě vrcholů CFR) rovnoměrně vybírat body ze stran CFR s odstupem přibližně třetiny průměrné délky strany v celém CFR.

Napadla nás ještě jedna (volitelná) možnost modifikace předvýběru kandidátů – pokud CFR obsahuje nějaké malé polygony (díry), tak chceme primárně zaplnit tyto malé díry. Pokud by ale CFR obsahoval například pouze 2 celkem velké regiony, tak nalezení nejvhodnějšího kandidáta chceme plně přenechat picking policy. Rozhodli jsme se vybírat malé díry, pokud existuje díra menší než 10násobek plochy umisťovaného tvaru.
Jak co nejlépe vybrat jenom ty nejmenší díry? Nakonec jsme se rozhodli vybírat ty díry, jejichž plocha je nejvýše rovna dvojnásobku plochy nejmenší díry + třetině plochy umisťovaného tvaru.

4.3 Picking policies

To nejdůležitější v online algoritmu je picking policy - algoritmus, který z kandidátů vybere toho nejlepšího, kam se pak nakonec umístí tvar. Vyvinuli jsme několik vlastních algoritmů a pro srovnání naimplementovali několik zmiňovaných v literatuře:

1) **Bottom left** je již zmiňovaná picking policy, která vybere kandidáta, který je nejníže položený (má nejnižší hodnoty souřadnice y), v případě více takových kandidátů vybere toho nejvíce nalevo (má nejnižší hodnoty souřadnice x).

2) **Minimal convex hull** je založený na tom, že u každého kandidáta spočítá plochu konvexního obalu sjednocení všech již položených polygonů a pokládaného tvaru a vybere vybere toho s nejmenší.

3) U **Maximal bounding box overlap** se spočítá sjednocení bounding boxů (nejmenší obdélník se stranami rovnoběžnými s osami souřadnic, do kterého se daný geometrický objekt vejde) již položených polygonů, a potom se spočte průnik bounding boxu tvaru a zmíněného sjednocení. Vybere se s největší plochou tohoto průniku.

4) Picking policy s pracovním názvem **My own old** se trochu podobá picking policy našeho naivního algoritmu a je založená na několika fázích. Nejdříve se zjistí, do kterého volného polygonu by byl kandidát umístěn, a vypočte se rozdíl

daného volného polygonu a velmi mírně zvětšeného tvaru (o $\epsilon = 2 \cdot 10^{-8}$). Z toho rozdílu získá každý kandidát 2 atributy – jak moc se zvětšil obvod (délka rozdílu − délka volného polygonu) a kolik vzniklo waste, v tomhle případě je to plocha rozdílu, pokud je rozdíl jeden polygon v celku, nebo, pokud je rozdíl tvořen několika polygony, plocha rozdílu − plocha největšího z polygonů rozdílu. Potom se vyberou takoví kandidáti, kteří mají zvětšení obvodu nejvýše rovné minimálnímu zvětšení obvodu + nějaké číslo, v našem případě desetina obvodu tvaru. Z nich se vybere kandidát s nejmenší waste.
Algoritmus je založený na myšlence, aby se primárně vybírali kandidáti, jejichž strana se přesně či téměř přesně dotýká strany nějakého již položeného polygonu, a pokud takoví nebudou (nebo budou pouze takoví, kteří se dotýkají již položeného polygonu jenom malým kouskem), tak se vybere takový který vytvoří nejmenší díru.

5) K vymyšlení picking policy **Minimal surrounding waste** nám pomohl tento způsob uvažování: chceme dát kandidáta na místo, kde bude těsně obklopen co nejvíce již položenými tvary, a ceníme si například kandidátů, jejichž strana se dotýká přesně strany nějakého již položeného polygonu (svírají úhel 0°), ale i těch, jejichž strana se nějaké strany skoro dotýká (např. svírají úhel 5°).
Pro každého kandidáta si vytvoříme jeho zvětšení o vzdálenost d a vypočteme si průnik s volným místem v nádobě. Vybereme kandidáta s nejmenší plochou takového průniku.
Vzdálenost d jsme určili následovně: po celou dobu běhu programu si udržujeme průměrnou odmocninu z plochy již položených polygonů (a toho právě pokládaného), a kandidáta zvětšíme o desetinu této hodnoty.

6) **Minimal surrounding waste plus** je vylepšením předchozí policy. Jednak jsme na základě série pokusů vylepšili na čtvrtinu zmíněné průměrné odmocniny, jednak jsme vyřešili problém, že vnějšek nádoby se nepočítal jako volné místo, a tak průnik s volným místem u kandidátů blízko hranice nádoby byl obecně menší, protože část takového zvětšeného kandidáta přesahovala hranice nádoby, a tato část průniku měla vždy nulovou plochu, a tím pádem se algoritmus skoro za každou cenu snažil umístit tvar podél hranic nádoby. Vyřešili jsme to tak, že se odděleně vypočítal průnik s volným místem uvnitř nádoby a průnik s vnějškem nádoby. Výsledné skóre je plocha průniku s volným místem + zlomek plochy průniku s vnějškem nádoby. Ze série testů nám vyšel nejlepší koeficient 0,08.

7) **Minimal convex hull and minimal surrounding waste plus** je hybrid mezi algoritmy 3) a 6). Nejdříve se vybere množina kandidátů, u kterých je plocha vytvořeného konvexního obalu menší než plocha v tomhle ohledu nejlepšího kandidáta + pětina plochy pokládaného tvaru. Potom se z této množiny vybere nejlepší kandidát na základě 6).

Poznámka: u 2) a 3) a 7) je první tvar je uložen libovolně, v naší implementaci se vybere kandidát, u něhož bude nejmenší délka rozdílu nádoby a velmi mírně (o $\epsilon = 2 \cdot 10^{-8}$) zvětšeného tvaru.

5. Technická implementace

5.1 Použité technologie

Pro implementaci jsme zvolili jazyk *Python*, který nám přišel pro vývoj řešení této úlohy nejvhodnější. V *Pythonu* se dobře píše kód, má dostupnou celou řadu knihoven a relativně jednoduché rozhraní pro volání např. *C++* kódu. Není to sice nejrychlejší jazyk, ale v případě potřeby optimalizace na rychlost lze přepsat odladěnou implementaci do kompilovaného jazyka jako je *C++*.

Z *Python* knihoven jsme využili:

- Pro geometrické operace a výpočty jsme vybrali *Shapely* založenou na *C++* knihovně pro prostorovou analýzu *GEOS* (Geometry Engine, Open Source), protože obsahuje ucelený balíček potřebných geometrických funkcí.
- Pro kreslení umisťovaných polygonů jsme využili *matplotlib* a *descartes*.
- Pro načítání dat z *.csv* souborů jsme použili *pandas*, pro převod datasetů do formátu *DXF dxfwrite* a pro čtení a následný výpočet výsledků benchmarků offline algoritmu *Packaide* (Sethapakdi a kol., 2021) nám posloužil *sgv.path*.

Nakonec jsme se ještě rozhodli použít knihovnu pro výpočet NFP a IFP. Výpočet NFP nekonvexních polygonů je složitá záležitost a je navíc potřeba ošetřit množství krajních případů, a tak se zdálo lepší použít odladěnou robustní a nejlépe i rychlou implementaci. Bohužel, v literatuře existuje několik popsaných přístupů k počítání NFP, ale žádný z nich nezveřejnil zdrojový kód.
Jediné 3 otevřené pravděpodobně robustní implementace, o kterých víme, jsou:

- Funkce *minkowski_sum_2* z *C++* knihovny *CGAL* počítá NFP i nekonvexních polygonů. Knihovnu *CGAL* jsme však ani po mnoha hodinách nebyli schopni nainstalovat na našem počítači s operačním systémem Windows 10 (poslední problém, přes který jsme se nedokázali dostat, byl ten, že instalace *CGAL* pomocí nástroje *vcpkg*, který interně používá nástroj *yasm*, vyhodila chybovou hlášku „Architecture 'x64' not supported by yasm-tool“).
- Implementace výpočtu NFP od Yang a Prinway (2020) v rámci jejich offline algoritmu. Tuto implementaci jsme nevyužili, protože v rámci pokusu spustit jejich program na malém datasetu z 50 polygonů jejich implementace nedokázala vypočíst, resp. vypočetla chybně, NFP jedné dvojice polygonů.
- *C++* knihovna *libnfporb* (Hassan a Hans, 2018) pro výpočet NFP. Výpočet NFP je založený na algoritmu založeném na orbital sliding approachi, který popsal Burke a kol. (2007). Její autoři ale museli přijít s několika improvizacemi, protože, jak tvrdí, článek od Burkeho obsahuje několik špatných předpokladů.
  libnfporb sice počítá pouze NFP, ale dokáže ho počítat pro polygony s dírami, čehož jsme využili a ve vlastní implementaci přidali do knihovny funkci pro výpočet IFP založenou na části kódu počítajícího NFP.
  Tuto knihovnu využívá také Švanda (2020), který pro ni napsal rozhraní pro volání z *Pythonu* a skripty pro kompilaci, které jsme po vlastních modifikacích také využili.

5.2 Popis implementace

Implementace se skládá z několika částí:

generator_placer_base vznikl již v době Studentského fakultního grantu, kdy každý ze 3 účastníků měl naimplementovat vlastní algoritmus pro ukládání polygonů, a tak sloužil hlavně jako rozhraní se základními třídami, v jejichž potomcích se implementovala hlavní funkcionalita.
Symmetry je třída obsahující seznam povolených rotací, resp. os symetrie.
ShapeGenerator je základní třída, která má na starosti generování tvarů za běhu (implementované jejími potomky) a umisťování do požadované nádoby (ve vstupními parametry souřadnicemi a rotací umístění), kde se provede nezávislá kontrola, zda se umisťovaný tvar nepřekrývá s již umístěnými polygony a nevyčnívá z nádoby. Kvůli četným problémům floating-point aritmetiky se nastavila tolerance, o jak velkou plochu se můžou překrývat, na 10^{-8}. V praktickém využití to nebude vadit, protože při zjišťování tvarů krystalků plánujeme rovnou započítat nějaký prostor kolem nich. Nádoba i vygenerované tvary jsou reprezentovány pomocí seznamu souřadnic (dvojice či seznam o délce 2 floating point čísel) daného polygonu. *ShapeGenerator* nedovoluje vygenerovat nový tvar, pokud stávající nebyl umístěn.
Placer je rozhraní pro třídu, která dostane nějaký *ShapeGenerator*, ze kterého si generuje tvary a kam ukládá tvary na pozice a rotace, které spočte.

shape_generators je soubor obsahující implementace všech implementací *ShapeGeneratoru*, čili: *SquareShapeGenerator* generující čtverce zadané velikosti, *LShapeGenerator* generující polygony ve tvaru připomínajícím písmeno „L“, který by šel složit ze 3 stejných čtverců, *CuttedShapeGenerator* generující tvary vzniklé rozřezáním kruhu pomocí několika přímek, *RandomShapeGenerator* generující náhodné čtyřúhelníky, *RealShapeGenerator* a *CecoinGenerator*, které mají seznam konkrétních polygonů získaných z tvarů vyfocených skutečných krystalků a po vyčerpání seznamu vygenerují nový s trochu zvětšenými či zmenšenými a zrotovanými polygony oproti původním, a *RandomHexagonShapeGenerator* generující náhodné šestiúhleníky. Poté máme třídy *SG1* až *SG7*, což jsou potomci daných generátorů se specifikovanými parametry jako je průměr či povolené rotace. Nakonec zde máme *FromCSVFileShapeGenerator*, který načte a naškáluje dataset z určeného *.csv* souboru, který obsahuje 2 sloupce – *num* udávající počet polygonů zadaného tvaru a *polygon* obsahující v úvozovkách obalenou *Pythonovou* reprezentaci seznamu souřadnic, kde každá souřadnice je seznam 2 floating point čísel (souřadnice x a y).

parse_and_save_datasets je soubor obsahující pomocné funkce pro načítání a převody mezi různými formáty datasetů, které byly potřeba pro různé offline algoritmy, které jsme porovnávali s tím naším. Díky něm jsme například i přeškálovali některé datasety na podobné rozměry či vygenerovali datasety z naších generátorů.

damians_implementation obsahuje jádro programu *DamiansPlacer* – naší implementaci *Placeru*. Pracuje s datovými strukturami *Polygon* a *MultiPolygon* (kolekce *Polygonů*) z knihovny *Shapely*. Díky ní můžeme využívat zabudované funkce jako *area* (vrací plochu daného geometrického objektu), *length* (vrací délku

obvodu), *box* (vrací bounding box), *intersection* (průnik), *union* (sjednocení), *difference* (rozdíl) a *unary_union* (rychlé sjednocení seznamu geometrických objektů).
Během celé doby běhu si *DamiansPlacer* udržuje *Polygon* nádoby (v případě kruhové nádoby ji zaproximuje pravidelným n-úhelníkem pro relativně vysoké n), seznam již umístěných polygonů či *MultiPolygon* rozdílu nádoby a již umístěných polygonů, čili datovou strukturu reprezentující volné místo v nádobě.
Funkce *run* nastaví některé parametry a nechává si generovat nové tvary a umisťovat je pomocí funkce *place_generated_shape*. Ta si pro všechny povolené rotace nechá spočítat množinu kandidátů, vybrat nejlepšího pomocí funkce *choose_the_best_candidate*, zjistí jeho rotaci, uloží ho pomocí *ShapeGeneratoru* a updatuje vlastní datové struktury s již umístěnými polygony. *choose_the_best_candidate* vybere pouze kandidáty v malých regionech CFR v případě nastaveného parametru a předá seznam kandidátů implementaci specifikované picking policy. V některých okrajových případech se stává, že vybraného kandidáta nelze umístit, v tom případě se *choose_the_best_candidate* zavolá sama sebe s odstraněným dotyčným kandidátem.

geometry_tools obsahuje několik pomocných geometrických funkcí, ale hlavně *get_CFR_candidate_points*, která je zodpovědná za výpočet kandidátů. Nejdříve si nechá s využitím volání *libnfporb* knihovny vypočíst IFP nádoby a pokládaného tvaru a pro každý již umístěný polygon NFP jeho a pokládaného tvaru. Poté spočte *unary_union* všech NFP a (pokud se nejedná o první pokládaný tvar) CFR, čili rozdíl IFP a sjednocení NFP. Potom se na základě argumentu *CFR_mode* provede následující: v případě hodnoty 0 se neudělá nic – nechá se všechny vrcholy CFR, v případě hodnoty 1 se redukují vrcholy na ty, které patří do konvexních obalů jednotlivých polygonů CFR, v případě hodnoty 2 se k vrcholů přidají i rovnoměrně umístěné body na hranách CFR.
Knihova *libnfporb* je sice relativně robustní, ale přesto v některých okrajových případech, nejspíše kvůli problémům, které s sebou přináší floating-point aritmetika, vyhodí výjimku při výpočtu NFP nebo dojde k vypočtení takových NFP, že se ve výsledném CFR neobjeví degenerované případy. Proto bylo potřeba strávit spoustu práce ošetřováním těchto případů, až nakonec vznikl poměrně robustní systém, který dokáže ošetřit pravděpodobně všechny možné výjimky a produkuje co nejsprávnější množinu kandidátů. Případ, kdy je vítězný kandidát nesprávný, je ošetřen v *choose_the_best_candidate*. Množství výjimek vyvolaných knihovnou *libnfporb* by se teoreticky dal snížit aktivací počítání s nekonečnou přesností v této knihovně, ale jednak by to značně zpomalilo výpočty, jednak to vyžaduje knihovnu *gmplib*, kterou se nám nepodařilo nainstalovat na operačním systému Windows 10.

drawing obsahuje třídu *show_in_pyplot*, která má na starosti tvorbu obrázků umístěných polygonů s využitím knihovny *matplotlib*. Obsahuje množství konfigurovatelných parametrů, aby pokryla potřeby vizualizace při ladění, ale i pro generaci obrázku ukazujícího nádobu s uloženými polygony.

6. Benchmarky

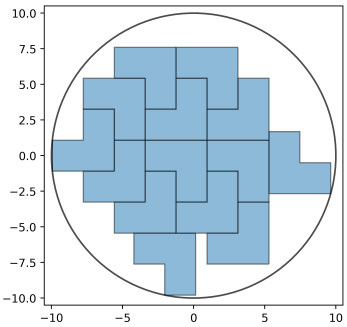
U naimplementovaných algoritmů jsme se rozhodli experimentálně zjistit, které z nich jsou nejlepší, analyzovat jejich rychlost i porovnat s nejlepšími offline algoritmy.

6.1 Výběr nejlepšího implementovaného algoritmu

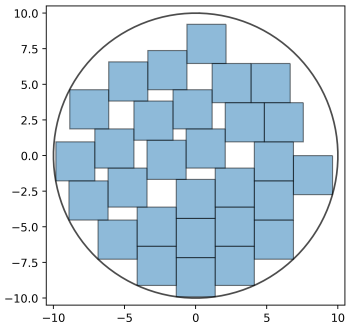
Pro naše benchmarky jsme využili data ze *ShapeGeneratorů SG1* až *SG7*. Některé z nich používají náhodu při generování tvaru, ale tu jsme zafixovali pevně daným seedem.

Spustili jsme všech 7 datasetů na všech existujících kombinacích nádoby (kulatá nádoba o poloměru 10 a čtvercová nádoba o straně 18), picking policy (*My own old*, *Bottom left*, *Minimal convex hull*, *Maximal bounding box overlap*, *Minimal convex hull and minimal surrounding waste plus*, *Minimal surrounding waste* a *Minimal surrounding waste plus*), CFR mode (0 – vrcholy CFR, 1 – jenom konvexní vrcholy regionů CFR, 2 – vrcholy + rovnoměrně body na hranicích CFR), Choose least CFR area – zda předvybrat pouze malé díry v případě jejich existence ($False$, $True$).

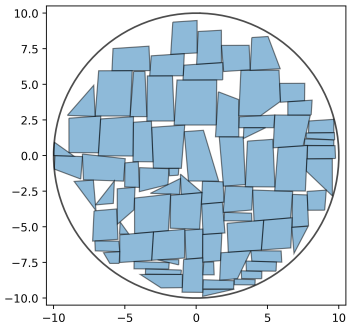
Několik příkladů z finálních rozložení polygonů můžeme vidět na obrázcích 6.1, 6.2, 6.3, 6.4, 6.5 a 6.6.



Obrázek 6.1: *SG1* s parametry *Maximal bounding box overlap*, 0, $True$.

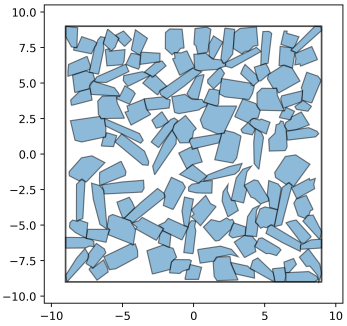


Obrázek 6.2: *SG2* s parametry *Bottom left*, 0, $False$.

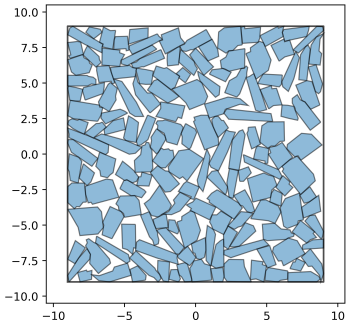


Obrázek 6.3: *SG3* s parametry *Minimal convex hull*, 2, $True$.

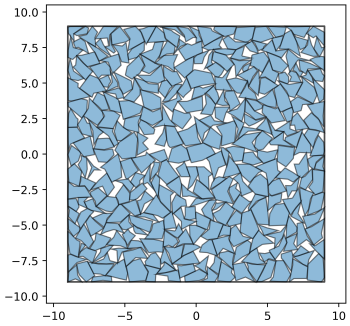
Číselné výsledky všech testů udávající počet položených polygonů jsou na obrázku 6.7. Je z něj vidět, že jednoznačně nejlepší je námi implementovaná picking policy *Minimal surrounding waste plus*, následovaná její základní verzí *Minimal surrounding waste* a její hybribní funkcí *Minimal convex hull and minimal surrounding waste plus*. Naopak nejhůře dopadly *My own old*, „základní“ policy *Bottom left* a *Minimal convex hull*. U té si ty horší výsledky interpretujeme tak, že se snaží za každou cenu mít co polygony co nejkompaktněji poskládané v každém okamžiku běhu algoritmu, i za cenu, že se dá na místo, na které nepadne příliš dobře. Výsledek *My own old* nás trochu překvapil, ale vlastně ho chápeme,



Obrázek 6.4: *SG4* s parametry *My own old*, 0, *False*.



Obrázek 6.5: *SG4* s parametry *My own old*, 1, *True*.



Obrázek 6.6: *SG7* s parametry *Minimal surrounding waste plus*, 0, *False*.

vzhledem k tomu, že primárně na sebe lepí přiléhající tvary bez jakékoliv koncepce – i za cenu, že takovým řetězením vznikne spousta nevyužitelného místa. A v druhé řadě upřednostňuje rozdělení volného prostoru na 2 části, což také není vždy optimální.

Items placed	Square container							Circle container						
Picking policy, CFR mode, Choose least CFR area	SG1	SG2	SG3	SG4	SG5	SG6	SG7	SG1	SG2	SG3	SG4	SG5	SG6	SG7
My own old, 0, False	21	36	79	121	236	193	223	14	31	74	120	224	192	220
My own old, 0, True	21	36	79	143	250	211	262	15	31	78	138	239	207	252
My own old, 1, False	21	36	80	145	244	211	258	15	29	75	138	232	203	246
My own old, 1, True	20	36	84	152	253	215	275	15	29	79	147	240	209	259
My own old, 2, False	21	36	75	121	236	193	223	14	31	61	120	224	186	220
My own old, 2, True	21	36	82	143	250	213	262	14	31	75	138	239	207	252
Bottom left, 0, False	16	36	80	152	240	215	271	14	29	80	147	244	209	262
Bottom left, 0, True	16	36	81	152	250	215	274	14	29	79	147	244	207	264
Bottom left, 1, False	16	36	80	152	240	215	271	14	29	80	147	244	209	262
Bottom left, 1, True	16	36	81	152	250	215	274	14	29	79	147	244	207	264
Bottom left, 2, False	16	36	80	152	240	215	271	14	29	80	147	244	209	262
Bottom left, 2, True	16	36	81	152	250	215	274	14	29	79	147	244	207	264
Minimal convex hull, 0, False	17	36	81	148	246	207	269	15	31	77	147	240	193	255
Minimal convex hull, 0, True	17	36	82	152	250	215	273	15	31	78	147	240	207	264
Minimal convex hull, 1, False	17	36	81	153	246	212	274	15	29	76	147	242	207	258
Minimal convex hull, 1, True	17	36	82	153	253	216	280	16	29	79	148	246	209	262
Minimal convex hull, 2, False	17	35	81	148	235	207	262	13	29	76	143	236	206	247
Minimal convex hull, 2, True	17	35	79	148	244	213	271	13	29	76	147	240	207	258
Maximal bounding box overlap, 0, False	16	36	80	152	249	207	278	13	31	76	147	240	193	262
Maximal bounding box overlap, 0, True	16	36	79	152	253	215	280	14	31	79	148	246	207	271
Maximal bounding box overlap, 1, False	18	36	82	152	246	215	280	14	29	76	152	246	207	274
Maximal bounding box overlap, 1, True	17	36	85	155	256	217	283	14	29	79	152	249	211	278
Maximal bounding box overlap, 2, False	16	36	75	147	244	207	280	13	31	71	147	242	193	274
Maximal bounding box overlap, 2, True	16	36	80	154	256	215	281	14	30	76	148	246	207	268
Min. conv. hull and min. surr. waste +, 0, False	19	36	85	155	256	215	280	15	31	79	149	249	215	271
Min. conv. hull and min. surr. waste +, 0, True	19	36	85	155	259	223	281	15	31	79	151	249	213	275
Min. conv. hull and min. surr. waste +, 1, False	21	36	81	156	253	215	280	15	29	79	152	246	207	262
Min. conv. hull and min. surr. waste +, 1, True	21	36	81	161	257	221	280	15	29	79	148	253	212	278
Min. conv. hull and min. surr. waste +, 2, False	19	36	85	155	257	217	278	15	31	80	152	244	213	274
Min. conv. hull and min. surr. waste +, 2, True	19	36	85	161	259	220	280	15	31	79	152	252	212	272
Minimal surrounding waste, 0, False	20	36	84	168	272	227	291	13	31	79	162	268	215	289
Minimal surrounding waste, 0, True	20	36	87	170	279	231	299	14	31	80	163	268	216	290
Minimal surrounding waste, 1, False	20	36	85	168	273	226	301	15	30	80	165	269	215	291
Minimal surrounding waste, 1, True	21	36	85	169	275	227	300	15	30	79	164	270	218	291
Minimal surrounding waste, 2, False	20	36	80	168	275	227	300	13	31	79	152	264	215	287
Minimal surrounding waste, 2, True	20	36	85	169	278	229	299	14	31	79	162	268	215	291
Minimal surrounding waste plus, 0, False	21	36	86	174	286	231	309	16	31	83	168	278	220	299
Minimal surrounding waste plus, 0, True	21	36	87	174	282	232	304	16	31	85	170	271	219	298
Minimal surrounding waste plus, 1, False	21	36	85	174	284	231	306	17	30	83	170	275	218	299
Minimal surrounding waste plus, 1, True	21	36	85	171	282	234	301	17	29	80	168	270	217	295
Minimal surrounding waste plus, 2, False	21	36	86	174	286	231	309	16	31	83	168	278	220	299
Minimal surrounding waste plus, 2, True	21	36	87	174	282	234	304	16	31	83	170	271	218	298

Obrázek 6.7: Výsledky benchmarků implementovaných algoritmů. V každém sloupci jsou údaje obarvené na škále červená (nejhorší) – žlutá – zelená (nejlepší).

Také můžeme pozorovat, že přednostní vyplňování malých děr (Choose least CFR area = *True*) obecně pomáhá až *Minimal surrounding waste plus*, kde je to nejednoznačné – v některých případech to trochu pomohlo, v některých trochu uškodilo.

Co se týče CFR mode, tak oproti vybírání pouze vrcholů (0) nebylo vybírání bodů z hran navíc (2) obecně lepší – v některých případech to mírně pomohlo, v některých mírně uškodilo. A navíc to bylo mnohem pomalejší.
Zato vybírání pouze konvexních vrcholů (1) obecně pomáhalo u horších algoritmů (kromě *Bottom left*, kde to na výsledek nemělo vliv), ale u těch lepších byl tento vliv nejednoznačný.

Nakonec jsme jako nejlepší námi navrhovaný algoritmus vybrali použití čistého *Minimal surrounding waste plus*, tj. s CFR mode = 0 a Choose least CFR area = *False*, který dále budeme porovnávat s ostatními algoritmy.

Také jsme pro lepší ilustraci vytvořili videa toho, jak postupně naplňují kruhovou nádobu nejlepší *Minimal surrounding waste plus* a pro srovnání čistá „základní“ policy *Bottom left*.

6.2 Časová analýza

V této sekci se budeme zabývat rychlostí běhu algoritmu.

Times taken	Square container							Circle container						
Picking policy, CFR mode, Choose least CFR area	SG1	SG2	SG3	SG4	SG5	SG6	SG7	SG1	SG2	SG3	SG4	SG5	SG6	SG7
My own old, 0, False	2.80	1.05	7.36	54.16	205.62	186.93	233.84	5.51	2.30	13.96	69.08	229.04	244.44	293.25
My own old, 0, True	2.72	1.04	6.01	42.51	147.95	115.56	173.35	5.97	2.22	11.34	59.45	185.83	140.25	198.97
My own old, 1, False	2.68	1.04	5.57	38.66	123.98	96.78	136.77	5.93	2.11	11.15	54.45	157.45	127.22	162.06
My own old, 1, True	2.50	1.05	5.77	41.07	128.20	90.72	146.88	5.90	2.12	10.59	56.43	156.50	116.87	167.58
My own old, 2, False	3.35	1.20	17.90	186.83	657.45	980.42	894.32	8.18	3.97	41.28	216.67	705.96	1160.08	1111.96
My own old, 2, True	3.40	1.20	10.15	83.33	300.46	278.36	392.70	7.79	3.87	20.48	119.46	388.91	378.06	454.65
Bottom left, 0, False	1.75	1.02	4.97	41.03	114.58	86.21	134.88	6.11	2.02	9.91	55.63	153.80	113.41	161.28
Bottom left, 0, True	1.75	1.03	5.15	39.33	117.58	86.90	141.02	5.50	1.99	10.01	54.17	151.66	108.61	164.70
Bottom left, 1, False	1.74	1.04	5.01	39.73	113.13	87.04	135.64	5.34	2.03	9.87	53.96	152.12	110.79	160.38
Bottom left, 1, True	1.74	1.07	5.06	39.18	117.99	85.83	138.53	5.36	2.02	9.87	53.85	154.44	109.73	165.01
Bottom left, 2, False	1.94	1.08	6.09	44.97	123.38	99.08	153.71	6.18	2.45	13.33	66.78	190.52	139.80	192.91
Bottom left, 2, True	1.91	1.08	6.16	44.77	132.65	99.03	154.22	6.23	2.48	13.11	66.25	188.99	136.14	195.04
Minimal convex hull, 0, False	2.07	1.23	8.17	82.74	295.35	240.59	320.29	5.81	2.18	14.02	98.96	352.87	209.44	360.72
Minimal convex hull, 0, True	2.02	1.13	7.85	67.97	219.61	162.56	271.60	5.82	2.19	12.08	92.26	308.82	177.05	340.68
Minimal convex hull, 1, False	1.93	1.11	6.15	47.29	147.58	105.37	175.73	5.82	2.09	11.27	79.55	225.05	156.39	257.59
Minimal convex hull, 1, True	1.93	1.09	6.12	45.66	141.63	106.34	175.60	6.22	2.06	11.46	69.36	227.45	153.73	240.59
Minimal convex hull, 2, False	2.23	1.18	14.82	194.74	1031.12	690.22	789.79	6.67	3.27	37.24	308.11	1353.30	832.33	1210.96
Minimal convex hull, 2, True	2.25	1.16	12.30	120.32	395.14	282.15	564.74	6.34	3.01	22.11	257.62	783.63	627.54	972.27
Maximal bounding box overlap, 0, False	1.92	1.19	7.13	48.09	135.68	105.62	176.31	4.99	2.26	10.94	64.42	192.41	130.50	195.98
Maximal bounding box overlap, 0, True	1.88	1.12	5.73	43.11	131.17	95.26	158.58	5.35	2.21	10.84	61.15	171.76	119.63	186.68
Maximal bounding box overlap, 1, False	2.30	1.18	5.83	41.23	116.34	89.92	147.85	5.37	2.08	10.12	60.38	165.19	114.80	181.75
Maximal bounding box overlap, 1, True	2.16	1.15	6.54	43.41	126.36	88.75	147.00	5.34	2.07	10.50	60.55	163.39	117.49	185.49
Maximal bounding box overlap, 2, False	1.99	1.21	13.70	70.34	281.34	214.64	278.94	6.66	3.35	21.90	121.66	357.25	380.70	379.76
Maximal bounding box overlap, 2, True	1.97	1.23	9.62	63.21	170.25	125.30	209.37	7.00	3.13	18.44	95.81	284.85	209.15	300.57
Min. conv. hull and min. surr. waste +, 0, False	2.84	1.18	8.83	78.51	288.95	194.60	344.88	5.82	2.22	14.68	108.34	336.12	247.10	425.02
Min. conv. hull and min. surr. waste +, 0, True	2.34	1.16	7.97	70.45	238.67	175.14	281.53	5.74	2.18	12.01	92.11	291.46	194.02	338.85
Min. conv. hull and min. surr. waste +, 1, False	2.67	1.09	6.15	48.02	152.57	108.54	184.11	5.73	2.04	11.88	80.39	238.70	146.43	273.07
Min. conv. hull and min. surr. waste +, 1, True	2.70	1.08	6.31	51.33	145.51	105.72	170.21	5.71	2.04	11.11	69.06	241.26	160.09	258.76
Min. conv. hull and min. surr. waste +, 2, False	2.58	1.18	15.64	188.59	874.03	439.73	1007.19	7.62	3.20	38.47	323.00	1287.24	829.85	1348.71
Min. conv. hull and min. surr. waste +, 2, True	2.57	1.18	12.65	133.74	493.14	362.91	721.63	7.39	3.13	30.28	251.74	1171.42	594.89	1072.40
Minimal surrounding waste, 0, False	2.82	1.12	7.66	75.23	229.18	173.83	254.51	5.25	2.33	12.23	87.89	269.18	189.51	302.89
Minimal surrounding waste, 0, True	2.70	1.12	7.51	68.77	201.28	142.76	223.77	5.57	2.26	11.33	78.94	224.12	162.69	251.48
Minimal surrounding waste, 1, False	2.64	1.08	6.40	52.81	153.93	102.92	182.57	6.27	2.21	10.97	68.21	190.48	124.90	205.74
Minimal surrounding waste, 1, True	2.79	1.07	6.06	51.60	149.66	100.77	173.58	6.00	2.16	10.39	66.98	186.89	125.70	202.33
Minimal surrounding waste, 2, False	3.51	1.43	21.17	180.06	612.42	688.92	723.41	7.59	3.85	28.45	200.48	717.91	651.60	777.94
Minimal surrounding waste, 2, True	3.48	1.48	15.13	148.13	415.43	341.74	459.58	7.76	3.58	22.74	165.68	495.06	414.25	566.67
Minimal surrounding waste plus, 0, False	3.12	1.14	8.83	75.85	237.15	168.13	263.65	6.98	2.57	17.09	94.13	280.64	194.21	303.38
Minimal surrounding waste plus, 0, True	2.92	1.12	7.84	72.21	208.84	153.61	236.43	6.94	2.50	13.16	90.93	246.88	173.78	273.35
Minimal surrounding waste plus, 1, False	2.78	1.10	6.25	54.93	162.08	105.23	183.69	8.17	2.34	12.30	72.36	201.96	129.85	217.13
Minimal surrounding waste plus, 1, True	3.17	1.18	6.24	53.14	161.43	107.04	173.06	8.12	2.27	11.45	70.61	191.54	125.76	207.55
Minimal surrounding waste plus, 2, False	4.31	1.61	20.29	185.98	625.56	508.92	684.29	11.44	5.12	40.56	225.35	720.09	601.42	793.92
Minimal surrounding waste plus, 2, True	4.23	1.61	18.79	174.89	492.41	374.61	548.95	11.26	4.71	26.55	203.56	582.82	474.89	647.37

Obrázek 6.8: Čas běhu v sekundách jednotlivých benchmarků, v každém sloupci jsou hodnoty zvýrazněné na škále červená (nejdelší) – žlutá – zelená (nejkratší).

Nejprve se zaměříme na čas běhu jednotlivých benchmarků, viz obrázek 6.8. Všechny algoritmy jsou naimplementovány jednovláknově a spouštěly se na procesoru *Ryzen 3700X* s 8 jádry, 16 vlákny a základním taktem 3,6 GHz. Ačkoliv jsme během benchmarků nespouštěli více než 4 různé picking policy najednou, aby se jednotlivá jádra pořád využívala na maximum, tak uvedené časy je potřeba vnímat pouze orientačně – každá hodnota pochází z pouze jednoho měření.

Můžeme z toho nicméně dedukovat, že oproti vybírání všech vrcholů CFR (CFR mode 0) vybírání pouze konvexních vrcholů (CFR mode 1) obecně zrychlovalo dobu běhu, v některých případech až o třetinu času. Naopak vybírání navíc bodů z hran (CFR mode 2) zásadně zpomalovalo běh algoritmu, v některých případech až více než trojnásobně.

Primární vybírání malých děr (Choose least CFR area $True$) v kombinaci s CFR mode 0 obecně trochu zrychlovalo dobu běhu, v kombinaci s CFR mode 1 občas mírně zrychlovalo, občas mírně zpomalovalo, a v kombinaci s CFR mode 2 značně zrychlovalo, v některých případech až trojnásobně.

Tyto informace nám pouze napoví, zda dané techniky zpomalují nebo zrychlují dobu běhu, ale neřeknou, kolikanásobné zrychlení či zpomalení bude při použití násobně větší nebo menší nádoby.

6.3 Časová složitost

V této sekci budeme analyzovat časovou složitost vybraného nejlepšího algoritmu v závislosti na počtu položených tvarů n. Použijeme několik odhadů časových složitostí knihovních funkcí a provedeme několik menších a jedno zásadního zjednodušení. Tím je předpoklad, že všechny polygony na vstupu, včetně samotné nádoby, mají počet vrcholů shora omezený nějakou konstantou.

Nejprve si zanalyzujeme výpočet kandidátů pomocí CFR.
Neznáme časovou složitost výpočtu NFP ani IFP, ale díky tomu, že jak pokládaný tvar, tak nádoba mají konstantou omezený počet vrcholů, tak je délka výpočtu jednoho NFP i IFP omezena konstantou, čili běží v $\mathcal{O}(1)$.
Pokud budeme uvažovat okrajové případy, tak se původně extrémně málo často se stávalo, že se volání knihovního výpočtu NFP zacyklilo ve *while* smyčce. To jsme ošetřili tím, že jsme nastavili maximální počet opakování této smyčky, a v případě překročení tohoto maxima funkce vrátí, co vypočetla, i když to nemusí být úplně správné. Občasné výjimky vzniklé při knihovním výpočtu NFP, jsme ošetřili tak, že se funkce výpočtu NFP zavolá nejvýše 20x, přičemž se po každém neúspěchu posune celý ukládaný tvar a pak se mu náhodně posunou jednotlivé vrcholy o číslo v řádu 10^{-10}. Platí tedy, že i při započtení těchto okrajových případů máme konstantní čas výpočtu IFP i NFP.
Jedno volání CFR udělá $\mathcal{O}(1)$ výpočtů IFP a nejvýše $\mathcal{O}(n)$ výpočtů NFP. Potom se spočítá *unary_union* všech NFP. Tato *Shapely* funkce volá příslušnou funkci *C++* knihovny *GEOS*. Algoritmus výpočtu funguje na iniciálním rozdělení blízkých oblastí (aby se během sjednocení odstranilo co nejvíce vrcholů) pomocí R-stromu a algoritmu Sort-Tile-Recursive (viz dokumentace). Potom se množiny polygonů rekurzivně rozdělují na zhruba stejné poloviny až do momentu, kdy

zůstanou pouze 2 polygony, které se sjednotí pomocí normálního geometrického sjednocení, ty se sjednotí s jiným sjednocením 2 polygonů atd. Jaká je ale složitost běžného sjednocení? Margalit a Knott (1989) ukazují algoritmus s časovou složitostí $\mathcal{O}((a+b+k)log(a+b+k))$, kde a je počet vrcholů, resp. hran prvního polygonu, b počet vrcholů druhého polygonu a k počet průsečíků, který obecně může být až $\mathcal{O}(a \cdot b)$. Použití algoritmu s touto časovou složitostí napovídá práce Lin a Li (2009), kteří přímo zmiňují informační systém GIS, který nejspíše využívá knihovnu *GEOS*.
Množství práce ve vrcholu rekurze je tedy $\mathcal{O}((n+k) \cdot \log(n+k))$, kde k je až $\mathcal{O}(n^2)$. Domníváme se, že vzhledem k používání NFP generovaných z tvarů, které se nepřekrývají, by se dal počet průsečíků v každé vrstvě rekurze omezit na $\mathcal{O}(n)$, díky čemuž by časová složitost *unary_union* byla $\mathcal{O}(n \log^2(n))$, ale neumíme to dokázat. Proto omezíme k pomocí $\mathcal{O}(n^2)$, což nám dá složitost *unary_union* $\mathcal{O}(n^2 \log(n^2)) = \mathcal{O}(n^2 \log(n))$, která nám ale v konečném důsledku naštěstí nezhorší časovou složitost programu.
Poté se provede rozdíl IFP (IFP má $\mathcal{O}(1)$ vrcholů – nádoba i pokládaný tvar mají $\mathcal{O}(1)$ vrcholů) a sjednocení NFP, který zabere $\mathcal{O}(n \log(n))$ času. Nakonec se ještě počítá plocha regionů CFR, ale to zabere $\mathcal{O}(n)$ času.

Výpočet CFR se při pokládání jednoho tvaru provede tolikrát, kolik je povolených rotací, což je konstanta. Potom se zavolá funkce výpočtu nejlepšího kandidáta a potom se z dražších operací pouze provede rozdíl *MultiPolygonu* reprezentující volné místo a pokládaného tvaru, což je v $\mathcal{O}(n \log(n))$. Funkce výpočtu nejlepšího kandidáta zavolá kód zvolené picking policy a provede 1 až 2 rozdíly, oba v $\mathcal{O}(n \log(n))$. V okrajových případech se tato funkce zavolá sama na sebe se seznamem kandidátů ochuzeného o zvoleného kandidáta, který byl špatný. Pro zjednodušení předpokládejme, že se nám to za celou dobu běhu programu stane nanejvýše $\mathcal{O}(n)$krát.

Výpočet *Minimal surrounding waste plus* provede pro každého kandidáta, kterých je $\mathcal{O}(n)$, 1 průnik s nádobou a 1 rozdíl s nádobou, oba v $\mathcal{O}(1)$, a 1 rozdíl s volným prostorem v nádobě v $\mathcal{O}(n \log(n))$, takže celkově jeden výpočet nejlepšího kandidáta podle této picking policy je v $\mathcal{O}(n^2 \log(n))$.

Protože se funkce pokládání tvaru se provede celkem n-krát, tak celková časová složitost našeho algoritmu je $\mathcal{O}(n^3 \log(n))$.

Jaká je ale časová složitost pro průměrné případy? To se obtížně analyzuje, ale vyzkoušeli jsme si změřit časy na *SG1*, *SG2*, *SG4*, *SG5*, *SG6* a *SG7* (*SG3* není stavěn pro větší nádoby) s nejlepší picking policy a čtvercovou nádobou o různých velikostech. Poté jsme vypočetli tabulku časů dělených danou funkcí závislou na n, abychom porovnali, zda se při rostoucí velikosti nádoby zmenšuje tento poměr (funkce roste rychleji než čas) nebo naopak zvětšuje (funkce roste pomaleji než čas), jak ukazuje obrázek 6.9.
Jak je vidět z obrázku, čas pro uložení n polygonů s rostoucím n roste rozhodně rychleji než funkce n^2 (až na *SG2*, kde jsou vstupní polygony stejně velké čtverce) a možná podobně jako $n^2 \log(n)$, ale jednak neznáme časy pro ještě větší n, jednak by tento vztah platil pouze pro vstupy z našich generátorů, nikoliv obecně.

Min. surr. waste, 0, False	Time taken (t)						# of placed items (n)					
Length of side of container	SG1	SG2	SG4	SG5	SG6	SG7	SG1	SG2	SG4	SG5	SG6	SG7
18	3.12	1.14	75.85	237.15	168.13	263.65	21	36	174	286	231	309
22	7.79	2.52	170.00	558.14	385.03	650.07	33	64	257	432	335	462
26	12.98	4.12	340.10	1085.78	642.14	1189.77	38	81	363	586	449	642
30	24.75	5.95	660.26	1967.25	1114.44	2122.19	54	100	489	779	579	857
34	46.48	11.38	1044.55	3203.85	1854.45	3676.54	73	144	625	995	737	1103
38	85.87	15.65	1777.35	5060.65	2977.27	7645.62	92	169	787	1243	940	1390
42	97.30	26.99	2584.63	8072.44	4435.76	9604.98	116	225	954	1523	1117	1725
46	198.14	34.99	3916.08	11392.29	6732.55	14247.12	137	256	1146	1823	1342	2046
50	188.76	56.64	5307.86	17148.21	9445.60	-	156	324	1349	2157	1580	2414

	10000 * t / (n * n)						10000 * t / (n * n * Log2(n))					
18	70.67	8.77	25.05	28.99	31.51	27.61	16.09	1.70	3.37	3.55	4.01	3.34
22	71.50	6.15	25.74	29.91	34.31	30.46	14.17	1.03	3.22	3.42	4.09	3.44
26	89.90	6.28	25.81	31.62	31.85	28.87	17.13	0.99	3.04	3.44	3.62	3.10
30	84.89	5.95	27.61	32.42	33.24	28.90	14.75	0.90	3.09	3.37	3.62	2.97
34	87.22	5.49	26.74	32.36	34.14	30.22	14.09	0.77	2.88	3.25	3.58	2.99
38	101.45	5.48	28.70	32.75	33.69	39.57	15.55	0.74	2.98	3.19	3.41	3.79
42	72.31	5.33	28.40	34.80	35.55	32.28	10.54	0.68	2.87	3.29	3.51	3.00
46	105.57	5.34	29.82	34.28	37.38	34.03	14.87	0.67	2.93	3.16	3.60	3.09
50	77.56	5.40	29.17	36.86	37.84		10.65	0.65	2.81	3.33	3.56	

Obrázek 6.9: Nahoře časy a počty položených polygonů pro dané *ShapeGeneratory* a různé velikosti čtvercové nádoby, dole porovnání růstu času s funkcemi n^2 a $n^2 \log(n)$ na škále žlutá (nejmenší hodnota) – zelená (největší hodnota).

6.4 Výběr nejlepšího online algoritmu

V této sekci jsme chtěli zjistit, který online algoritmus je nejlepší vůbec, a tím pádem porovnat ten náš s ostatními dostupnými implementacemi online 2D irregular BPP, ale všechny implementace 2D irregular BPP, na které jsme narazili a které se nám podařilo zprovoznit, používají offline algoritmus – některé používají konstruktivní přístup, který je sám o sobě online, ale na začátku si setřídí vstupní polygony od největšího po nejmenší, takže je budeme porovnávat až v rámci porovnání s offline algoritmy.

Jedinou výjimkou jsou algoritmy od kolegů z doby Studentského fakultního grantu, které jsou online.
První algoritmus, Filipův, je ve stylu tetrisu. Nejdříve si najde rotaci s minimální šířkou, potom spouští tvar seshora dolů, dokud nedojde ke kolizi, a případně ho pak ještě posune doleva. Poté si posune doprava, případně zresetuje na začátek, horizontální pozici, ze které spouští pokládaný tvar, a jde na nový tvar. Po zaplnění nádoby tímto způsobem se ještě aplikovat stejný postup, akorát tvary nespouští seshora, ale ze strany.
Druhý, úspěšnější algoritmus, Danův, počítá Minkowského sumu jako NFP konvexních obalů umisťovaného tvaru a již uložených polygonů a vybere nejlepšího kandidáta metodou *Bottom left*. Porovnáme-li procentuální zaplnění nádoby ($\frac{plocha\ umístěných\ polygonů}{plocha\ nádoby}$), tak na *SG1* až *SG7* s kruhovou nádobou byl zhruba o $4-5\%$ horší než naše implementace této metody, ale v přepočtu na jeden umístěný polygon 12x rychlejší.

Můžeme tedy konstatovat, že námi vytvořený algoritmus s picking policy *Minimal surrounding waste plus* je podle našich znalostí nejlepší online algoritmus řešící 2D irregular BPP.

6.5 Porovnání s offline algoritmy

V táto sekci porovnáme náš algoritmus s nejlepšími offline algoritmy řešícími 2D irregular BPP.

Hledali jsme jak komerční software, tak hotové programy či volně dostupné repositáře s fungujícím kódem.

Ačkoliv jsme narazili na asi 15 open source implementací 2D irregular BPP, tak samotné jejich zprovoznění byl velmi náročný úkol. U některých kompletně chyběla dokumentace nebo alespoň návodné komentáře v kódu, některé vyžadovaly spoustu závislostí, které se nám nepodařilo nainstalovat ani po několika hodinách snahy, některé nešly vůbec zprovoznit. Když od toho ještě odečteme implementace, které neměly moc dobré výsledky, implementace, které řešily trochu jiný problém (například strip packing, kde je pevně daná šířka nádoby, ale nelze nastavit délku, a proto se umístí úplně všechny položky ze vstupu) a horší verze nějaké jiné implementace (například SVGNest (Qiao a kol., 2019), který má mnohem rychlejší implementaci v podobě Deepnestu), tak nám zbývají 4 implementace: Dalsoo-Bin-Packing (Hua, 2020), Packaide, součást Fabricaide (Sethapakdi a kol., 2021), Deepnest (Qiao a bmtm, 2018) a implementace diplomové práce Ondřeje Švandy (Švanda, 2020).

Z komerčního software se nám podařilo získat přístup do NestFabu (zde patří poděkování Tomu Buddinovi z NestFabu za opakované prodloužení zkušební licence) a PowerNestu, kde je zdarma dostupné demo vyhovující našim potřebám. Oba programy o sobě tvrdí, že jsou nejlepší na světě.
Pro úplnost zde uvádíme komerční programy, u kterých se nám nepodařilo získat ani zkušební licenci: Autonester-T, Almacam Cut a SigmaNEST.

Pro srovnávání s offline algoritmy jsme vždy používali obdélníkové nádoby, protože většina offline algoritmů nepodporuje jiné tvary nádob.
Vytvořili jsme si datasety z *SG1* až *SG7* a to tak, že jsme si nechali vygenerovat o několik polygonů více než umístil námi navrhovaný algoritmus, aby offline algoritmy nezískaly výhodu výběru z mnohem většího množství kandidátů, než kolik vůbec lze umístit.
Rozhodli jsme se také použít několik obecně známých ESICUP datasetů, konkrétně datasety *albano*, *marqes* a *swim* pocházející z textilního průmyslu, *dighe1* a *dighe2* tvořené z dílků, které do sebe přesně zapadají a *shapes* tvořený z několika navrhnutých tvarů. U některých datasetů nebyla dostupná informace o výšce nádoby – tu jsme ad hoc vybrali, aby náš algoritmus umístil skoro všechny tvary z datasetu. U některých chyběl počet povolených rotací, ten jsme arbitrárně vybrali z hodnot 1 (žádné rotace), 2 (rotace o 180°), 4 (rotace o 90°). U *dighe1* a *dighe2* jsme mírně zvětšili rozměry nádob (ze stran délky 100 a délky 100,01), aby algoritmy měly šanci umístit všechny tvary i navzdory případným problémům, které s sebou přináší s aritmetika s floating point čísly.

Na zmíněných datasetech jsme otestovali zmíněné offline algoritmy, náš navrhovaný algoritmus, ale i náš navrhovaný algoritmus s předem setříděným vstupem polygonů od největší plochy po nejmenší, protože nás zajímalo, jak moc velké znevýhodnění je dostat náhodně vstup v náhodném pořadí oproti setříděnému

vstupu, který se obecně považuje za lepší.

Jednotlivé datasety měly následující počty polygonů:

albano:	24
dighe1:	16
dighe2:	10
marques:	24
shapes:	43
swim:	48
SG1:	21
SG2:	36
SG3:	93
SG4:	180
SG5:	300
SG6:	250
SG7:	330

Ještě zde uvedeme případná specifika jednotlivých programů.
Dalsoo-Bin-Packing byl spouštěn s parametrem *segment_max_length* 40.
Packaide se nám podařilo zprovoznit pouze v Ubuntu virtual machine a byl spouštěn s parametem *Discretization tolerance* 0.02.
Waste Optimizer Ondřeje Švandy bylo potřeba mírně modifikovat, aby fungoval i pro případy, kdy je více tvarů, než dokáže umístit, a byl spouštěn s parametry *small holes first True*, *use NFP True*, *Local optimization False*.
Deepnest dokázal využít více jader (i když ne na plno během celé doby běhu) a inkrementálně nacházel lepší a lepší řešení, proto jsme ho u každého datasetu spustili na 13 minut a akceptovali nejlepší výsledek po této době.
PowerNest, respektive jeho demo, mělo maximální dobu běhu 300s pro jednotlivé úlohy a počítalo je v cloudu. Nepodporuje možnosti 3 ani 6 povolených rotací, proto se spouštělo u *SG4* se 2 povolenými rotacemi (normálně jsou 3) a u *SG5* se 4 rotacemi (normálně je 6).
NestFab dokázal naplno využít téměř celého výkonu procesoru a inkrementálně nacházel lepší a lepší řešení, proto jsme ho u každého datasetu spustili na 10 minut a akceptovali nejlepší řešení po této době. Také nepodporuje možnosti 3 ani 6 povolených rotací, proto se u *SG4* spouštěl s 2 rotacemi a u *SG5* s 4 rotacemi.

Výsledky zobrazující poměr zaplněného prostoru jsou na obrázku 6.10.

ratio of filled area	albano	dighe1	dighe2	marques	shapes	swim
Dalsoo	0.8072	0.6062	0.6002	0.8058	0.5818	0.6348
Packaide	0.8131	0.6047	0.6662	0.7925	0.5491	0.6216
Švanda Ondřej	**-	0.7048	0.6651	0.8276	0.5709	0.6713
DeepNest	0.8506	0.9102	0.7878	0.8551	0.6473	0.6744
Min. surr. waste plus	0.7556	0.8602	0.3758	0.7469	0.6618	0.6410
Sorted min. surr. waste plus	0.8395	0.5401	0.7195	0.7382	0.6036	0.6511
PowerNest	*0.8705	0.8925	0.9361	*0.8647	0.7064	*0.7373
NestFab	*0.8705	*0.9998	*0.9998	*0.8647	*0.7164	*0.7373

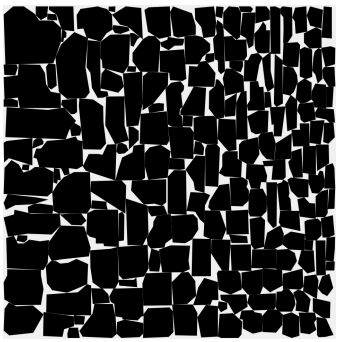
	SG1	SG2	SG3	SG4	SG5	SG6	SG7
Dalsoo	0.8320	*0.8386	0.8496	0.6954	0.7506	**-	0.7072
Packaide	0.7882	*0.8386	0.8719	0.7387	0.7535	0.8456	0.7099
Švanda Ondřej	0.7444	*0.8386	0.8746	0.7517	0.7765	**-	0.7481
DeepNest	0.8759	*0.8386	0.8907	0.7321	0.7519	0.8365	0.7140
Min. surr. waste plus	*0.9196	*0.8386	0.8401	0.8093	0.8384	0.8593	0.7926
Sorted min. surr. waste plus	*0.9196	*0.8386	0.8981	0.8188	0.8420	0.8586	0.7995
PowerNest	*0.9196	*0.8386	0.9425	0.8367	0.8710	0.8983	0.8455
NestFab	*0.9196	*0.8386	0.9461	0.8428	*0.8760	0.9071	*0.8554

*all items from dataset were placed
**program crashed or stopped working

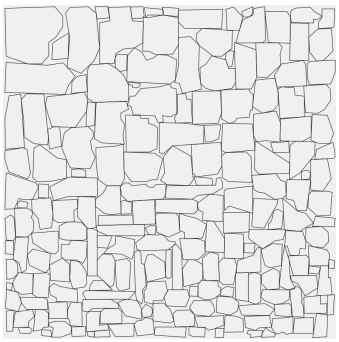
Obrázek 6.10: Poměry zaplněného prostoru pro jednotlivé algoritmy a datasety, s hodnotami zabarvenými škále žlutá (nejhorší) – zelená (nejlepší).
*algoritmu se podařilo umístit všechny předměty.
**program během počítání na daném vstupu spadnul.

Ukázky finálních umístění můžeme vidět na obrázcích 6.11, 6.12, 6.13, 6.14,

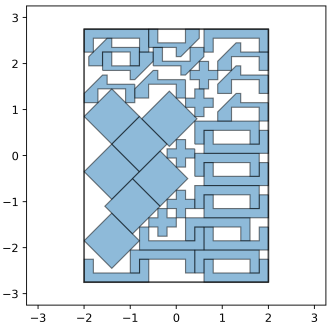
6.15, 6.16, 6.17, 6.18 a 6.19.



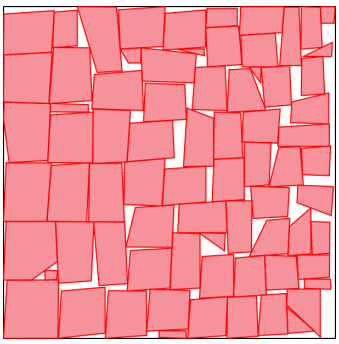
Obrázek 6.11: Packaide na *SG6* datasetu.



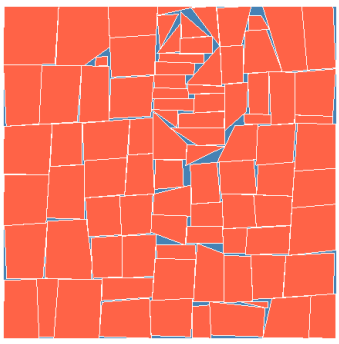
Obrázek 6.12: Deepnest na *SG6* datasetu.



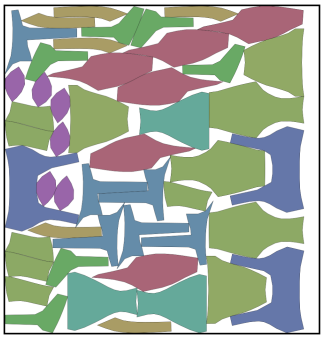
Obrázek 6.13: *Minimal surrounding waste plus* na *shapes* datasetu.



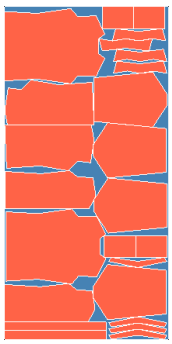
Obrázek 6.14: Waste Optimizer na *SG3* datasetu.



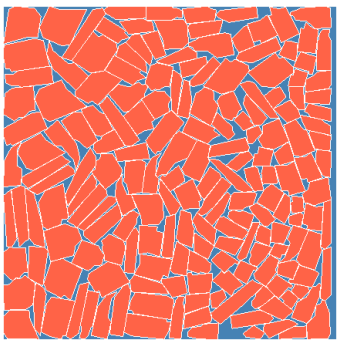
Obrázek 6.15: NestFab na *SG3* datasetu.



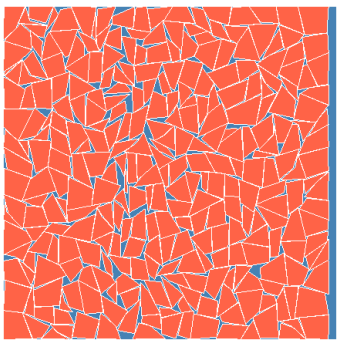
Obrázek 6.16: PowerNest na *swim* datasetu.



Obrázek 6.17: NestFab na *albano* datasetu.



Obrázek 6.18: NestFab na *SG4* datasetu.



Obrázek 6.19: NestFab na *SG5* datasetu.

Výsledky jsou zajímavé z několika hledisek. Jednak to ukazuje jasnou převahu komerčních aplikací, které dokážou najít skutečně velmi dobré řešení; z konkrétních obrázků rozložení u NestFabu se zdá, že dokáže najít řešení, které je velmi blízko skutečně optimálnímu řešení.
Překvapily nás také výsledky našeho vlastního algoritmu v porovnání s open source offline algoritmy. Na *ESICUP* datasetech dosáhl porovnatelných výsledků

a na na *SG* datasetech s více než stovkou umisťovaných tvarů byl dokonce suverénně nejlepší. Domníváme se, že u datasetů s nejvýše malými desítkami předmětů se můžou velmi dobře projevit účinky offline optimalizací, ale u větších datasetů je důležité mít kvalitní picking policy při sestavování počátečního konstruktivního řešení či nějaké části řešení.
Můžeme také pozorovat, že spuštění našeho algoritmu na předem setříděných datasetech většinou přineslo lepší výsledek. Výsledek se naopak zhoršil u některých malých datasetů, kde vstupní pořadí více hraje roli. Nevýhodou bylo navíc to, že náš algoritmus při tvaru, který nedokáže umístit, skončí, namísto aby dané tvary nepokládal, ale čekal na další, menší tvary, u kterých je větší šance, že se je ještě podaří umístit.

6.6 Efektivita algoritmu na větších nádobách

Nakonec nás zajímalo, jak se změní efektivita našeho algoritmu, pokud zvětšíme velikost nádoby, a tím pádem i počet umístěných tvarů. Po položení posledního umístitelného tvaru přirozeně v nějaké oblasti zůstával volný prostor. Předpokládali jsme, že s rostoucí velikostí nádoby se poměr takového nevyužitého místa bude zmenšovat. Chtěli jsme tímto způsobem odhadnout, jaká by byla efektivita na daných datasetech, pokud by se jejich velikost blížila nekonečnu.

Spustili jsme tedy náš algoritmus na *SG1*, *SG2*, *SG4*, *SG4*, *SG5* a *SG6* a různých velikostech nádob. Výsledky můžeme vidět na obrázku 6.20

ratio of filled area, picking policy Minimal surrounding waste plus, 0, False

Size of square bin	SG1	SG2	SG4	SG5	SG6	SG7
10x10	0.7094	0.6792	0.7731	0.7867	0.7904	0.7628
14x14	0.8686	0.9626	0.7919	0.8208	0.8552	0.7765
18x18	0.9196	0.8386	0.8093	0.8384	0.8593	0.7926
22x22	0.9674	0.9980	0.8144	0.8489	0.8572	0.8078
26x26	0.7975	0.9043	0.8255	0.8495	0.8560	0.8185
30x30	0.8513	0.8386	0.8331	0.8539	0.8579	0.8224
34x34	0.8959	0.9401	0.8383	0.8606	0.8649	0.8271
38x38	0.9039	0.8833	0.8398	0.8635	0.8737	0.8299
42x42	0.9330	0.9626	0.8448	0.8689	0.8716	0.8342
46x46	0.9186	0.9131	0.8474	0.8704	0.8745	0.8386
50x50	0.8853	0.9781	0.8511	0.8734	0.8775	0.8381
60x60	0.9143	0.9245				
70x70	0.9150	0.9626				
80x80	0.8801	0.9917				

Obrázek 6.20: Poměry zaplněného prostoru pro náš algoritmus s jednotlivými datasety a velikostmi nádob, s hodnotami zabarvenými na škále červená (nejhorší) – žlutá – zelená (nejlepší).

U *SG4* až *SG7* můžeme vidět, že s rostoucí velikostí nádoby stoupá efektivita, a že efektivita u nádoby 50x50 vzrostly zhruba o $2\% - 4\%$ oproti původní velikosti nádoby 18x18. Můžeme pozorovat, že u různých datasetů je maximální efektivita různá.
Naopak u *SG2*, datasetu se stejně velkými čtverci, skáče efektivita téměř náhodně. Ve všech případech algoritmus našel optimální řešení, ale u různých velikostí nádoby zbýval různý poměr volného místa. U *SG1* datasetu se stejnými „L“ tvary

efektivita také dost skáče, což je zapříčiněné jak množstvím volného místa u optimálního umístění, ale také tím, že náš algoritmus nenacházel vždy optimální výsledky. Je potřeba zmínit, že u datasetů jako *SG1* nebo *shapes* měl náš algoritmus časté problémy s generací kandidátů z CFR, respketive s výpočtem NFP, který často neuspěl.

Repositář se zdrojovým kódem, datasety i všemi výsledky (jak číselnými, tak obrázky vygenerovaných uložení) se nachází jak v příloze A.1, tak ve formě Github repositáře (Wałoszek, 2021).

Závěr

V této práci jsme rozsáhle popsali 2D irregular bin packing problem a pro jeho online variantu variantu také vyvinuli a implementovali vlastní algoritmus. Námi vyvinutý algoritmus je dokonce podle našich znalostí nejlepší online algoritmus řešící tento problém.

V praxi budeme chtít na destičku uložit nízké stovky krystalků tvarů podobných těm, které generují *SG4* a *SG6*. V těchto případech náš algoritmus zaplní nádobu v poměru 0,8093 a 0.8593 k velikosti nádoby.
Nejlepší offline algoritmus zaplní nádobu v poměru 0.8428 a 0.9071.

Můžeme tedy použít online s velmi dobrým výsledkem, nebo chtít nejlepší efektivitu rozložení, což znamená zaplnění o zhruba 4 až 5,5% více plochy za cenu pořízení licence a 2x více práce během umisťování krystalků.
Pokud bychom chtěli ušetřit za licenci, tak bychom mohli zvolit open source offline algoritmus, ale všechny takové se pro praktické řešení našeho problému ukázaly jako horší než náš online algoritmus.
Dalším problémem u nejlepšího offline algoritmu je, že neobsahuje možnost umisťovat předměty do jiných než obdélníkových nádob, čímž se pro nás stává nepoužitelným.

Další možností by bylo vyfotit všechny krystalky kamerou s nižším rozložením, přibližně si je setřídit podle velikosti a poté použít náš online algoritmus. Získali bychom díky tomu zlepšení o 0 až 1%.

Existuje několik způsobů, kterými by šlo navázat na naši práci.
První věcí je samotná integrace algoritmu v rámci ALSA zařízení, kde po vyfocení kamerou se pomocí knihovny OpenCV zjistí přesný tvar krystalku a pošle našemu algoritmu, aby našel nejlepší umístění.
Dále by bylo možné použít picking policy *Minimal surrounding waste plus* jako součást offline algoritmu, například při konstrukci počátečního řešení. V tomto případě by bylo vhodné algoritmus urychlit, ať už přepsáním do kompilovaného jazyka nebo použitím chytré datové struktury, která by urychlila počítání rozdílu pokládaného tvaru s volným místem v nádobě.
Nakonec zde vidíme malý potenciál ve vylepšení parametrů *Minimal surrounding waste plus* pomocí experimentování na větším množství datasetů a vyzkoušením větší škály hodnot a kombinací parametrů.

Seznam použité literatury

Abeysooriya, R. P., Bennell, J. A. a Martinez-Sykora, A. (2018). Jostle heuristics for the 2d-irregular shapes bin packing problems with free rotation. *International Journal of Production Economics*, **195**, 12–26. ISSN 0925-5273. doi: 10.1016/j.ijpe.2017.09.014. URL https://www.sciencedirect.com/science/article/pii/S0925527317302980.

Balogh, J., Békési, J., Dósa, G., Epstein, L. a Levin, A. (2017). A new and improved algorithm for online bin packing. *CoRR*, **abs/1707.01728**. URL http://arxiv.org/abs/1707.01728.

Balogh, J., Békési, J., Dósa, G., Epstein, L. a Levin, A. (2018). A new lower bound for classic online bin packing. *CoRR*, **abs/1807.05554**. URL http://arxiv.org/abs/1807.05554.

Balogh, J., Békési, J., Dósa, G., Sgall, J. a van Stee, R. (2019). The optimal absolute ratio for online bin packing. *Journal of Computer and System Sciences*, **102**, 1–17. ISSN 0022-0000. doi: 10.1016/j.jcss.2018.11.005. URL https://www.sciencedirect.com/science/article/pii/S0022000019300029.

Bansal, N. a Khan, A. (2014). Improved approximation algorithm for two-dimensional bin packing. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on discrete algorithms*, pages 13–25. SIAM. doi: 10.1137/1.9781611973402.2. URL https://epubs.siam.org/doi/10.1137/1.9781611973402.2.

Bansal, N., Caprara, A. a Sviridenko, M. (2009). A new approximation method for set covering problems, with applications to multidimensional bin packing. *SIAM J. Comput.*, **39**, 1256–1278. doi: 10.1137/080736831. URL https://epubs.siam.org/doi/10.1137/080736831.

Bennell, J. A. a Song, X. (2008). A comprehensive and robust procedure for obtaining the nofit polygon using minkowski sums. *Computers & Operations Research*, **35**(1), 267–281. ISSN 0305-0548. doi: 10.1016/j.cor.2006.02.026. URL https://www.sciencedirect.com/science/article/pii/S0305054806000669. Part Special Issue: Applications of OR in Finance.

Bennell, J. A., Dowsland, K. A. a Dowsland, W. B. (2001). The irregular cutting-stock problem — a new procedure for deriving the no-fit polygon. *Computers & Operations Research*, **28**(3), 271–287. ISSN 0305-0548. doi: 10.1016/S0305-0548(00)00021-6. URL https://www.sciencedirect.com/science/article/pii/S0305054800000216.

Błażewicz, J., Hawryluk, P. a Walkowiak, R. (1993). Using a tabu search approach for solving the two-dimensional irregular cutting problem. *Annals of Operations Research*, **41**(4), 313–325. doi: 10.5555/160231.160259. URL https://www.semanticscholar.org/paper/

Using-a-tabu-search-approach-for-solving-the-Blazewicz-Hawryluk/35c23fb57f3a369300b7a06e928fae350207ee83.

Burke, E., Hellier, R., Kendall, G. a Whitwell, G. (2007). Complete and robust no-fit polygon generation for the irregular stock cutting problem. *European Journal of Operational Research*, **179**(1), 27–49. ISSN 0377-2217. doi: 10.1016/j.ejor.2006.03.011. URL https://www.sciencedirect.com/science/article/pii/S0377221706001639.

Čermák, P. (2021). [Successful] Czech JUNIOR STAR GAČR Project application + evaluation. URL https://figshare.com/articles/online_resource/_Successful_Czech_JUNIOR_STAR_GA_R_Project_application_evaluation/14256521/1.

Čermák, P., Schneidewind, A., Liu, B., Koza, M. M., Franz, C., Schönmann, R., Sobolev, O. a Pfleiderer, C. (2019). Magnetoelastic hybrid excitations in CeAuAl3. *Proceedings of the National Academy of Sciences*, **116**(14), 6695–6700. ISSN 0027-8424. doi: 10.1073/pnas.1819664116. URL https://www.pnas.org/content/116/14/6695.

Chlebík, M. a Chlebíková, J. (2006). Inapproximability results for orthogonal rectangle packing problems with rotations. In *Italian Conference on Algorithms and Complexity*, pages 199–210. Springer. doi: 10.1007/11758471_21. URL https://doi.org/10.1007/11758471_21.

Christensen, H. I., Khan, A., Pokutta, S. a Tetali, P. (2017). Approximation and online algorithms for multidimensional bin packing: A survey. *Computer Science Review*, **24**, 63–79. ISSN 1574-0137. doi: 10.1016/j.cosrev.2016.12.001. URL https://www.sciencedirect.com/science/article/pii/S1574013716301356.

Dalalah, D., Khrais, S. a Bataineh, K. (2014). Waste minimization in irregular stock cutting. *Journal of Manufacturing Systems*, **33**(1), 27–40. ISSN 0278-6125. doi: 10.1016/j.jmsy.2013.11.003. URL https://www.sciencedirect.com/science/article/pii/S0278612513001209.

Dósa, G. a Sgall, J. (2013). First Fit bin packing: A tight analysis. In Portier, N. a Wilke, T., editors, *30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*, volume 20 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 538–549, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-939897-50-7. doi: 10.4230/LIPIcs.STACS.2013.538. URL http://drops.dagstuhl.de/opus/volltexte/2013/3963.

Dósa, G. a Sgall, J. (2014). Optimal analysis of best fit bin packing. In Esparza, J., Fraigniaud, P., Husfeldt, T. a Koutsoupias, E., editors, *Automata, Languages, and Programming*, pages 429–441, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. ISBN 978-3-662-43948-7. doi: 10.1007/978-3-662-43948-7_36. URL doi.org/10.1007/978-3-662-43948-7_36.

Epstein, L. (2010). Two-dimensional online bin packing with rotation. *Theoretical Computer Science*, **411**(31), 2899–2911. ISSN 0304-3975. doi: 10.1016/j.tcs.2010.04.021. URL https://www.sciencedirect.com/science/article/pii/S0304397510002161.

Epstein, L. (2019). A lower bound for online rectangle packing. *Journal of Combinatorial Optimization*, **38**(3), 846–866. doi: 10.1007/s10878-019-00423-z. URL https://doi.org/10.1007/s10878-019-00423-z.

Epstein, L. a van Stee, R. (2006). This side up! **2**(2), 228–243. ISSN 1549-6325. doi: 10.1145/1150334.1150339. URL https://doi.org/10.1145/1150334.1150339.

Garey, M. a Johnson, D. (1978). Computers and intractability: A guide to the theory of np-completeness.

Hassan, A. a Hans, M. (2018). libnfporb. URL https://github.com/kallaballa/libnfporb.

Hifi, M. a M'Hallah, R. (2003). A hybrid algorithm for the two-dimensional layout problem: the cases of regular and irregular shapes. *International Transactions in Operational Research*, **10**(3), 195–216. doi: 10.1111/1475-3995.00404. URL https://onlinelibrary.wiley.com/doi/10.1111/1475-3995.00404.

Hoberg, R. a Rothvoss, T. (2015). A logarithmic additive integrality gap for bin packing.

Hua, H. (2020). Dalsoo-bin-packing. URL https://github.com/whitegreen/Dalsoo-Bin-Packing.

Jakobs, S. (1996). On genetic algorithms for the packing of polygons. *European Journal of Operational Research*, **88**(1), 165–181. ISSN 0377-2217. doi: 10.1016/0377-2217(94)00166-9. URL https://www.sciencedirect.com/science/article/pii/0377221794001669.

Jansen, K. a Prädel, L. (2013). New approximability results for two-dimensional bin packing. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, **74**, 919–936. doi: 10.1007/s00453-014-9943-z. URL https://doi.org/10.1007/s00453-014-9943-z.

Johnson, D. S. (1973). Near-optimal bin packing algorithms. URL https://dspace.mit.edu/bitstream/handle/1721.1/57819/17595570-MIT.pdf?sequence=2.

Karpinski, J., Zhigadlo, N., Katrych, S., Bukowski, Z., Moll, P., Weyeneth, S., Keller, H., Puzniak, R., Tortello, M., Daghero, D., Gonnelli, R., Maggio-Aprile, I., Fasano, Y., Fischer, O., Rogacki, K. a Batlogg, B. (2009). Single crystals of $lnfeaso_{1-x}f_x(ln = la, pr, nd, sm, gd)$ and $ba_{1-x}rb_xfe_2as_2$: Growth, structure and superconducting properties. *Physica C: Superconductivity*, **469**(9), 370–380. ISSN 0921-4534. doi: 10.1016/j.physc.2009.03.048. URL https://www.sciencedirect.

com/science/article/pii/S0921453409000707. Superconductivity in Iron-Pnictides.

Kaul, A. (1991). Computing minkowski sums of regular polygons. In *Proceedings of the 3rd Canadian Conf. on Computational Geometry, 1991*. URL https://ci.nii.ac.jp/naid/10022403089/.

Kysela, M. (2014). Plošná optimalizace - bin packing problem. URL https://theses.cz/id/stqw29/.

Lee, C. C. a Lee, D. T. (1985). A simple on-line bin-packing algorithm. *Journal of the ACM (JACM)*, **32**(3), 562–572. doi: 10.1145/3828.3833. URL https://www.scopus.com/record/display.uri?eid=2-s2.0-0022093691&origin=inward.

Lin, Z. a Li, Y. (2009). An efficient algorithm for intersection, union and difference between two polygons. In *2009 International Conference on Computational Intelligence and Software Engineering*, pages 1–4. doi: 10.1109/CISE.2009.5364458. URL https://ieeexplore.ieee.org/document/5364458.

Liu, H.-y. a He, Y.-j. (2006). Algorithm for 2d irregular-shaped nesting problem based on the nfp algorithm and lowest-gravity-center principle. *Journal of Zhejiang University SCIENCE A*, **7**, 570–576. doi: 10.1631/jzus.2006.A0570. URL https://doi.org/10.1631/jzus.2006.A0570.

Lopez, E., Ochoa, G., Terashima-Marín, H. a Burke, E. (2013). An effective heuristic for the two-dimensional irregular bin packing problem. *Annals of Operations Research*, **206**, 241–264. doi: 10.1007/s10479-013-1341-4. URL https://www.researchgate.net/publication/256497455_An_effective_heuristic_for_the_two-dimensional_irregular_bin_packing_problem.

Margalit, A. a Knott, G. (1989). An algorithm for computing the union, intersection or difference of two polygons. *Computers & Graphics*, **13**, 167–183. doi: 10.1016/0097-8493(89)90059-9.

Martinez-Sykora, A., Alvarez-Valdes, R., Bennell, J., Ruiz, R. a Tamarit, J. (2017). Matheuristics for the irregular bin packing problem with free rotations. *European Journal of Operational Research*, **258**(2), 440–455. ISSN 0377-2217. doi: 10.1016/j.ejor.2016.09.043. URL https://www.sciencedirect.com/science/article/pii/S0377221716307950.

Nguyen Huu, L. (2015). Filtering techniques and search heuristics based on placement domain analysis for the two-dimensional irregular packing problem analysis for the two-dimensional irregular packing problem. Master's thesis, Keio University. URL https://www.researchgate.net/publication/340083116_Filtering_Techniques_and_Search_Heuristics_Based_on_Placement_Domain_Analysis_for_the_Two-Dimensional_Irregular_Packing_Problem_Analysis_for_the_Two-Dimensional_Irregular_Packing_Problem.

Nye, T. (2000). Stamping strip layout for optimal raw material utilization. *Journal of Manufacturing Systems*, **19**(4), 239–248. ISSN 0278-6125. doi: 10.1016/S0278-6125(01)80003-0. URL https://www.sciencedirect.com/science/article/pii/S0278612501800030.

Oliveira, J. F., Gomes, A. M. a Ferreira, J. S. (2000). Topos–a new constructive algorithm for nesting problems. *OR-Spektrum*, **22**(2), 263–284. doi: 10.1007/s002910050105. URL https://doi.org/10.1007/s002910050105.

Qiao, J. a bmtm (2018). Deepnest. URL https://github.com/Jack000/Deepnest/.

Qiao, J., bmtm, Nordby, J., Daniel, Collaud, G., Lejeune, O., Rees, J. a Sitkin, J. (2019). Svgnest. URL https://github.com/Jack000/SVGnest.

Rocha, P. (2019). Robust nfp generation for nesting problems. *ArXiv*, **abs/1903.11139**. URL https://arxiv.org/pdf/1903.11139.pdf.

Sato, A. K., Martins, T. C. a Tsuzuki, M. S. G. (2012). An algorithm for the strip packing problem using collision free region and exact fitting placement. *Computer-Aided Design*, **44**(8), 766–777. ISSN 0010-4485. doi: 10.1016/j.cad.2012.03.004. URL https://www.sciencedirect.com/science/article/pii/S0010448512000565.

Sethapakdi, T., Anderson, D., Reginald Chua Sy, A. a Mueller, S. (2021). Fabricaide: Fabrication-aware design for 2d cutting machines. URL https://github.com/DanielLiamAnderson/Packaide.

Shalaby, M. A. a Kashkoush, M. (2013). A particle swarm optimization algorithm for a 2-d irregular strip packing problem. doi: 10.4236/ajor.2013.32024. URL https://www.scirp.org/journal/paperinformation.aspx?paperid=29233.

Skiena, S. S. (1999). Who is interested in algorithms and why? lessons from the stony brook algorithms repository. *SIGACT News*, **30**(3), 65–74. ISSN 0163-5700. doi: 10.1145/333623.333627. URL https://dl.acm.org/doi/10.1145/333623.333627.

Song, Y., Van Dyke, J., Lum, I., White, B., Jang, S., Yazici, D., Shu, L., Schneidewind, A., Čermák, P., Qiu, Y. a kol. (2016). Robust upward dispersion of the neutron spin resonance in the heavy fermion superconductor ce 1- x yb x coin 5. *Nature communications*, **7**(1), 1–10. doi: 10.1038/ncomms12774. URL https://www.nature.com/articles/ncomms12774.

Sosa, A., Terashima-Marín, H., Ortiz-Bayliss, J. C. a Conant-Pablos, S. (2016). Grammar-based selection hyper-heuristics for solving irregular bin packing problems. pages 111–112. doi: 10.1145/2908961.2908970. URL https://dl.acm.org/doi/10.1145/2908961.2908970.

Švanda, O. (2020). Software pro efektivní využití materiálu při 2d obrábění. Master's thesis, Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky. URL http://hdl.handle.net/11012/191851.

Terashima-Marín, H., Ross, P., Farías-Zárate, C., López-Camacho, E. a Valenzuela-Rendón, M. (2010). Generalized hyper-heuristics for solving 2d regular and irregular packing problems. *Annals of Operations Research*, **179**(1), 369–392. doi: 10.1007/s10479-008-0475-2. URL https://www.researchgate.net/publication/220462558_Generalized_hyper-heuristics_for_solving_2D_Regular_and_Irregular_Packing_Problems.

Torres, J., Hitschfeld, N., Ruiz, R. O. a Ortiz-Bernardin, A. (2020). Convex polygon packing based meshing algorithm for modeling of rock and porous media. In *International Conference on Computational Science*, pages 257–269. Springer. doi: 10.1007/978-3-030-50426-7_20. URL https://doi.org/10.1007/978-3-030-50426-7_20.

Vidal, R. V. (1993). *Applied simulated annealing*, volume 396. Springer. doi: 10.1007/978-3-642-46787-5. URL https://doi.org/10.1007/978-3-642-46787-5.

Wałoszek, D. (2021). Online2dirregularbpp. URL https://github.com/Naimad1CZ/Online2DIrregularBPP.

Yang, S. a Prinway (2020). 2d-irregular-packing-algorithm. URL https://github.com/seanys/2D-Irregular-Packing-Algorithm/.

Zilu, W. a Shan, Y. (2020). Learn to pack with reinforcement learning and shape signature. URL https://github.com/seanys/Learn-to-Pack.

Seznam obrázků

A. Přílohy

A.1 První příloha

Repositář obsahující zdrojový kód, datasety a výsledky benchmarků včetně obrázků vypočtených umístění polygonů.