FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Rozmisťování objektů pro 3D tisk jako problém s omezeními
Student:	Lukáš Pokorný
Vedoucí:	prof. RNDr. Pavel Surynek, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

3D tisk se v současnosti stává stále důležitější výrobní technologií a některé oblasti, jako je prototypování, jsou již bez 3D tisku nepředstavitelné. Pro efektivní využití 3D tiskárny je klíčové rozmístit na povrch tiskového plátu co nejvíce objektů [1], čímž se redukuje čas na zahájení tisku a čas na obsluhu tiskárny. Zanedbáme-li třetí rozměr, můžeme úlohu chápat jako variantu rozmisťování 2D objektů do omezené části roviny tak, aby se nepřekrývaly [2], což je těžká kombinatorická úloha. K řešení rozmisťování objektů pro 3D tisk bychom chtěli využít formulace problému jako splňování omezení [3]. Úkoly pro řešitele jsou následující:

1. Seznamte se s problematikou 3D tisku a technikami splňování omezení.

2. Formulujte problém rozmisťování objektů pro 3D tisk jako problém splňování omezení, zaměřte se na vhodné předzpracování úlohy, například 2D objekty reprezentujte vhodně zdecimovanou obálkou.

3. Otestujte možnosti řešení navržené formulace pomocí současných řešičů problémů s omezeními, jako je Gecode, nebo OR-Tools. K testům využijte objekty z existujících databází pro 3D tisk, jako je Printables.

[1] Stefan Edelkamp, Paul Wichern: Packing Irregular-Shaped Objects for 3D Printing. KI 2015: 45-58.

Elektronicky schválil/a Ing. Magda Friedjungová, Ph.D. dne 12. února 2024 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

[2] Fowler, Robert J.; Paterson, Michael S.; Tanimoto, Steven L. (1981-06-13). "Optimal packing and covering in the plane are NP-complete". Information Processing Letters. 12 (3): 133–137.

[3] Rina Dechter: Constraint processing. Elsevier Morgan Kaufmann 2003, ISBN 978-1-55860-890-0, pp. I-XX, 1-481

Bakalářská práce

ROZMISŤOVÁNÍ OBJEKTŮ PRO 3D TISK JAKO PROBLÉM S OMEZENÍMI

Lukáš Pokorný

Fakulta informačních technologií
Katedra aplikované matematiky
Vedoucí: prof. RNDr. Pavel Surynek, Ph.D.
16. května 2024

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat především doc. RNDr. Pavlu Surynkovi, Ph.D.
za ochotu a vedení při psaní této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 16. května 2024

Abstrakt

Tato práce se bude zabývat rozmístěním 3D objektů, které se redukují na 2D objekty pohledem ze shora, na tiskovou plochu 3D tiskárny, pomocí Problému s omezeními(CSP). To znamená představit možné řešení zpracování STL souboru na 2D objekt, který se dál bude používat pro řešiče OR-Tools a Gecode, pro které se představí formulace a následná implementace. Výsledky z obou řešičů se poté srovnají a následně se srovnají i použití samotných řešičů.

Klíčová slova Problém s omezeními, OR-Tools, Gecode, 3D tisk, optimalní balení v rovině, C++

Abstract

This thesis is going to deal with the layout of 3D objects, which are reduced to 2D objects from the top view, on the print surface of a 3D printer, using the Constraints Satisfaction Problem (CSP). This means presenting a possible solution for processing the STL file into a 2D object, which will further be used for OR-Tools and Gecode solvers, for which the formulation and subsequent implementation will be presented. The results from both solvers are then compared, and then the use of the solvers themselves is also compared.

Keywords Constraint satisfaction problem, OR-Tools, Gecode, 3D printing, Optimal packing in the plane, C++

Seznam zkratek

CSP	Problém s omezeními
CP-SAT	Řešič problému s omezeními za pomoci SAT

Úvodní část

Úvod

3D tisk se jak v Česku, tak i ve světě stal fenoménem a nástrojem, který se dá použít takřka ve všech odvětvích, jako příklad lze uvézt výroba součástek, umělecká díla, koníček v domácím prostředí.

3D tisk přináší spoustu problémů, jeden z nich je rozmístění objektů na tiskové ploše, ať už z důvodu využití nahřívání tiskové plochy ve středu tiskové podložky, nebo i možnost umístit co nejvíce objektů na tiskovou plochu a tím optimalizovat počet tisknutých objektů. Tímto problémem se tato práce zaobírá, samotný problém je těžký a najít optimální řešení je na dlouhou dobu, musela by se vyzkoušet každá kombinace pozice a otočení objektu, naštěstí ne vždy je potřeba výsledek optimální, stačí nám takové řešení, které je považováno za skoro optimální. Tuto skutečnost se snaha dosáhnout za použití metody problému s omezeními. Metoda funguje na principu přidání omezení na proměnné, prohledání těchto kombinací, a následné zvolení takového řešení, které splňuje daná omezení, najití takového řešení může být o dost rychlejší jak vyzkoušení všech řešení, za ceny toho, že řešení nemusí být optimální.

Pro samotnou metodu již existuje řada řešičů, k nejznámějším open-source řešičům patří OR-Tools od společnosti Google a řešič Gecode od Christian Schulteho, po jeho smrti spravován komunitou. Práce si tedy dává za cíl ukázat rozdílnosti mezi těmito řešiči pro řešení problému s omezeními. Práce je určená pro další rozšíření v rámci problému s omezeními, ať už jako výzkum, či další, kteří by podobné problémy chtěli řešit metodou problému s omezeními.

Práce se nejprve bude zaměřovat na potřebné pojmy pro 3D tisk, zpracování souborů ve formátu STL pro potřeby problému s omezeními [1], problému s omezeními a charakterizaci řešičů. Dále se představí formulace problému s omezeními pro 3D tisk, kde objekty se budou umisťovat k prostředku tiskové plochy, což umožňuje lepší efektivitu vyhřívání. V další části se formulace následně implementuje pro řešiče OR-Tools a Gecode v programovacím jazyku C++. Poslední část, experimentální část, se zabývá porovnáním výsledků z řešičů, porovnání jejich rychlostí, schopností, či obtížnost jejich používání.

Cíl práce

Cílem teoretické části je vysvětlit potřebné pojmy týkající se problému úsporného rozmístění na tiskovém plátu za použití metody problému s omezeními, neboli constraint satisfaction problem. Dále si tedy vysvětlit problém s omezeními a práci s STL soubory, neboli zkratka slova Stereolitografie, jejich použitou 2D vizualizaci a modifikaci. Další cíl je seznámit se současnými řešičemi problémů s omezeními, konkrétně OR-Tools a Gecode.

Cílem praktické části je navrhnout formulaci problému na základě teoretické části. Dalším cílem je danou formulaci implementovat pomocí současných řešičů problémů s omezeními, jako je OR-Tools a Gecode. Výsledkem praktické části je poté porovnání použitých řešičů, pro Constraint satisfaction problem, na navržené formulaci problému. Mezi porovnání patří rychlost, kvalita výsledku a jednoduchost, či obtížnost formulaci konkrétního problému v řešiči.

Kapitola 1
Teoretická část

Tato část se zabývá vysvětlením potřebných informací o 3D tisku, mezi které patří hlavně formát STL, a také jak 3D tiskárna vlastně funguje. Na 3D tisk přímo navazuje optimální balení a krytí v rovině, neboli problém, který přímo souvisí s tématem této práce. Následuje základní pojmy problému s omezeními, s příkladem problému rectangle packing. Nakonec části problému s omezeními je vysvětlena propagace proměnných pro problém s omezeními a reifikace. Teoretická část je zakončena krátkým seznámením se s řešiči, které budou figurovat v pozdější části, neboli části ponechanou pro implementaci problému rozmisťování objektů na tiskové ploše s oemzeními.

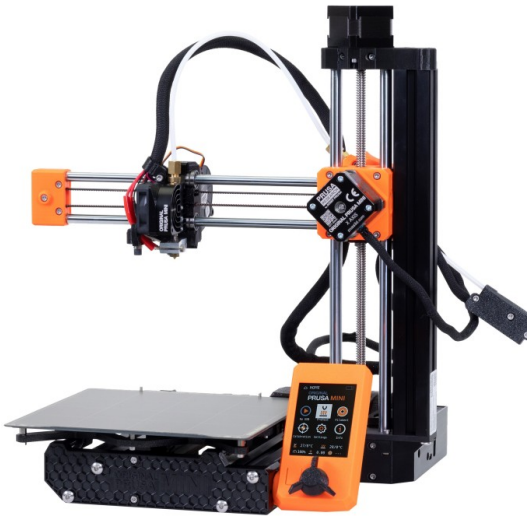
1.1 3D tisk

3D tisk je proces, při kterém se třírozměrné objekty vytvářejí postupným vrstvením materiálu na základě digitálního modelu. Tento proces umožňuje vytvářet složité geometrické tvary, které by jinak byly obtížné nebo nemožné vyrobit tradičními metodami. Princip 3D tisku spočívá v tom, že digitální model objektu je rozdělen na tenké vrstvy a poté je postupně vytisknut pomocí 3D tiskárny. Existuje několik různých metod tisku, včetně Fused Deposition Modeling (FDM), Stereolithography (SLA), Selective Laser Sintering (SLS) a další. V každé metodě se používají různé typy materiálů, jako jsou plastové filamenty, pryskyřice nebo práškové materiály, které se postupně nanášejí a spojují do požadovaného tvaru. 3D tiskárny často pracují s velkým rozsahem materiálů, což umožňuje vytvářet objekty s různými vlastnostmi, jako jsou pevnost, pružnost, transparentnost a další.

Výhody 3D tisku zahrnují možnost vytvářet prototypy, testovat návrhy s minimálními náklady a časem, možnost vytvářet jedinečné a přizpůsobené produkty a schopnost rychle reagovat na změny nebo individualizované požadavky zákazníků. Nicméně, 3D tisk má také některé omezení a výzvy, jako je omezená velikost tisknutelných objektů, omezené vlastnosti materiálů a potřeba specifických znalostí a dovedností pro efektivní použití těchto technologií. Celkově je 3D tisk technologie s širokým spektrem aplikací od průmyslové výroby po tvorbu uměleckých děl a medicínské aplikace.

1.1.1 STL formát

Definice Mesh: V kontextu 3D tisku, termín mesh odkazuje na digitální reprezentaci objektu, která je složena z množiny trojúhelníků (nebo jiných jednoduchých geometrických tvarů), které spolu tvoří síť [3].
Definice Facet: Facet je pojem, který se používá pro popis jednoho z geometrických tvarů, které tvoří mesh, nejčastěji to jsou trojúhelníky [3].



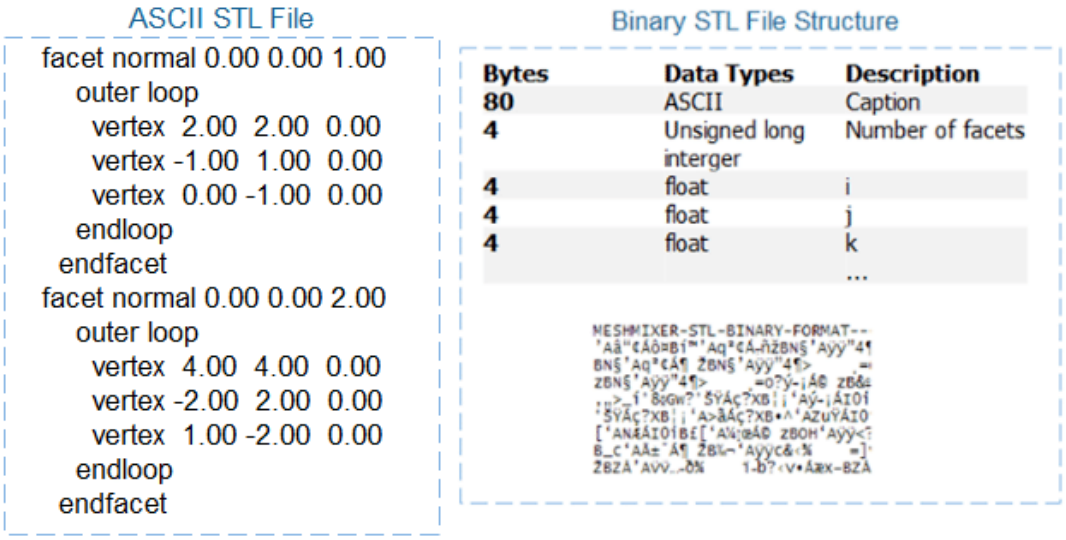
Obrázek 1.1 Částečně sestavená 3D tiskárna Original Prusa MINI+ [2]

Existuje mnoho způsobů jak reprezentovat 3D modely, ale pro 3D tisk se nejčastěji používá trojúhelníková síť. Je to kolekce bodů, hran a stěn ve trojrozměrném kartézském souřadném systému. Facetem v případě této práce je vždy trojúhelník. Výhoda použití trojúhelníku ve 3D prostoru je, že tři body, neležící na stejné přímce, vždy tvoří trojúhelník. Formát STL, který slouží právě pro ukládání triangulární meshe, je zkratka slova stereolitography, neboli česky stereolitografie (stereolitografie je metoda vytváření objektů pomocí postupného vytvrzování polymerů), byl vyvinut v roce 1987 společností 3D Systems, jako univerzální formát pro rapidní prototypování. STL není otevřeným formátem, ale je velmi rozšířen, podporuje jej mnoho programů, nejen ze světa 3D tisku. STL není jediný existující formát pro ukládání triangulární meshe, mezi další patří formáty OBJ, AMF, 3MF, ale spousta dalších. Pro čtení formátu v C++ lze využít například knihovnu CGAL [4], nebo Admesh [5].

1.1.1.1 STL reprezentace

STL reprezentace je převzata ze stránky `https://www.fabbers.com/tech/STL_Format#Sct_ASCII` sloužící pro archivaci materiálů ohledně "digital fabricators"(digitální fabrikátory, jedno z původních jmén pro 3D tiskárny) [6] tato stránka je pak převzata z knížky Automated Fabrication od Marshalla Burnse [7].

ASCII reprezentace začíná klíčovým slovem solid a názvem samotného meshe. Poté následuje definice všech facetů a soubor končí direktivou endsolid a opět názvem meshe. Definice facetu obsahuje informaci o normálovém vektoru a o třech vrcholech. Pořadí vrcholů facetu musí splňovat pravidlo pravé ruky, tedy jestliže palec ukazuje ve směru normály (ven z objektu), stočené prsty udávají pořadí vrcholů. Jednotlivá čísla se dají reprezentovat pomocí notace čísel s plovoucí desetinou čárkou, samozřejmě jako jinde v programátorském světě, místo desetinné čárky se používá desetinná tečka, lze se tedy setkat např. s hodnotami 1, 0.5 nebo $2.648000e-002$. Na ušetření místa se poté často používá binární formulace STL souboru, tím se šetří místo, soubor se tak ale stane velmi těžce čitelný pro lidi.

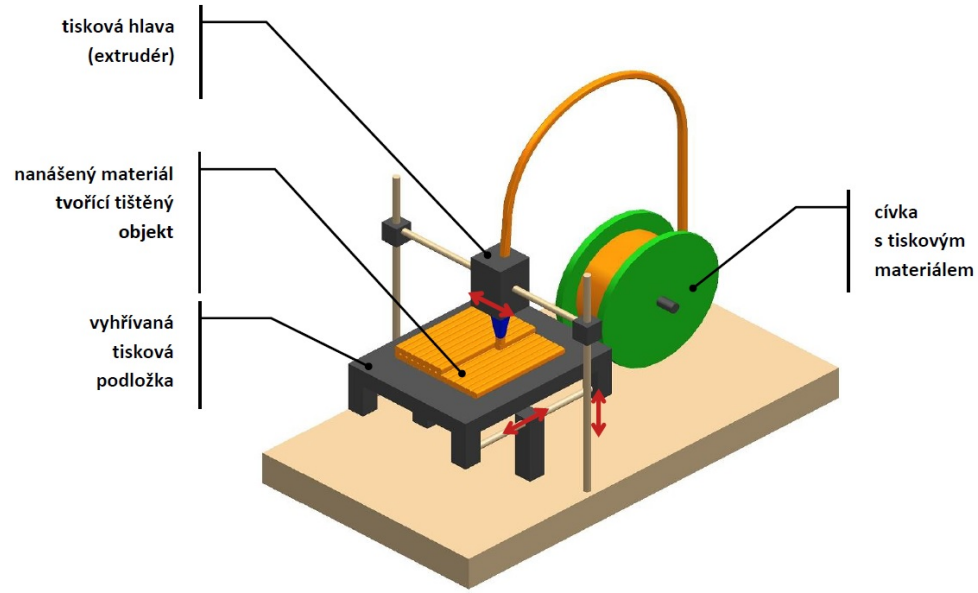


Obrázek 1.2 Příklad STL reprezentace v ASCII a v binární struktuře [8]

1.1.2 3D tiskárna

3D tiskárny fungují podobně jako 2D tiskárny, na tiskárnu se také neodesílají přímo .pdf nebo .tex dokumenty, ale musejí být převedeny do PostScriptu. 3D tiskárny tedy konkrétně nepotřebují PostScript, ale přímé instrukce, kdy a kde má tiskárna pohnout tiskovou hlavu nebo posunout válec [9]. Na trhu je spoustu 3D tiskáren, mezi české výrobce patří například Prusa Research a.s. `www.prusa3d.com`, nebo i například je možnost si pouze koupit základní tiskárnu a postupně si jí vylepšovat pomocí samotné 3D tiskárny, projekt RepRap je pokus o open-source 3D tiskárnu, kde díly si tiskne sám uživatel tiskárny. Více o projektu reprap na stránce `https://reprap.org/`. Mezi důležité častí 3D tiskárny patří [9]:

- **Filament**: Materiál využívaný pro tisk objektů.
- **Extruder**: Přenáší zahřátý tiskový materiál na tiskovou plochu.
- **Tryska**: Aplikuje roztavený filament na tiskovou plochu.
- **Tisková plocha**: Podložka pro tisknutý objekt, může být nahřívaná na lepší přilnavost tisknutého objektu.



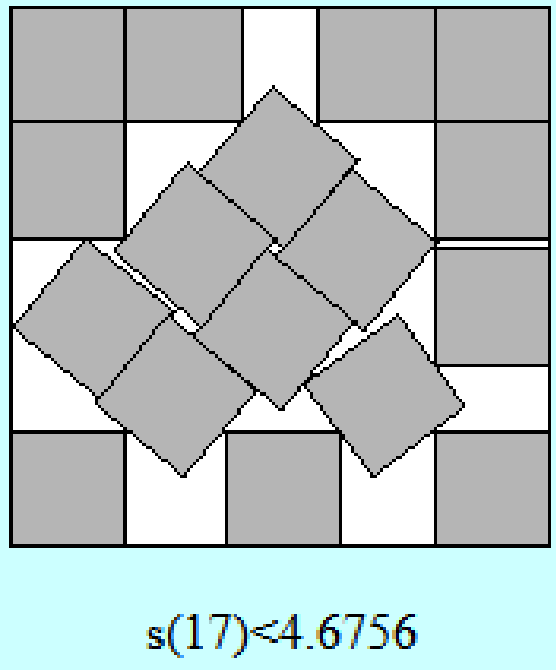
Obrázek 1.3 Části 3D tiskárny [10]

Tiskovými daty pro 3D tiskárny jsou přesné instrukce kdy, odkud, kam a jak rychle se má tryska tiskárny pohnout a v jaký moment a kolik se vytlačí filamemtu [9]. Tyto příkazy tiskárna poslouchá a provádí je bez ohledu na to, aby měla tiskárna ponětí o tom, co vlastně ve výsledku tiskne. Tím pádem se pro daný objekt musí před tiskem nastavit pozici objektu, příp. objektů a to už není jednoduchý úkol, když mají objekty různé rozměry a tvary [11].

1.2 Optimální balení a krytí v rovině

Problém optimálního krytí v rovině je rozhodnutí, zda nějaká množina O geometrických objektů může kompletně pokrýt nějakou množinu S bodů v rovině. Pro tuto práci je ale důležitější problém optimálního balení v rovině, kde se musí rozhodnout jestli nějaká množina O geometrických objektů může být položena neprotínajícím se způsobem do nějakého regionu R (ne nutně spojitým objektem) v rovině. V [12] již dokázali, že oba tyto problémy, i v jejich velmi omezené formě, jsou NP-úplný, pomocí redukce na 3-SAT.

Příklad takového problému je jestliže existují objekty, v tomto případě čtverce o hraně velikosti 1, se kterými se musí pokrýt co nejmenší region čtverce, aniž by se objekty v regionu překrývaly. Nejlepší možný zatím známý výsledek [13] velikosti regionu V pro 17 objektů je $V < 4,6756$.



Obrázek 1.4 Pokrytí regionu tvaru čtverce 17 menšími čtverci o hraně velikosti 1 [13]

1.3 Problém s omezeními

Problém s omezeními vychází hlavně z práce [14]. Následující část je z velké části přebrána, nebo zpracována právě z knihy Constraint processing [14].

Obecně platí, že problémy splňující omezení zahrnují dvě důležité složky proměnných s přidruženými doménami a omezeními. Následující část definuje každou základní komponentu problému s omezeními. Za prvé, každý problém s omezeními musí zahrnovat proměnné: objekty nebo položky, které mohou nabývat různých hodnot. Množina možných hodnot pro danou proměnnou se nazývá její doména. Jako příklad se dá uvést snaha najít přijatelné uspořádání sedadel pro večeři, jedna z možných interpretací je, že jako proměnné slouží židle, z nichž každá má stejnou doménu, což je seznam všech hostů.

Druhou složkou každého problému s omezeními je soubor samotných omezení. Omezení jsou pravidla, která omezují hodnoty, který lze proměnné nebo kombinaci proměnných přiřadit. Pokud musí hostitel a hostitelka sedět na dvou koncích stolu, jejich výběr míst je omezený. Pokud dva znesváření hosté nesmí být umístěni vedle sebe nebo přímo proti sobě, musí se toto omezení zahrnout do celkového prohlášení o problému.

Je důležité si všimnout, že často existuje více než jeden způsob, jak modelovat problém. V předchozím příkladu se pomocí stejné logiky mohlo za proměnné vzít hosty a jejich domény sada židlí u stolu. V tomto případě, za předpokladu vzájemné korespondence mezi židlemi a hosty, výběr nečiní žádný rozdíl, ale v jiných případech se jedna formulace problému může hodit k technikám řešení snadněji než jiná.

Model, který zahrnuje proměnné, jejich domény a omezení, se nazývá omezující síť neboli ”constraint network”, v této práci bude používáno pojmenování *splňování omezení*, nebo také *problém s omezeními*. Použití termínu ”síť”lze vysledovat k počátkům práce na splnění omezení, kdy bylo zaměření výzkumu omezeno na soubory omezení, jejichž závislosti byly zachyceny pomocí jednoduchých grafů, v dnešní době může tento pojem způsobit zmatení, jelikož je používán ve spoustě odvětví informačních technologií. Řešením je přiřazení jedné hodnoty z její domény každé proměnné tak, aby nedošlo k porušení žádného omezení. Problém může mít jedno, mnoho

nebo žádné řešení. Problém, který má jedno nebo více řešení, je splnitelný nebo konzistentní. Pokud neexistuje možné přiřazení hodnot proměnným, které by vyhovovalo všem omezením, pak je problém nesplnitelný nebo nekonzistentní. Typické úkoly v problému s omezeními jsou určení, zda řešení existuje, nalezení jednoho nebo všech řešení, zjištění, zda lze částečné řešení rozšířit na úplné řešení, a nalezení optimálního řešení vzhledem k dané nákladové funkci (angl. cost function). Takové úkoly se označují jako problémy splňující omezení (CSP).

1.3.1 Základní pojmy

Splňování omezení [14] R se skládá z konečný množiny proměnných $X = \{x_1, ..., x_n\}$, ke kterým jsou přiděleny jejich domény $D_i = \{v_1, ..., v_k\}$ a množina omezení $C = \{C_1, ..., C_t\}$. Splňování omezení, neboli problém s omezeními je tedy trojice (X, D, C).

Omezení C_i je relace R_i definovaná na podmnožině proměnných $S_i, S_i \subseteq X$. Relace označuje současné přiřazení legálních hodnot proměnným. S_i se nazývá rozsah R_i. Jestliže $S_i = \{x_{i_1}, ..., x_{i_\tau}\}$, potom R_i je podmnožinou kartézského produktu $D_{i_1} \times \cdots \times D_{i_\tau}$. Na omezení se lze dívat jako na pár $C_i = \langle S_i, R_i \rangle$. Pokud je identita rozsahu zřejmá, poté se často omezení C_i identifikuje s relací R_i. Alternativně, pro jasnost, relace omezení může být označena jako R_{S_i}. Rozsah je označován explicitně jako $\{x, y, z\}$, nebo, pokud není možnost záměny, jako xyz. Například, omezení definované na proměnných x, y a z, jejich relace je R označíme R_{xyz}, nebo jako $\langle xyz, R \rangle$. Množinu rozsahů $S = \{S_1, ..., S_i, ..., S_t\}$ se nazývá schéma problému s omezeními. Bez újmy na obecnosti, předpokládáme pro relační omezení, že máme definované pouze jedno omezení nad podmnožinou S_i v S. A to, jestliže $i \neq j$, potom $S_i \neq S_j$. Tento předpoklad může být ignorován, pokud je téma algebraická nebo booleovská omezení.

Arita omezení odkazuje na kardinalitu, nebo velikost jejího rozsahu. Unární omezení je definováno na jedné proměnné; binární omezení, na dvou proměnných. Binární splňování omezení má pouze unární a binární omezení.

1.3.2 Řešení splňování omezení

Jestliže je proměnné přidělena hodnota z její domény, potom je proměnná ***dosazena***.

Dosazení podmnožiny proměnných je přidělení hodnoty z jejich domén každé proměnné z podmnožiny. Formálně, dosazení množiny proměnných $\{x_{i_1}, ..., x_{i_k}\}$ je uspořádáná n-tice párových hodnot $(\langle x_{i_1}, a_{i_1} \rangle, ..., \langle x_{i_k}, a_{i_k} \rangle)$, kde pár $\langle x, a \rangle$ representuje přiřazení hodnoty a proměnné x, a kde a je v doméně x. Také je možnost použít notaci $(x_1 = a_1, ..., x_i = a_i)$. Pro jednoduchost se často zkracuje $(\langle x_1, a_1 \rangle, ..., \langle x_i, a_i \rangle)$ na $\bar{a} = (a_1, ..., a_i)$.

1.3.2.1 Splnění omezení

Dosazení nebo n-tice $\bar{a}$ splňuje omezení $\langle S, R \rangle$, tehdy a pouze tehdy jestliže je definována na všech proměnných z množiny S a komponenty n-tice $\bar{a}$ asociovaná s S jsou přítomny v relaci R. Například, nechť $R_{xyz} = \{(1, 1, 1), (1, 0, 1), (0, 0, 0)\}$. Potom dosazení $\bar{a}$, jejíž rozsah je $\{x, y, z, t\}$, daná $\bar{a} = (\langle x, 1 \rangle, \langle y, 1 \rangle, \langle z, 1 \rangle, \langle t, 0 \rangle)$ splňuje R_{xyz}, protože její projekce na $\{x, y, z\}$ je $(1, 1, 1)$, což je element R_{xyz}. Ale, dosazení $(\langle x, 1 \rangle, \langle y, 0 \rangle, \langle z, 0 \rangle, \langle t, 0 \rangle)$ porušuje R_{xyz}, protože $(1, 0, 0)$ není členem R_{xyz}.

1.3.2.2 Konzistentní částečné dosazení

Částečné doszení je konzistenstní, jestliže splňuje všechny omezení, jejichž rozsah nemá žádné nedosazené proměnné. Projekce n-tice $\bar{a}$ na podmnožinu jejího rozsahu S je označována jako $\bar{a}[S_i]$ (nebo taky $\pi_{S_i}(\bar{a})$). Za použití této notace, $\bar{a}$ nad S je konzistenstní relativně na splňování omezení R, tehdy a pouze tehdy, když pro všechny S_i ve schématu R, tak že $S_i \subseteq S, \bar{a}[S_i] \in R_{S_i}$.

1.3.2.3 Řešení

Řešení splňování omezení $R = (X, D, C)$, kde $X = \{x_1, ..., x_n\}$ je dosazení všech proměnných sítě, který splňují všechny omezení. Relace řešení $sol(R)$, také označována jako ρ_X, je definována jako $sol(R) = \{\bar{a} = (a_1, ..., a_n) | a_i \in D_i, \forall S_i \in \text{schéma } R, \bar{a}[S_i] \in R_i\}$. Problém z omezení vyjadřuje, či reprezentuje relace na všech jejích řešeních, jestliže splňování omezení R, nad X a podmnožinu proměnných $A \subseteq X, sol(A)$, nebo ρ_A je množina všech konzistentních dosazení nad A.

1.3.3 Numerické a booleovské omezení

Formulace problému pomocí relací, kde jsou vyjmenovány všechny možnosti je sice dobré pro základní vysvětlení problému s omezeními, ale pro práci se moc nehodí, problém se tímto velmi znepříjemní a samotná formulace je tak časově náročnější než poté najití celého řešení. Pro pohodlnost a stručnost se tak často používá místo relací, formulace problému pomocí numerických, nebo booleovských výrazů. Aritmetické a booleovské omezení je příklad alternativního jazyka pro popis omezení. První umožňuje stručnější výrazy, zatímco booleovské výrazy, nejenom, že jsou omezený na pouze domény o velikosti 2, vyjadřují zakázané n-tice, oproti povoleným. Numerické omezení vyjadřují omezení pomocí aritmetických výrazů.

Když hodnoty proměnných omezení jsou pouze přes 2 hodnoty, poté v takových případech k popisu různých vztahů je často používán výrokový jazyk.

1.3.4 Problém rectangle packing

Problém je převzán z knihy Heuristic Search [15]. Rectangle Packing představuje snahu položit množinu obdélníků do ohraničujícího obdélníku, tak aby se žádný dva obdélníky neprotínaly. Pro každý obdélník existuje proměnná, jejíž možné hodnoty jsou pozice, které by mohl zaujímat, aniž by překročil hranice obklopujícího obdélníku. Navíc máme mezi každou dvojicí obdélníků binární omezení, že se nemohou překrývat. Jsou dvě verze tohoto problému, rozhodovací verze a optimalizační verze.

Rozhodovací verze rectangle packing
Ve své rozhodovací verzi problém obsahuje množinu obdélníku r_i s jejich šířkou w_i a výškou h_i, $i \in \{1, ..., n\}$ a jeden ohraničující obdélník o šířce W a výšce H. Úkolem je najít takové přiřazení hodnot souřadnic v levém horním rohu obdelníků (x_i, y_i), že pro všechny obdélníky r_i platí:

- Každý obdélník je zcela obsažen v ohraničujícím obdélníku, to znamená, pro všechny $i \in \{1, ..., n\}$ máme $0 \leq x_i$, $0 \leq y_i$, $x_i + w_i \leq W$ a $y_i + h_i \leq H$.
- Žádné dva obdélníky r_i a r_j, s $1 \leq i \neq j \leq n$, se neprotánají, neboli platí $(x_i + w_i \leq x_j \vee x_j + w_j \leq x_i) \wedge (y_i + h_i \leq y_j \vee y_j + h_j \leq y_i)$.

Optimalizační verze rectangle packing
Jejím úkolem je najít nejmenší ohraničující obdélník, u kterého je možné najít takové přiřazení obdelníků, že žádný dva obdélníky se neprotínají.

Rectangle packing je důležitý pro plánovací úlohy. Uvažujme n úloh, kde každá úloha i vyžaduje určitý počet strojů m_i a určitou dobu zpracování d_i, $i \in \{1, ..., n\}$. Nalezení plánu s minimálními náklady je ekvivalentní problému rectangle packing s $w_i = d_i$ a $h_i = m_i$ pro $i \in \{1, ..., n\}$

Pro tento problém se může formulovat množina poměnných, domén a nakonec i množinu omezení v jazyce problému s omezeními.

- Proměnné X, proměnné jsou souřadnice obdélníků z množiny, pokud máme n obdélníků, poté $X = \{(x_1, y_1), ..., (x_n, y_n)\}$.

- Domény proměnných D, domény proměnných z množiny X, jsou všechny možné, přiřazení proměnných, tak, aby přiřazená hodnota byla stále v prostoru ohraničujícího obdélníku, tedy $D = \{(D_{1_x}, D_{1_y}), ..., (D_{n_x}, D_{n_y})\}$, pro dvojici (D_{i_x}, D_{i_y}) platí $(0 \leq D_{i_x} \wedge D_{i_x} \leq W) \wedge (0 \leq D_{i_y} \wedge D_{i_y} \leq H)$.

- Omezení C se skládá ze dvou množin, první C_1 obsahuje omezení, že každý obdélník je zcela obsažen v ohraničujícím obdélníku, neboli $C_1 = \{(0 \leq x_1 \wedge 0 \leq y_1 \wedge x_1 + w_1 \leq W \wedge y_1 + h_1 \leq H), ..., (0 \leq x_n \wedge 0 \leq y_n \wedge x_n + w_n \leq W \wedge y_n + h_n \leq H)\}$, podmnožina C_2 se poté skládá z omezení, že žádný dva obdélníky se neprotínají, $C_2 = \{(x_1 + w_1 \leq x_2 \vee x_2 + w_2 \leq x_1) \wedge (y_1 + h_1 \leq y_2 \vee y_2 + h_2 \leq y_1), ..., (x_1 + w_1 \leq x_n \vee x_n + w_n \leq x_1) \wedge (y_1 + h_1 \leq y_n \vee y_n + h_n \leq y_1), ..., (x_{n-1} + w_{n-1} \leq x_n \vee x_n + w_n \leq x_{n-1}) \wedge (y_{n-1} + h_{n-1} \leq y_n \vee y_n + h_n \leq y_{n-1})\}$.

Jelikož je ale výčtová definice množiny pro omezení složitá, ve formulacích problému rozmisťování objektů pro 3D tisk jako problém s omezeními se množina omezení bude definovat pomocí charakteristické společné vlastnosti neprotínání, která je pro rectangle packing obsazena v popisu rozhodovací verze problému. Problém recktangle packing [15] je stejně jako problém optimálního krytí a balení NP-těžký [12].

1.3.5 Propagace proměnných

Propagace proměnných, neboli inference, v rámci problému s omezeními zahrnuje proces, který při aplikování omezení omezuje splňování omezení, tak, že vytváří ekvivalentní, ale více explecitní splňování omezení. To znamená, že propagace má za cíl snížit oblast možných hodnot pro jednotlivé proměnné na základě omezujících podmínek. Tento proces slouží k urychlení vyhledávání řešení tím, že se snižuje počet kombinací, které je třeba zvážit. Omezení může být i tak účinné, že se doména omezí na pouze 1 hodnotu. Existuje několik technik na propagaci proměnných, mezi nejvíce používané patří:

- **Hranové konzistence** (Arc-consistency) jejím principem je zajistit, že každá proměnná v problému s omezeními má doménu, která je hranově konzistentní ("arc-consistent"), což znamená, že žádná hodnota v této doméně neporušuje žádnou omezující podmínku s jinou proměnnou.

- **Konzistence cesty** (Path-consistency) se zabývá kontrolou, zda jsou splněny omezující podmínky pro posloupnosti (cesty) proměnných.

- **Mezní konzistence** (Bounds-consistency) pro každou proměnnou určí dolní a horní mez, které omezují možné hodnoty této proměnné tak, aby byly splněny všechny omezující podmínky s ostatními proměnnými. V některých případech je hranová konzistence moc drahá, dobrý kompromis je mezní konzistence, která omezí doménu na interval hodnot, kde pouze konce intervalu musí splňovat hranovou konzistenci, pokud to doména nesplňuje, poté se může zmenšovat, dokud jí nesplňuje.

Propagace pak často funguje, tak že při přiřazení jedné hodnoty k proměnné se omezí i další proměnné, na které mohlo mít přiřazení efekt. Tyto konzisteční algoritmy, narozdíl od problémů, které se často pomocí problému s omezeními formulují, pracují v polynomiálním čase.

1.3.6 Prohledávání prostoru problému s omezeními

Po samotné propagaci, nastává výpočet a někdy se prostě stane, že ani samotná propagace úplně nepomohla, problém má stále hodně možností jak zvolit, pro najití řešení je potom nezbytné zkusit uhádnout, kterou proměnnou a jaká hodnota z její domény jí bude přiřazena, to může být náhodně, nebo lépe, za pomoci heuristiky, dobrý odhad vede k novému stavu, který je blíže

cíli. Často nastává situace, kdy jsou ještě nepřiřazené proměnné, ale není možnost je přiřadit, jakékoliv přiřazení porušuje omezení. Tento problém řeší algoritmus ***Backtracking***.

Algoritmus Backtracking u problému s omezeními počínaje první proměnnou postupně přiřazuje hodnotu každé proměnné, přičemž se ujistí, že každá přiřazená hodnota je konzistentní s dosud přiřazenými hodnotami, než přejde k další proměnné. Když algoritmus narazí na proměnnou, pro kterou není žádná hodnota z její domény konzistentní s předchozím přiřazením, nastane ***slepá ulička*** (dead-end). V tomto okamžiku dochází k backtrackingu, neboli hodnota přiřazená proměnné bezprostředně předcházející slepé uličce se změní, pokud je to možné, a hledání pokračuje. Algoritmus se zastaví buď, když je nalezen požadovaný počet řešení, nebo když lze dojít k závěru, že žádné řešení nebo žádné další řešení, neexistují. Problém algoritmu Backtracking je tzv. "***Thrashing***", kde algoritmus znovu objevuje již nalezené nesrovnalosti, neboli objevuje slepé uličky, Předejít thrashingu úplně nelze, ale lze ho omezit, pomocí ***Look-ahead*** algoritmů a ***Look-back*** algoritmů.

Look-ahead algoritmny Look-ahead se aktivují pokaždý, když se Backtracking připravuje vybrat další proměnnou, nebo další hodnotu z její domény. Algoritmus se snaží z omezených informací a z omezeného množství konzistence zjistit jak případná volba ovlivní další proměnné a jejich hodnoty, jakmile Look-ahead udělá dostatečný výpočet, např. do dostatečné hloubky, tak algoritmus může tento výsledek použít pro dosazení proměnné a jakou hodnotu proměnné přiřadit. Příklad takových algoritmů je Forward-Checking, nebo Arc-Consistency Look-Ahead.

Look-back Tyto algoritmy se aktivují, když Backtracking narazí na slepou uličku. Jsou dva typy, první typ si klade za cíl přeskočit ty proměnné, které jsou problémové při backtrackingu, neboli místo pouze jednoho kroku zpět může udělat větší, například na základě heuristiky, příklad takového algoritmu je algoritmus Backjumping. Druhý typ je zapamatovat si důvody pro slepou uličku a vytvořit nová omezení, které jí zabraňují, aby nenastaly stejné konflikty v procházení, na této bázi funguje například algoritmus Graph-Based Learning.

1.3.7 Reifikace

Hodně omezení existuje v reifikované variantě [16], tj. validita omezení je reflektována na kontrolní proměnnou typu bool. Vedle úplné reifikace existuje i poloviční reifikace. Pro proměnné typu integer x a y a jejich bool kontrolní proměnná b vytvoří propagátor pro reifikované omezení $(b = true) \Leftrightarrow (x = y)$, který propaguje podle následujících pravidel:

- Platí $b = true$, poté se propaguje $x = y$,
- platí $b = false$, poté se propaguje $x \neq y$,
- platí $x = y$, poté se propaguje $b = true$,
- platí $x \neq y$, poté se propaguje $b = false$.

Toto je úplná reifikace, propaguje ekvivalenci mezi omezením a kontrolní bool proměnnou. Vedle úplné reifikace ještě existuje polovniční reifikace, která propaguje pouze jedním směrem, tedy propaguje pouze implikaci. Například pro omezení $(b = true) \Rightarrow (x = y)$ poloviční reifikace funguje podle následujících pravidel:

- platí $b = true$, poté se propaguje $x = y$,
- platí $x \neq y$, poté se propaguje $b = false$.

1.3.8 Řešiče

Pro metodu problému s omezeními existují řešiče pro mnoho jazyků, pro Javu je to např. open-source knihovna Choco-solver, pro C++ je to například Gecode a OR-Tools. OR-Tools je sice napsán v jazyce C++, ale navíc poskytuje balíčky pro jazyky Java, Python a C#. Existuje i spousta

větví derivovaná již z existujících řešičů, například jazyk MiniZinc, postavěný na knihovně Gecode. Práce se bude zabývat hlavně právě řešiči Gecode a OR-Tools.

1.3.8.1 OR-Tools

OR-Tools je otevřený software pro kombinatorickou optimalizaci [17], který se snaží najít nejlepší řešení problému z velmi rozsáhlé množiny možných řešení. Několik příkladů problémů, které OR-Tools řeší:

- **Směrování vozidel:** Najít optimální trasy pro vozové parky, které vyzvedávají a doručují balíky s ohledem na omezení (např. „toto nákladní auto nemůže pojmout více než 20 000 liber“ nebo „všechny dodávky musí být uskutečněny během dvou hodin“).
- **Plánování:** Najít optimální plán pro komplexní sadu úloh, z nichž některé je třeba provést dříve než jiné, na pevné sadě strojů nebo jiných zdrojů.
- **Balení do přihrádek:** Zabalit co nejvíce předmětů různých velikostí do pevně stanoveného počtu přihrádek s maximální kapacitou.

Ve většině případů mají problémy, jako tyto, obrovské množství možných řešení, neboli příliš mnoho na to, aby je počítač všechny prohledal. K překonání tohoto problému používá OR-Tools nejmodernější algoritmy k zúžení vyhledávací množiny s cílem nalézt optimální (nebo téměř optimální) řešení.

OR-Tools zahrnuje řešiče pro [18]:

- **Programování s omezeními** Soubor technik pro nalezení proveditelných řešení problému vyjádřeného jako omezení (např. místnost nemůže být použita pro dvě události současně nebo vzdálenost od plodin musí být menší než délka hadice nebo nelze nahrát víc jak 5 TV pořadů najednou).
- **Lineární a smíšené celočíselné programování** Lineární optimalizátor Glop najde optimální hodnotu lineární cílové funkce s ohledem na sadu lineárních nerovností jako omezení (např. přiřazení lidí k práci nebo nalezení nejlepší alokace sady zdrojů při minimalizaci nákladů).
- **Směrování vozidel** Specializovaná knihovna pro identifikaci nejlepších tras vozidel s danými omezeními.
- **Grafové algoritmy** Kód pro hledání nejkratších cest v grafech, tocích minimálních nákladů, maximálních tocích a přiřazení lineárních součtů.

Rychlost OR-Tools dokazuje výhra zlata v mezinárodní soutěži v programování s omezeními MiniZinc Challenge každý rok od roku 2013 [19]. MiniZinc Challenge je každoroční soutěž řešitelů programování s omezeními v různých měřítcích. Koná se každoročně od roku 2008, přičemž vítězové jsou vyhlašováni na každoroční mezinárodní konferenci ACP (Asociace pro Programování s Omezeními) [20].

1.3.8.2 Gecode

Gecode je další z state of the art řešičů pro problémy s omezeními [21]. Je to open source C++ nástroj pro práci s problémy s omezeními a používá se i pro aplikace. Gecode je postaven otevřeně, to znamená, že umožňuje relativně lehké vytváření nových omezení, strategie pro větvení, nebo i vytváření vlastních prohledávačů. Potvrzení rychlosti Gecode je právě z již zmíněné soutěže Minizinc, kde Gecode získal zlaté medaile ve všech kategoriích mezi lety 2008 až 20212 [19]. Další výhoda Gecode je, že obsahuje rozsáhlou příručku [21], která obsahuje nejenom informace ohledně Gecode, ale dá se použít i na jiné řešiče, jejíž informace jsou ocenitelné i pro jiné řešiče, nebo i celkově pro téma problému s omezeními. Řešič umožňuje použití jak diskrétních, tak i spojitých proměnných [21].

Kapitola 2

Implementace problému rozmisťování objektů s omezeními

Tato část bude sloužit jako návrh a zpracování vlastního řešení, problém nstává hned na začátku formulace, při snaze najít takové omezení, které řeší právě neprotínání objektů. Knihovny OR-Tools a Gecode obsahují funkci pro obdélníky, které jsou rovnoběžné s osama x a y ve 2D prostoru, tyto funkce jsou perfektní pro řešení rectangle packing 1.3.4. To bohužel nesplňuje podmínky rotace, tato implementace by byla schopná fungovat pouze s rotací 0 stupňů. To znamená, že formulace problému rozmisťování objektů na tiskové ploše musí být formulována za použití jiných, ale již existujících omezení.

2.1 Ztvárnění objektu

Před samotným formulováním problému je zde problém samotného ztvárnění objektu, kde pokud se zanechá původní tvar objektu, bez ubrání na přesnosti, tak pravděpodobně bude výpočet samotného rozhodnutí, zda dva objekty se protínají dlouhý, ale samotné zpracování do řešičů bude, bez velkých kompromisů, velmi obtížné a v případě OR-Tools by to mohlo zahrnovat vytvoření nového omezení speciálně pro přesný tvary a jejich protnutí. Tato možnost je označována samotným tvůrcem jako velmi obtížná [22], kde se musí zahrnout všechny možnosti a omezení ještě speciálně formulovat pro CP-SAT řešič, který OR-Tools využívá. Kompromis se musí najít ještě před samotnou výpočetní částí řešiče. Možným řešením pro 2D prostor je použít čtyřstrom, myšlenka přímo vychází z článku "Packing Irregular-Shaped Objects for 3D Printing"[1], kde je použit oktálový strom, neboli taková struktura stromu, kde každý vnitřní vrchol má 8 potomků.

2.1.1 Příprava objektu

Čtyřstrom
Čtyřstromy jsou stromové datové struktury, ve kterých každý uzel ohraničuje čtvercový úsek prostoru a každý vnitřní uzel má přesně 4 potomky. Čtyřstrom je vhodný pro objekty ve 2D prostoru, pro 3D objekty je tu dále oktálový strom, ve kterém uzel ohraničuje krychlový úsek prostoru a má přesně 8 potomků. Důležité je samotné rozdělení to je uděláno pomocí bodů, které daný objekt, neboli v rovině mnohouhelník, vytvářejí.

Vytvoření čtyřstromu umožňuje knihovna pro tvary CGAL. CGAL je open source softwarový projekt, který poskytuje snadný přístup k efektivním a spolehlivým geometrickým algoritmům ve formě knihovny C++. CGAL se používá v různých oblastech vyžadujících geometrické výpočty,

jako jsou geografické informační systémy, počítačem podporované navrhování, molekulární biologie, lékařské zobrazování, počítačová grafika a robotika [23]. Samotný strom se vytváří pomocí upřesnění [24], které dále pracuje s 2 atributy:

- **Maximální hloubka:** jak moc hluboký bude vytvořený čtyřstrom.
- **Velikost vrcholu:** kolik maximálně bodů mnohouhelníku bude vrchol reprezentovat.

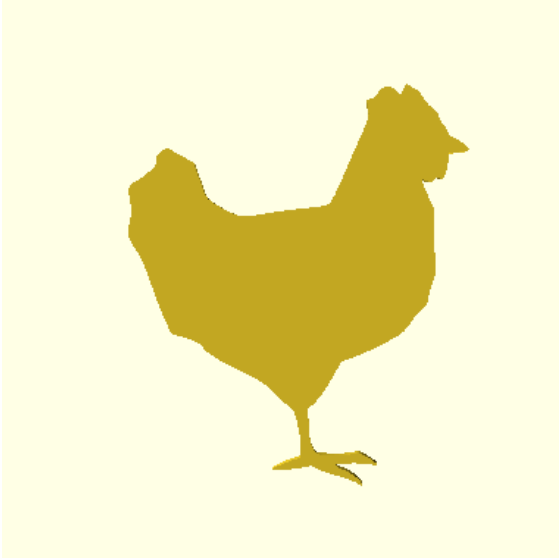
Upřesňování se zastaví, jakmile je porušena jedna z podmínek:

- Pokud má uzel více bodů mnohouhelníku než je hodnota atributu *velikostvrcholu*, ale čtyřstrom je již na maximální hloubce.
- Pokud nemá uzel maximální hloubku, ale má v sobě méně bodů mnohouhelníku než je atribut *velikostvrcholu*, poté taky není vrchol dále rozdělen.

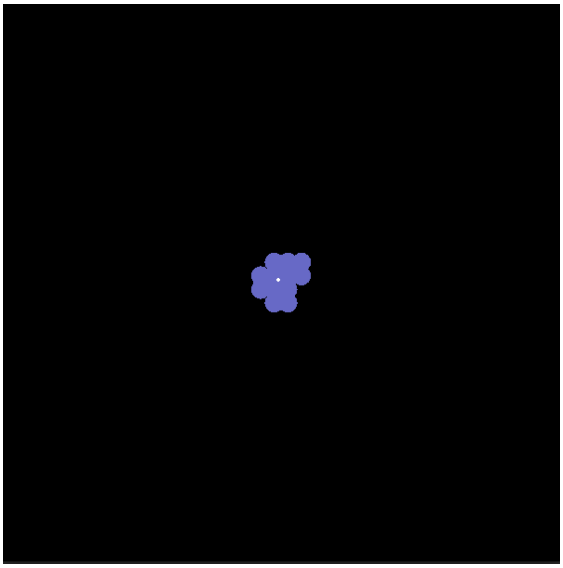
Další problém je jak z čtyřstromu získat potřebné datové struktury, které poté pomůžou s řešením problému. Na takovou strukturu jsou kladeny následující požadavky:

- Struktura zachycuje původní tvar objektu.
- Struktura umožňuje rotace.

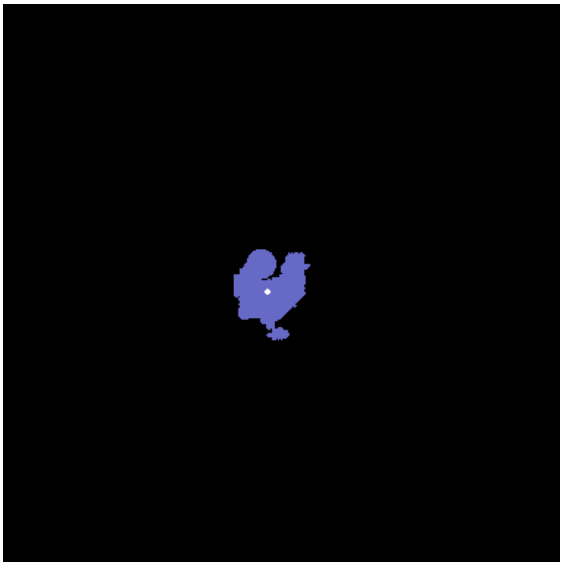
Tyto 2 podmínky pak ještě udávají třetí podmínku, Zachycení původního tvaru při rotaci. například takový čtverec, který by šel přímo vzít z čtyřstromu, by umožnil rychlý výpočet, zda se 2 čtverce, se souřadnicema x_1, y_1 a x_2, y_2 a délkou strany d_1 a d_2 protínají:
$(x_1 + d_1 \leq x_2) \vee (x_2 + d_2 \leq x_1) \vee (y_1 + d_1 \leq y_2) \vee (y_2 + d_2 \leq y_1)$, zkrátka je nutné zkontrolovat, jestli je jeden z útvarů mimo druhý. Problém nastává v případě rotací, neboli posun kolem středu původního mnohouhelníku a nastavení rotace samotnému čtverci. Rotace tedy bude na samotný čtverec, ale i na jeho tvar, buď je tvar čtverce ponechán, tedy čtverec bude pořád kolmý na osy x a y, tím se ztratí zachycení původního tvaru, nebo se i čtverec bude otáčet a to způsobí zhoršení náročnosti výpočtů. Kompromis mezi těmito možnostmi je použít, místo čtverce, kružnici. Kružnice nemusí tak dobře zachycovat původní tvar z čtyřstromu, ale rotace jsou mnohem jednodušší a nemají vliv na tvar samotné kružnice. Pro co největší zachování tvaru mnohouhelníku při rotacích, se použije kružnice. Např. kružnice $k_i \in K$, kde K je množina všech kružnic z čtyřstromu, každá kružnice si pamatuje vlastní střed $[x_i, y_i]$, výpočet souřadnic a poloměru se provede z původních čtverců čtyřstromu: $x_i = \frac{a_i}{2}$, $y_i = \frac{b_i}{2}$, kde a_i a b_i jsou souřadnice levého dolního rohu čtverce, poloměr r_i, $r_i = \sqrt{{d_i}^2 + {d_i}^2}$, kde d je délka strany čtverce. Rotace u kružnice se poté dá uskutečnit pomocí goniometrických funkcí pouze na samotný střed kružnice, tvar a poloměr poté samozřejmě zůstává stejný. Použitím kružnice místo čtverce je získán kompromis mezi výpočetní složitostí a zachováním tvaru při rotacích. Poté pomocí knihovní funkce je zjištěno, které kružnice jsou uvnitř objektu a které mimo, tím je získána přibližná reprezentaci objektu v kružnicích. Myšlenka zpracování objektu přes čtyřstrom přímo pochází z článku "Packing Irregular-Shaped Objects for 3D Printing"od Stefana Edelkampa a Paula Wicherna [1].



Obrázek 2.1 Stl objekt Chicken.stl před zpracováním, pohled ze shora



Obrázek 2.2 Stl objekt Chicken.stl při zpracování čtyřstromu na maximální hloubku 2



Obrázek 2.3 Stl objekt Chicken.stl při zpracování čtyřstromu na maximální hloubku 6

2.1.2 Formulace problému s omezeními

S připravenými objekty do vhodné 2D reprezentace je čas na konečnou formulaci problému s omezeními a následné implementace formulací v řešících OR-Tools a Gecode. Pro každý objekt jakožto výstup z problému s omezeními je požadována poloha a otočení, to pro 2D prostor znamená, souřadnice x, y a úhel ϕ. Práce počítá se dvěma vzájemně podobnými, ale trošku různými formulacemi, které se následně implementují, důvod jejich počtu je rozdílnost v možnostech a funkčnostech řešičů Gecode a OR-Tools. Pro všechny formulace ale platí následující konstanty:

- n: Počet objektů.
- V_x, V_y: Délky hran tiskové plochy, tisková plocha se bere pouze jenom s tvarem obdélníku.
- S_t: Střed tiskové plochy, bod ke kterému se vzdálenost objektů minimalizuje.
- $pocetBituKombinaciUhlu$: Počet bitů, do kterých se zakóduje počet rotací, nabývá hodnot od 0 do 7, tedy, např. pokud $pocetKombinaciUhlu = 3$, poté je počet rotací 2^3, pokud $pocetBituKombinaciUhlu = 6$, poté je počet rotací 2^6, je to z důvodu použití reifikace při implementaci rotací.
- $meritko$ Jelikož je problém spojitý a řešič OR-Tools neumí znázornit spojité proměnné, je důležité vzdálenosti objektů, tiskové plochy, středu, přenásobit měřítkem.
- $uhelOtoceni$: Nejmenší jednotka otočení, odvozuje se od parametru $pocetBituKombinaciUhlu$, výpočet je poté $uhelOtoceni = 360/2^{p}ocetBituKombinaciUhlu$, jednotka je spojitá, ale pro θ_i, neboli i-tý úhel se používá zaokrouhlení dolů, např. $pocetBituKombinaciUhlu = 4$, počet rotací je tedy 4, $uhelOtoceni = 360/16$ a $i = 5$ bude výpočet $\theta_i = \theta_5 = \lfloor 5 * uhelOtoceni \rfloor = \lfloor 5 * 22,5 \rfloor = \lfloor 112,5 \rfloor = 112$.

Implementace často používá 2 operace
Posunutí bodu, kolem vlastního středového bodu, do konkrétního bodu je operace posunu kružnice, která je součástí objektu, který má svůj vlastní střed, vypočteme vektor, který potřebujeme přičíst středu kružnici, tak že posuneme střed objektu do konkrétního bodu. a stejný vektor přičteme ke středu kružnice.

Otočení se provadí za pomoci rotační matice R_θ [25]. Rotační matice je transformační matice, která se používá pro rotaci objektů v euklidovském prostoru. Ve 2D má rotační matice následující tvar:

$$R_\theta = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix}$$

Výpočet nové pozice středu kružnice po otočení úhlem θ je poté prosté maticové násobení rotační matice a bodu. Např. bod X', se souřadnicema x' a y', po otočení bodu x, se sořadnicema x a y úhlem θ, je následující

$$X' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix} = \begin{matrix} x' = x * cos(\theta) + (-y * (sin(\theta))) \\ y' = x * sin(\theta) + y * cos(\theta) \end{matrix}$$

Jelikož se ale body kružnice neotáčí kolem bodu $(0, 0)$, ale kolem středu objektu, musí se střed objektu posunout do bodu $(0, 0)$, o stejný posun posunout střed kružnice, následně ho otočit pomocí rotační matice, poté objekt posunout do středu tiskové plochy a o stejný posun se posune i střed kružnice. Tím bude objekt otočen a zároveň bude na středu tiskové plochy. Například pro bod kružnici se středovým bodem $X = (10, 15)$, kde střed objektu je $(8, 10)$, tedy posunutí do bodu $(0, 0)$ je $(-8, -10)$, středem tiskové plochy $(20, 20)$ a úhel otočení $\theta = 90°$, vyjde bod X' následovně:

$$X' = \begin{bmatrix} cos(90°) & -sin(90°) \\ sin(90°) & cos(90°) \end{bmatrix} * \left(\begin{bmatrix} 10 \\ 15 \end{bmatrix} + \begin{bmatrix} -8 \\ -10 \end{bmatrix} \right) + \begin{bmatrix} 20 \\ 20 \end{bmatrix}$$

Víme, že

$$cos(90°) = 0, sin(90°) = 1$$

Výsledek operací posunutí, otočení a posunutí objektu do středové plochy pro kružnici je následující:

$$X' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} ((10 - 8) * 0 + (-(15 - 10) * 1)) + 20, \\ ((10 - 8) * 1 + (-(15 - 10) * 0)) + 20 \end{bmatrix} = \begin{bmatrix} 15 \\ 22 \end{bmatrix}$$

Formálně se operace posunutí bodu, kolem vlastního středového bodu, do konkrétního bodu zadefinuje následovně. $bodX'$, bod kam bod X bude posunut je definován následovně:

$$posunX(X, Y, Z) = x_X + x_Y - x_Z,$$
$$posunY(X, Y, Z) = y_X + y_Y - y_X$$

$$X' = (x', y') = (posunX(X, Y, Z), posunY(X, Y, Z)) = (x_X + x_Y - x_Z, y_X + y_Y - y_X)$$

Bod X je středový bod kružnice, který bude posunut, Y je bod, kam bude bod X posunut a Z je středový bod objektu, jehož je kružnice, se středovým bodem, X součástí. Pro operaci $posunX(X, Y, Z)$ platí, že x_X, x_Y a x_Z značí souřadnici x bodu uvedený v indexu, stejně to je i pro operaci $posunY(X, Y, Z)$.

Operace otočení pro bod X', z bodu X, se souřadnicema (x_X, y_X) se zadefinuje následovně:

$$otoceniX(\theta, X) = x_X * cos(\theta) - y_X * sin(\theta)$$
$$otoceniY(\theta, X) = x_X * sin(\theta) + y_X * cos(\theta)$$

$$X' = (x', y') = (otoceniX(\theta, X), otoceniY(\theta, X)) = (x * cos(\theta) - y * sin(\theta), x * sin(\theta) + y * cos(\theta))$$

Důvod rozdělení operací na dvě, je že každý objekt bude v problému s omezeními posunutí od středu tiskové plochy posunut o vektor (x_i, y_i), kde i značí i-tý objekt a výpočty tak budou na jednotlivých souřadnicích kružnic, která patří k objektu, ne na celých bodech středů kružnic.

2.1.2.1 První formulace problému s omezeními

Jelikož se řešiče OR-Tools a Gecode, ale i spousta dalších, liší v jejich možnostech, tak je možnost problém s omezeními formulovat několikrát, druhá formulace je právě myšlena pro řešič OR-Tools, kde formulace používá funkčností, které Gecode naobsahuje. První formulace je společná pro Gecode a OR-Tools, aby šlo výsledky lépe porovnat. Formalizace problému s omezeními je zavedení trojice X, D, C, neboli množina proměnných, domény proměnných a omezení na proměnné a jejich domény.

Množina proměnných X
$X = \{x_1, y_1, \Phi_1, x_2, y_2, \Phi_2, ..., x_n, y_n, \Phi_n\}$, kde x_i a y_i značí posun i-tého objektu od středu tiskové plochy a množina Φ_i obsahuje proměnné sloužící pro rotaci i-tého objektu. Množiny proměnných sloužící pro rotaci jsou typu bool, neboli typu int, ale nabývají hodnot pouze dvou hodnot a to 0, nebo 1. Jejich počet se rovná hodnotě parametru $2^{pocetBituKombinaciUhlu}$. Pokud $pocetBituKombinaciUhlu = 5$, poté $\Phi_i = \{\phi_1, \phi_2, \phi_3, ..., \phi_{2^5-1}, \phi_{2^5}\}$, kde i značí i-tý objekt.

Domény proměnných D
Domény proměnných sloužící pro posun, záléží na velikosti tiskové plochy 3D tiskárny, proměnné sloužící pro posun po ose x mají doménu všechna celá čísla od $\lfloor -V_x/2 \rfloor * meritko$ do $\lfloor V_x/2 \rfloor * meritko$ a doména proměnných sloužící pro posun po ose y jsou všechna celá čísla od $\lfloor -V_y/2 \rfloor * meritko$ do $\lfloor V_y/2 \rfloor * meritko$. Jak již bylo řečeno u proměnných, doména proměnných sloužící pro otočení jsou pouze dvě hodnoty, 0 a 1, neboli $false$ a $true$, protože se jedná o proměnný typu bool. Formálně tedy platí $D = \{D_{x_1}, D_{y_1}, D_{\Phi_1}, ..., D_{x_n}, D_{y_n}, D_{\Phi_n}\}$. D_{x_i} a D_{y_i} jsou domény proměnných, D_{Φ_i} je množina domén, její velikost záleží na parametru $pocetBituKombinaciUhlu$, velikost množiny se rovná $2^{pocetBituKombinaciUhlu}$.

- Pro D_{x_i} platí $D_{x_i} = \{[\lfloor -V_x/2 \rfloor, \lfloor V_x/2 \rfloor], [\lfloor -V_x/2 \rfloor, \lfloor V_x/2 \rfloor]\}$.
- Pro D_{y_i} platí $D_{y_i} = \{[\lfloor -V_y/2 \rfloor, \lfloor V_y/2 \rfloor], [\lfloor -V_y/2 \rfloor, \lfloor V_y/2 \rfloor]\}$.
- Pro D_{Φ_i} platí $D_{\Phi_i} = \{[0,1], ..., [0,1]\}$, kde počet intervalů domén proměnných se rovná parametru $2^{pocetBituKombinaciUhlu}$.

Nakonec z trojice zbývá formalizace omezení $C = \{\forall i, j, i >= 0 \wedge i < n \wedge j > i$: objekty i a j se neprotínají $\}$. Po inicializaci proměnných s jejich doménami. Každý objekt se skládá z kružnic, pro každou kružnici X se uloží výsledek otočení do souřadnic x', y', zvlášť pro souřadnici na ose x a y, neboli se využijí zřetězené operace:

- $x' = posunX(otoceniX(\theta, posunX(X, (0,0), stredObjektu)), S_t, (0,0))$,
- $y' = posunY(otoceniY(\theta, posunY(X, (0,0), stredObjektu)), S_t, (0,0))$.

Proměnná x' by se tedy omezila na všechny možný otočení, ale k řešení problému je potřeba omezit proměnnou pouze na jedno otočení, na porovnání s dalšími objekty. Toho se docílí pomocí poloviční reifikace, kdy pomocí proměnných určených pro rotaci Φ_i. Proměnné v Φ_i umožňují podmínit propagaci do x'. Neboli pokud je Φ_{i_j} rovno hodnotě 1, potom $x' = posunX(otoceniX(j * uhelOtoceni, posunX(X, (0,0), stredObjektu)), S_t, (0,0))$, neboli na každé Φ_i je omezení, že pouze jedna a právě jedna proměnná je rovna hodnotě 1, a právě tato proměnná určuje, který otočení se propaguje na otočení a posun středů kružnic objektu.

Nyní už je čas posouvat objekty a jejich kružnice ze středu tiskové plochy, tak aby se neprotínaly, ale zároveň i tak, aby se vešly na tiskovou plochu, tedy objekt nepřesahuje její rozměry. To se dá zajistit omezeními, kde je každá kružnice omezena, tak aby její poloměr nepřesahoval tiskovou plochu do žádného směru, tedy pokud má tisková plocha, obdélníkového tvaru, rozměry V_x, V_y, tak kružnice jsou omezeny, tak aby součet posunutí kružnice, středu kružnice a jejího poloměru nebyl větší jak rozměry tiskové plochy, nebo menší jak 0. Tím se zmenší počet možností, které řešič bude muset řešit u následujícího omezení neprotínání objektů.

Neprotínání objektů je poslední omezení, mimo minimalizace vzdálenosti objektů ke středu tiskové plochy, který je potřeba zformulovat. Neprotínání objektů znamená, že žádný dvě kružnice, který nejsou součástí stejného objektu se nesmí protínat, neboli překrývat. Jelikož bylo zavedeno, že objekt se skládá z kružnic, tak je to relativně intuitivní formulace, prostě se porovná vzdálenost středů kružnic se součtem jejich poloměrů. Pro vzdálenost středů kružnic je využita úprava euklidovské vzdálenosti, pro dvě kružnice k_1, se středem o souřadnicích x_1, y_1 a poloměrem r_1, a k_2 se středem x_2, y_2 a poloměrem r_2, poté $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$, jelikož není pro problém dále potřeba výsledek této operace, pouze porovnání, tak je možnost celou rovnici umocnit na druhou mocninu a tím se zbavit operace odmocnění, výsledek takové úpravy je $(x_1 - x_2)^2 + (y_1 - y_2)^2$, druhá strana nerovnice je součet poloměrů, $r_1 + r_2$, ale jelikož se levá strana nerovnice umocnila na druhou, tak je potřeba umocnit i pravou stranu, neboli $(r_1 + r_2)^2 = ({r_1}^2 + r_1 * r_2 + {r_2}^2)$, ale jelikož je poloměr z pohledu problému s omezeními pouze konstanta, tak umocnění navíc ničemu nevadí. Výsledný tvar nerovnice, neboli omezení je následující $(x_1 - x_2)^2 + (y_1 - y_2)^2 \geq (r_1 + r_2)^2$. Pokud se toto omezení rozšíří na všechny kružnice, které nejsou součástí stejného objektu, tak je tím omezení žádné 2 objekty se neprotínají vytvořeno.

Poslední část formulace je formulovat minimalizační proměnnou, *cil*, která značí manhattanskou vzdálenost posunu proměnných X_i a Y_i. $cil \geq |X_i| + |Y_i|$, jelikož cílem je proměnnou *cil* minimalizovat, tak proměnná *cil* nabývá nejmenší možné hodnoty, která je větší jak největší vzdálenost objektu od středu tiskové plochy.

2.1.2.2 Druhá formulace problému s omezeními

Zatímco první fomulace je hlavně myšlena pro srovnání řešičů OR-Tools a Gecode, myšlenka druhé formulace je zmenšit počet proměnných, jelikož je proměnných určených pro rotace hodně, myšlenka formulace je jejich počet zmenšit a to zakódováním reifikace do kombinací proměnných. Gecode tuto formulaci jednoduše neumožňuje a tak je poloviční reifikace formulována pomocí tolika počtu proměnných typu bool, kolik je počet rotací. Tato formulace je určena pro implementaci pro řešič OR-Tools, použití kombinací místo všech proměnných je jediný rozdíl mezi druhou a první formulací, tato část pro druhou formulaci je určena pouze pro vypsání těch částí formulace, které se liší od té první formulace.

Množina proměnných X
$X = \{x_1, y_1, \Phi_1, x_2, y_2, \Phi_2, ..., x_n, y_n, \Phi_n\}$, kde x_i a y_i značí posun i-tého objektu od středu tiskové plochy a množina Φ_i obsahuje proměnné sloužící pro rotaci i-tého objektu. Množiny proměnných sloužící pro rotaci jsou typu bool, neboli typu int, ale nabývají hodnot pouze dvou hodnot a to 0, nebo 1. Jejich počet se rovná hodnotě parametru $pocetBituKombinaciUhlu$. Pokud $pocetBituKombinaciUhlu = 5$, poté $\Phi_i = \{\phi_1, \phi_2, \phi_3, \phi_4, \phi_5\}$, kde i značí i-tý objekt.

Domény proměnných D
Domény proměnných sloužící pro posun, záléží na velikosti tiskové plochy tiskárny, proměnné sloužící pro posun po ose x mají doménu všechna celá čísla od $\lfloor -V_x/2 \rfloor * meritko$ do $\lfloor V_x/2 \rfloor * meritko$ a doména proměnných sloužící pro posun po ose y jsou všechna celá čísla od $\lfloor -V_y/2 \rfloor * meritko$ do $\lfloor V_y/2 \rfloor * meritko$. Jak již bylo řečeno u proměnných, doména proměnných sloužící pro otočení jsou pouze dvě hodnoty, 0 a 1, neboli $false$ a $true$, protože se jedná o proměnný typu bool. Formálně tedy platí $D = \{D_{x_1}, D_{y_1}, D_{\Phi_1}, ..., D_{x_n}, D_{y_n}, D_{\Phi_n}\}$. D_{x_i} a D_{y_i} jsou domény proměnných, D_{Φ_i} je množina domén, její velikost záleží na parametru $pocetBituKombinaciUhlu$.

- Pro D_{x_i} platí $D_{x_i} = \{[\lfloor -V_x/2 \rfloor, \lfloor V_x/2 \rfloor], [\lfloor -V_x/2 \rfloor, \lfloor V_x/2 \rfloor]\}$.
- Pro D_{y_i} platí $D_{y_i} = \{[\lfloor -V_y/2 \rfloor, \lfloor V_y/2 \rfloor], [\lfloor -V_y/2 \rfloor, \lfloor V_y/2 \rfloor]\}$.
- Pro D_{Φ_i} platí $D_{\Phi_i} = \{[0, 1], ..., [0, 1]\}$, kde počet intervalů domén proměnných se rovná parametru $pocetBituKombinaciUhlu$.

Nakonec z trojice zbývá formalizace omezení $C = \{\forall i, j, i >= 0 \wedge i < n \wedge j > i$: objekty i a j se neprotínají $\}$. Po inicializaci proměnných s jejich doménami. Každý objekt se skládá z kružnic, pro každou kružnici X si uložíme výsledek otočení do souřadnic x', y', zvlášť pro souřadnici na ose x a y neboli operace

- $x' = posunX(otoceniX(\theta, posunX(X, (0,0), stredObjektu)), S_t, (0,0))$,
- $y' = posunY(otoceniY(\theta, posunY(X, (0,0), stredObjektu)), S_t, (0,0))$.

Do proměnné x' se tedy přiřadí všechny možný otočení, ale k řešení problému je potřeba přiřazení pouze jednoho otočení, na porovnání s dalšími objekty. Toho se docílí pomocí poloviční reifikace, kdy pomocí proměnných určených na rotaci Φ_i. Kombinace proměnných v Φ_i umožňuje podmínit přiřazení do x'. Důvod použití kombinací a ne pro každou rotaci vlastní proměnnou je snaha co nejvíc limitovat počet proměnných.

Příklad přiřazení úhlu, samotnou hodnotu proměnných v Φ_i zajistí řešič, ale např. pokud $pocetBituKombinaciUhlu = 3$, tak jsou k dispozici tři bity na zakódování, kombinace vygenerovaných hodnot řešičem $\Phi_i = \{0,0,0\}$ znázorňuje úhel 0°, neboli hodnoty v Φ_i jsou brány jako bity a jsou převedeny na číslo v desítkové soustavě ($\Phi_i = \{0,0,0\} = 0$, $\Phi_i = \{1,1,0\} = 6$), takto je možné podmínit přiřazení výsledku pomocí proměnných typu bool.

Jelikož počet rotací, neboli úhlů na kterých se aplikují operace *posun* a *otoceni* je stejně jako počet všech kombinací hodnot v Φ_i pro i-tý objekt. Tím je získán v průběhu řešení pouze jedno otočení objektu a jeho posunutí do středu tiskové plochy, také je tím omezeno otočení na vygenerovanou kombinaci v proměnných Φ_i.
Formálním zápisem:
Kombinace Φ_{i_k}, kde k je číslo, který se vytvoří zapsáním hodnot do desítkové soustavy (např. $\Phi_{i_k} = \{0,1,1,0,1\} \Rightarrow k = 01101_2 = 13$) a i znázorňuje i-tou kružnici v objektu.

$$\Phi_{i_k} \Rightarrow x' = posunX(otoceniX(k * uhelOtoceni, posunX(X, (0,0), stredObjektu)), S_t, (0,0))$$

Všechno ostatní u druhé formulace je stejné jako u první formulace.

2.2 Implementace

Tato část náleží pro implementaci formulovaných problémů s omezeními. První formulace je určena pro porovnání výásledků mezi Gecode a OR-Tools, druhá formulace je pak myšlena pouze pro OR-Tools, kde je podpora funkcí pro reifikaci přes několik proměnných, ne jen přes jednu.

2.2.1 OR-Tools implementace

OR-Tools nepotřebuje implementovat žádný prostředí, prostě se založí objekt pro ukládání proměnných, jejich domén a omezení nad nima a ten se následně přendá vyhledávači, který může, ale samozřejmě nemusí, mít parametry, jako například počet vláken vyhledávače, lze v něm i limitovat čas výpočtu řešiče. Následující část je rozdělena na tyto části, kde bude kratší vysvětlení k jednotlivým částem implementace v C++:

- parametry funkce,
- inicializace,
- reifikační funkce,
- omezení otočení,
- omezení vzhledem k velikosti tiskové plochy,

- omezení žádné 2 objekty se neprotínají,
- minimalizace vzdálenosti k prostředku tiskové plochy,
- příprava řešiče, vypsání výsledku a uložení nejlepšího výsledku.

2.2.1.1 parametry funkce

Ještě před samotným vkládáním proměnných, domén a jejich omezení do řešiče je důležité poukázat na inicializaci. V OR-Tools to je funkce, ale nemusí být, řešič požaduje pouze vytvoření a předání vyhledávači, s názvem minimizeCenterOrTools(), možnosti, funkce OR-Tools a řešič (v našem případě "CP-SAT Solver") je obsažen v namespace **operations_research::sat** a samotná funkce ***minimizeCenterOrTools()*** má následující vstupní parametry:

- **objects**
  Pole o velikosti n připravených objektů.
- **center**
  Souřadnice S_t, kde se nachází střed tiskové plochy v milimetrech.
- **solutions**
  Pole pro uložení nejlepšího řešení.
- **printingAreaSize**
  Párová proměnná, která obsahuje souřadnice V_x, V_y, neboli velikost tiskové plochy.
- **settings**
  nastavení pro řešič, struktura *customSettings* obsahuje omezení délky výpočtu a počet vláken pro řešič.
- **anglesToCodeSize**
  Parametr zastupuje *pocetBituKombinaciUhlu* a určuje počet rotací objektů.
- **ratio**
  Zastupuje *meritko*, parametr slouží pro škálování výpočtů.

```
void minimizeCenterBoolOrTools(vector <objectInfo> &objects, pair<int, int> center,
    vector<pair<double, pair<int, int>>>& solutions, pair<int, int> printingAreaSize,
    customSettings &settings, double anglesToCodeSize, int ratio);
```

Výpis kódu 2.1 Funkce minimizeCenterBoolOrTools implementace pro OR-Tools

2.2.1.2 Inicializace

V inicializaci se inicializují počáteční proměnné, jako je posun po ose x a y a proměnné určené pro otáčení, ale i konstanty a hodnoty důležité pro budoucí omezení. Součástí této části je připravení úhlu otočení, konvertovat úhly na radiány a hodnoty výsledků otočení $sin()$ a $cos()$ uložit do pole, kde se dále využijí u omezení pro rotaci. Následuje inicializace proměnných $intVarsX$ sloužící pro posun po ose x, $intVarsY$ sloužící pro posun po ose y a $boolVars$ sloužící pro rotaci objektů. $intMultiplicationVarsX$ a $intMultiplicationVarsY$ slouží jako mezivýsledek výpočtu omezení otočení. Nakonec inicializace se zavede naše první omezení implementace a to, že pouze jedna a právě jedna proměnná v $boolVars[i]$ může mít hodnotu 1, neboli $true$, ostatní mají hodnotu 0, neboli $false$.

```
CpModelBuilder cp_model;
int printingAreaX = printingAreaSize.first/2;
int printingAreaY = printingAreaSize.second/2;
const double pi = std::acos(-1);
std::vector<double> sineValues;
std::vector<double> cosValues;
int angles = (int)pow(2,anglesToCodeSize);
double angleMultiplicator = 360./angles;
for (int i = 0; i < angles; i++) {
    sineValues.push_back(sin(double(int(i * angleMultiplicator)) * pi / 180));
    cosValues.push_back(cos(double(int(i * angleMultiplicator)) * pi / 180));
}
std::vector<std::vector<BoolVar>> boolVars;
std::vector<IntVar> intVarsX;
std::vector<IntVar> intVarsY;
std::vector<std::vector<IntVar>> intMultiplicationVarsX;
std::vector<std::vector<IntVar>> intMultiplicationVarsY;

for (int i = 0; i < objects.size(); i++) {
    boolVars.emplace_back();
    intVarsX.push_back(cp_model.NewIntVar({-printingAreaX*ratio, printingAreaX*ratio}));
    intVarsY.push_back(cp_model.NewIntVar({-printingAreaY*ratio, printingAreaY*ratio}));
    for (int j = 0; j < angles; j++) {
        boolVars[i].push_back(cp_model.NewBoolVar());
    }
}
for (int i = 0; i < objects.size(); i++) {
    cp_model.AddExactlyOne(boolVars[i]);
}
.
.
```

Výpis kódu 2.2 Inicializace problému s omezeními v OR-Tools

2.2.1.3 Omezení otočení

Po vytvoření proměnných pro souřadnice objektu a otočení. Je na čase uložit první výsledky po omezení AddEquality, to je znázorněno v kódu pomocí funkce **addOnlyIfRestriction()**, její parametry jsou:

- cp_model: reference na samotný řešič,
- intVar: proměnná problému s omezeními, do které se bude propagovat hodnota result,
- result: výsledek pro rotaci jedné kružnice v objektu,
- caseNumber: v pořadí jaký úhel se má uložit,
- boolVars: reference na proměnné pro rotaci, podle čísla caseNumber se určí, která proměnná bude použita pro poloviční reifikaci. Např. caseNumber 77 znamená 77. otočení podmínit na poloviční reifikaci na boolVars[77], tedy pokud $boolVars[77] == 1$, poté se propaguje $intVar == result$.

```
void addOnlyIfRestrictionBool(CpModelBuilder &cp_model, IntVar &intVar,
                              long int result, int caseNumber,
                              std::vector <BoolVar> &boolVars) {
    cp_model.AddEquality(intVar, result).OnlyEnforceIf(boolVars[caseNumber]);
}
```

Výpis kódu 2.3 Reifikace rotace pro střed jedné kružnice

```
for (int i = 0; i < objects.size(); i++) {
    intMultiplicationVarsX.emplace_back();
    intMultiplicationVarsY.emplace_back();
    for(int j = 0; j < objects[i].m_Circles.size(); j++) {
        intMultiplicationVarsX[i].push_back(
            cp_model.NewIntVar({0, printingAreaX*2*ratio}));
        intMultiplicationVarsY[i].push_back(
            cp_model.NewIntVar({0, printingAreaX*2*ratio}));
        for (int k = 0; k < angles; k++) {
            long long int resultX = (long long int) (ratio * (
                cosValues[k] * shiftPointToCenter(objects[i].m_Circles[j].second.first,
                    0, CGAL::to_double(objects[i].m_CenterPoint.x())) -
                sineValues[k] * shiftPointToCenter(objects[i].m_Circles[j].second.second,
                    0, CGAL::to_double(objects[i].m_CenterPoint.y())) +
                center.first));
            long long int resultY = (long long int) (ratio * (
                cosValues[k] * shiftPointToCenter(objects[i].m_Circles[j].second.second,
                    0, CGAL::to_double(objects[i].m_CenterPoint.y())) +
                sineValues[k] * shiftPointToCenter(objects[i].m_Circles[j].second.first,
                    0, CGAL::to_double(objects[i].m_CenterPoint.x())) +
                center.second));
            addOnlyIfRestrictionBool(cp_model, intMultiplicationVarsX[i].back(),
                                     resultX, k, boolVars[i]);
            addOnlyIfRestrictionBool(cp_model, intMultiplicationVarsY[i].back(),
                                     resultY, k, boolVars[i]);
        }
    }
}
```

Výpis kódu 2.4 Omezení otočení kružnice objektu

Výsledek rotace jedné kružnice jednoho objektu se uloží do pole pro mezivýsledky $intMultiplicationVarsX$ pro souřadnici na ose x a $intMultiplicationVarsY$ pro souřadnici na ose y, samotný výpočet je poté stejný jako ve formulaci, kde funkce shiftPointTocenter() funguje stejně jako $posunX$, nebo $posunY$. Výsledek je ještě přenásoben měřítkem $ratio$, aby byla dostatečná přesnost. Pomocí funkce addOnlyIfRestrictionBool() je výsledek podmíněn pomocí poloviční reifikace na propagaci pouze jedné hodnoty.

2.2.1.4 Omezení neprotínání

Tato část implementace se konečně dostává na implementaci omezení neprotínání. Implementace neprotínání jde přes čtyři cykly. první cyklus jde přes všechny objekty, druhý cyklus jde poté přes všechny objekty, se kterýma ještě i-tý objekt nemá vytvořené omezení neprotnutí, třetí

```
for(int i = 0; i < intMultiplicationVarsX.size(); i++){
  for(int j = 0; j < intMultiplicationVarsX[i].size(); j++){
    cp_model.AddLessThan(intVarsX[i] + intMultiplicationVarsX[i][j] +
    (long int)(ratio * objects[i].m_Circles[j].first), printingAreaX*2*ratio);
    cp_model.AddLessThan(intVarsY[i] + intMultiplicationVarsY[i][j] +
    (long int)(ratio * objects[i].m_Circles[j].first), printingAreaY*2*ratio);
    cp_model.AddGreaterThan(intVarsX[i] + intMultiplicationVarsX[i][j] -
    (long int)(ratio * objects[i].m_Circles[j].first), 0);
    cp_model.AddGreaterThan(intVarsY[i] + intMultiplicationVarsY[i][j] -
    (long int)(ratio * objects[i].m_Circles[j].first), 0);
  }
}
```

Výpis kódu 2.5 Omezení vzhledem k velikosti tiskové plochy

cyklus jde přes všechny kružnice i-tého objektu a poslední cyklus pak přes všechny kružnice j-tého objektu. OR-Tools neumožňuje jednoduché násobenní mezi proměnnými, jediná možnost jak vynásobit dvě proměnné je použití omezení AddMultiplicationEquality(), tato funkce jako parametry přijímá pouze samotné proměnné, proto se prvně ukládá výsledek $x1 - x_2$, a až poté se výsledek umocňuje, najednou to nelze. Nakonec se vypočíta druhá mocnina součtu poloměrů, která pro řešič funguje jako konstanta, poté se už aplikuje samotný omezení, že žádné dvě kružnice z různých objektů se neprotínají.

```
for (int i = 0; i < intMultiplicationVarsX.size(); i++) {
  for (int j = i + 1; j < intMultiplicationVarsX.size(); j++) {
    for(int k = 0; k < intMultiplicationVarsX[i].size(); k++) {
      for(int l = 0; l < intMultiplicationVarsX[j].size(); l++) {
        const IntVar tmpEqualX = cp_model.NewIntVar(
            {-printingAreaX*2*ratio, printingAreaX*2*ratio});
        const IntVar tmpEqualY = cp_model.NewIntVar(
            {-printingAreaY*2*ratio, printingAreaY*2*ratio});
        const IntVar tmpMultiplicationX = cp_model.NewIntVar(
            {0, printingAreaX*printingAreaX*ratio*ratio});
        const IntVar tmpMultiplicationY = cp_model.NewIntVar(
            {0, printingAreaY*printingAreaY*ratio*ratio});
        cp_model.AddEquality(tmpEqualX,
            intVarsX[i]+intMultiplicationVarsX[i][k]-
            intVarsX[j+intMultiplicationVarsX[j][l]);
        cp_model.AddEquality(tmpEqualY,
            intVarsY[i]+intMultiplicationVarsY[i][k]-
            intVarsY[j+intMultiplicationVarsY[j][l]);
        cp_model.AddMultiplicationEquality(tmpMultiplicationX, tmpEqualX, tmpEqualX);
        cp_model.AddMultiplicationEquality(tmpMultiplicationY, tmpEqualY, tmpEqualY);
        long long int radius = (long long int) (
            (ratio * objects[i].m_Circles[k].first +
            ratio * objects[j].m_Circles[l].first) *
            (ratio * objects[i].m_Circles[k].first +
            ratio * objects[j].m_Circles[l].first));
        cp_model.AddGreaterThan(tmpMultiplicationX + tmpMultiplicationY, radius);
      }
    }
  }
}
```

Výpis kódu 2.6 Omezení protnutí 2 kružnic, 2 různých objektů o rotaci

2.2.1.5 Minimalizace vzdálenosti k prostředku tiskové plochy

Jeden z cílů této práce je právě snaha o minimalizaci ke středu tiskové plochy, zatímco doteď se požadovalo po řešiči pouze nějaký výsledek, u kterého se nemohla porovnat jeho kvalita s jiným výsledkem, právě minimalizace je to jisté kritérium, které umožňuje porovnání, ať už výsledek v OR-Tools, tak i výsledek v Gecode, ale popřípadě i v jiných řešičích. Minimalizace je formulována pomocí manhattanské metriky, neboli vzdálenost je součet absolutních hodnot souřadnic posunutí objektů na tiskové ploše. Tato skutečnost je pro řešič napsána uvnitř cyklu. konečný příkaz $cp\_model.Minimize(objectiveDistance)$ je pouze příkaz pro řešič, že má minimalizovat hodnotu proměnné $objectiveDistance$.

```
const IntVar objectiveDistance = cp_model.NewIntVar({0, printingAreaX*2*ratio});
for(int i = 0; i < intVarsX.size(); i++){
    const IntVar absX = cp_model.NewIntVar({0, printingAreaX*ratio});
    const IntVar absY = cp_model.NewIntVar({0, printingAreaY*ratio});
    cp_model.AddAbsEquality(absX, intVarsX[i]);
    cp_model.AddAbsEquality(absY, intVarsY[i]);
    cp_model.AddGreaterOrEqual(objectiveDistance, absX + absY);
}
cp_model.Minimize(objectiveDistance);
```

Výpis kódu 2.7 Minimalizace vzdálenosti k prostředku tiskové plochy

2.2.1.6 Příprava řešiče a uložení výsledku

Těsně před spuštěním řešiče je ještě důležité připravit řešič, to znamená zadání parametrů jako je omezení délky výpočtu, nebo nastavení počtu vláken. Je tu i možnost říct řešiči jakou heuristiku má použít. Příklad toho je samotný příkaz pro minimalizaci $cp\_model.Minimize()$. Hned po přípravě je ještě důležité dát řešiči lambda funkci, co má dělat, když najde nějaký výsledek. Jelikož je možnost použít lambda funkci, s výsledkem lze dělat relativně cokoliv, cokoliv co umožňuje C++, ale zároveň je dobré limitovat čas ztrávený zpracováním výsledku, protože je tím zpomalován řešič. V implementaci se výsledek vypíše a nejlepší výsledek se uloží. Hned po této lambda funkci se pomocí příkazu $constCpSolverResponseresponse = SolveCpModel(cp\_model.Build(), \&model)$; zapne řešič. Výsledek je poté odsazení, na osách x a y, středů objektů od středu tiskové plochy a jejich otočení, dále výsledek minimalizace ke středu a čas od začátku výpočtu.

```
Model model;
SatParameters parameters;
parameters.set_max_time_in_seconds(settings.time);
parameters.set_num_search_workers(settings.threads);
model.Add(NewSatParameters(parameters));
model.Add(NewFeasibleSolutionObserver([&](const CpSolverResponse &r) {
for(int i = 0; i < intVarsX.size(); i++){
    int angle = 0;
    for(int j = 0; j < boolVars[i].size(); j++){
        if(SolutionBooleanValue(r, boolVars[i][j])){
            angle = (int)(j*angleMultiplicator);
            break;
        }
    }
    LOG(INFO) << "object: " << i << ", x = " << SolutionIntegerValue(r, intVarsX[i])
              << ", y = " << SolutionIntegerValue(r, intVarsY[i])
              << ", angle = " << angle;
    if(SolutionIntegerValue(r,objectiveDistance) < g_BestSolution){
        if(g_BestSolution > printingAreaSize.first*ratio){
            solutions.emplace_back((double)(angle),
                make_pair(SolutionIntegerValue(r, intVarsX[i]),
                          SolutionIntegerValue(r, intVarsY[i])));
        }
        else{
            solutions[i].first = (double)(angle);
            solutions[i].second =
                make_pair(SolutionIntegerValue(r, intVarsX[i]),
                          SolutionIntegerValue(r, intVarsY[i]));
        }
    }
}
if(SolutionIntegerValue(r,objectiveDistance) < g_BestSolution) {
    g_BestSolution = SolutionIntegerValue(r, objectiveDistance);
}
LOG(INFO) << "Minimized value = " << SolutionIntegerValue(r, objectiveDistance);
LOG(INFO) << "Time of solution since start: " <<
    std::chrono::duration<double>(
    std::chrono::system_clock::now() - g_WallTimeStamp).count();
LOG(INFO) << "////////////////////////////////////////////////////";
}));
const CpSolverResponse response = SolveCpModel(cp_model.Build(), &model);
LOG(INFO) << CpSolverResponseStats(response);
LOG(INFO) << response.solve_log();
```

Výpis kódu 2.8 Příprava řešiče, vypsání výsledku a uložení nejlepšího výsledku

2.2.2 Gecode implementace s použitím bool proměnných

Následující část je pro implementaci v Gecode za pomoci bool proměnných pro rotaci. Hlavní myšlenka použití bool proměnných, i když už to není potřeba je zachovat co největší podobnost implementaci v OR-Tools, pro porovnání výsledků. Základem Gecode je implementace prostředí,

neboli třídu dědící z IntMinimizeScript. Třída slouží pro minimalizaci hodnot typu IntVar, což je proměnná problému s omezeními typu int. Třída dále musí mít implementovaný následující metody:

- *minimizeCenterBoolCSP*(*const Options &opt*) konstruktor s parametrem Options, kde můžeme nastavovat některé parametry řešiče
- *minimizeCenterBoolCSP*(*minimizeCenterBoolCSP &s*) kopírující konstruktor
- Metoda *virtual Space *copy*(*void*); vrací novou instanci naší třídy
- Metoda *virtual void print*(*void*) *const*; slouží k zpracování výsledku, nebo jeho vypsání
- Metoda *virtual IntVar cost*() *const*; vrací minimalizovanou hodnotu, která musí být přiřazena, to znamená, v rámci Gecode, že má pouze jenom jednu hodnotu, není to interval hodnot.

Všechny metody, až na samotný konstruktor s parametrem *Options* a print mají víceméně danou podobu, metoda *cost*() pouze vrací minimalizovanou hodnotu a metoda *copy*() vrací novou instanci, kopírující konstruktor pouze kopíruje vnitřní data a *print*() vypisuje řešení, nebo v případě této implementace i ukládá řešení.

```
minimizeCenterBoolCSP(minimizeCenterBoolCSP &s) : IntMinimizeScript(s),
                                                  boolVars(s.boolVars.size()){
    objective.update(*this, s.objective);
    variables_x_y.update(*this, s.variables_x_y);
    for(int i = 0; i < boolVars.size(); i++){
        boolVars[i].update(*this, s.boolVars[i]);
    }
    angleMultiplicator = s.angleMultiplicator;
}
```

Výpis kódu 2.9 Kopírující konstruktor třídy minimizeCenterBoolCSP

```
virtual IntVar cost() const {
    return objective;
}
```

Výpis kódu 2.10 Metoda cost, návratová hodnota je minimalizovaná hodnota, která musí být přidělena

```
virtual Space *copy(void) {
    return new minimizeCenterBoolCSP(*this);
}
```

Výpis kódu 2.11 Metoda copy, její navrátová hodnota je nová instance třídy, řešič využívá klonování třídy k výpočtu

Metoda print slouží pro vypsání výsledku, ale může jako v případě OR-Tools sloužit i pro uložení výsledku. Výsledek je odsazení, na osách x a y, středů objektů od středu tiskové plochy a jejich otočení, dále výsledek minimalizace ke středu a čas od začátku výpočtu.

Nejdůležitější část se nachází v samotném konstruktoru, s parametrem **Options**, který může obsahovat např. počet vláken, nebo počet řešení, bohužel neobsahuje možnost limitovat čas výpočtu, ten se musí nastavit v možnostech vyhledávače. Tento konstruktor slouží pro inicializaci

```
virtual void print(std::ostream &os) const {
    if(objective.val() < g_BestSolution){
        for (int i = 0; i < variables_x_y.size()/2; i++) {
            os << "object: " << i << ", x =  " << variables_x_y[2*i].val()
               << ", y = " << variables_x_y[2*i+1].val() << ", angle = ";
            int objectAngle = 0;
            for(int j = 0; j < boolVars[i].size(); j++){
                if(boolVars[i][j].val() == 1){
                    objectAngle = (int)(j*angleMultiplicator);
                    cout << (int)(j*angleMultiplicator) << endl;
                    break;
                }
            }
            if(g_BestSolution > g_PrintSize.first*g_Ratio){
                g_Solutions.emplace_back((double)(objectAngle),
                make_pair(variables_x_y[2*i].val(), variables_x_y[2*i+1].val()));
            }
            else{
                g_Solutions[i].first = (double)(objectAngle);
                g_Solutions[i].second =
                    make_pair(variables_x_y[2*i].val(), variables_x_y[2*i+1].val());
            }
        }
        os << "Minimized value = " << objective.val() << endl;
        os << "Time of solution since start: " <<
            std::chrono::duration<double>(std::chrono::system_clock::now() -
            g_WallTimeStamp).count() << endl;
        os << "/////////////////////////////////////////////////////" << endl;
        g_BestSolution = objective.val();
    }
}
```

Výpis kódu 2.12 Metoda print, kde dochází k vypsání a uložení výsledku

proměnných, nastavení jejich domén a vytvoření omezení. Stejně jako v případě OR-Tools implementace se rozdělí na následující části, ale jelikož samotné vypsání je v jiné metodě, tak se v konstruktoru nenachází.

- parametry,
- inicializace,
- omezení otočení,
- reifikační funkce
- omezení vzhledem k velikosti tiskové plochy,
- omezení žádné 2 objekty se neprotínají,
- minimalizace vzdálenosti k prostředku tiskové plochy,
- příprava řešiče.

2.2.2.1 Parametry

Implementace v Gecode se od té v OR-Tools liší také parametry, sice jsou stejné hodnoty a proměnné, ale jejich název a umístění je jiné, v případě Gecode, který využívá klonování tříd, a tím by potřeboval zkopírovat i vnitřní části (např. zkopírovat informace o objektech by mohlo být zdlouhavé a hlavně paměťově náročné z důvodu implementací objektů v knihovně CGAL), tak proměnné jsou v podobě globálních proměnných.

- **g_Objects**: Informace o objektech, obsahuje hlavně kružnice objektu.
- **g_Center**: Souřadnice S_t, kde se nachází střed tiskové plochy v milimetrech.
- **g_Solutions**: Pole pro uložení nejlepšího řešení.
- **g_PrintSize**: Párová proměnná, která obsahuje hodnoty V_x, V_y, neboli velikost tiskové plochy.
- **Options**: nastavení pro řešič
- **g_RotationCombination**: Parametr $pocetBituKombinaciUhlu$ určuje počet rotací objektů.
- **g_Ratio**: Zastupuje *meritko*, parametr slouží pro škálování výpočtů.

2.2.2.2 Inicializace

Další rozdíl od OR-Tools je existence polí pro proměnné, pro proměnné typu int $IntVarArray$, pro proměnné typu bool $BoolVarArray$, s vlastním konstruktorem, který přijímá počet proměnných, a jejich doménu, tato notace urychluje samotné psaní implementace, ale jejich používání je důležité i pro samotný řešič, větvení na poli je efektivnější [26], jak větvení na samotných proměnných. Ke konci inicializace je ještě omezení celého pole proměnných typu bool, že pouze jedna proměnná, a právě jedna proměnná v $boolVars[i]$, neboli proměnný určený k rotacím bude rovno hodnotě 1. Zbytek incializace se svojí logikou podobá implementaci v OR-Tools, zahrnuje také konvertování úhlu na radiány a následný vyýpočet funkcí sinus a kosinus.

```
minimizeCenterBoolCSP(const Options &opt) :
        IntMinimizeScript(opt), variables_x_y(*this, g_AmountOfObjects*2,
        -g_PrintSize.first*g_Ratio/2, g_PrintSize.first*g_Ratio/2),
        objective(*this, 0, g_PrintSize.first*g_Ratio),
        boolVars(g_AmountOfObjects) {
    std::vector<IntVarArray> intVarArraysX(g_AmountOfObjects);
    std::vector<IntVarArray> intVarArraysY(g_AmountOfObjects);
    int angles = (int)pow(2,g_RotationCombination);
    angleMultiplicator = 360./angles;
    const double pi = std::acos(-1);
    for(int i = 0; i < g_Objects.getObjects().size(); i++){
        intVarArraysX[i] = IntVarArray(*this, g_Objects.getObjects()[i].m_Circles.size(),
        0, g_PrintSize.first*g_Ratio);
        intVarArraysY[i] = IntVarArray(*this, g_Objects.getObjects()[i].m_Circles.size(),
        0, g_PrintSize.first*g_Ratio);
    }
    std::vector<double> sineValues;
    std::vector<double> cosValues;
    for (int i = 0; i < angles; i++) {
        sineValues.push_back(sin(double(int(i * angleMultiplicator)) * pi / 180));
        cosValues.push_back(cos(double(int(i * angleMultiplicator)) * pi / 180));
    }
    for(int i = 0; i < g_Objects.getObjects().size(); i++){
        boolVars[i] = BoolVarArray(*this, angles, 0, 1);
        Gecode::linear(*this, boolVars[i], IRT_EQ, 1);
    }
    .
    .
    .
```

Výpis kódu 2.13 konstruktor třídy minimizeCenterBoolCSP sloužící pro formulaci problému s omezeními

2.2.2.3 Omezení otočení

Omezení pro rotace, omezení vzhledem k velikosti tiskové plochy a omezení protnutí dvou kružnic v Gecode se od implementace v OR-Tools v logice vůbec nemění, jediný rozdíl je použití funkcí řešiče Gecode, namísto OR-Tools a jiné pojmenování proměnných.

```
for (int i = 0; i < g_Objects.getObjects().size(); i++) {
    for(int j = 0; j < g_Objects.getObjects()[i].m_Circles.size(); j++) {
        for (int k = 0; k < angles; k++) {
            int resultX = (int) (g_Ratio *
                (cosValues[k] * (
                shiftPointToCenter(g_Objects.getObjects()[i].m_Circles[j].second.first,
                    0, CGAL::to_double(g_Objects.getObjects()[i].m_CenterPoint.x()))) -
                (sineValues[k] * (
                shiftPointToCenter(g_Objects.getObjects()[i].m_Circles[j].second.second,
                    0, CGAL::to_double(g_Objects.getObjects()[i].m_CenterPoint.y()))))) +
                g_Center.first
            ));

            int resultY = (int) (g_Ratio *
                (cosValues[k] * (
                shiftPointToCenter(g_Objects.getObjects()[i].m_Circles[j].second.second,
                    0, CGAL::to_double(g_Objects.getObjects()[i].m_CenterPoint.y()))) +
                (sineValues[k] * (
                shiftPointToCenter(g_Objects.getObjects()[i].m_Circles[j].second.first,
                    0,CGAL::to_double(g_Objects.getObjects()[i].m_CenterPoint.x()))))) +
                g_Center.second
            ));
            addImplicationRestriction(intVarArraysX[i][j], resultX, k, boolVars[i]);
            addImplicationRestriction(intVarArraysY[i][j], resultY, k, boolVars[i]);
        }
    }
}
```

Výpis kódu 2.14 Omezení pro rotace v Gecode

2.2.2.4 Reifikační funkce

Poloviční reifikace nepožaduje speciální funkce, nebo motedu. V Gecode ji lze aplikovat pomocí operátoru >> ve funkci *Gecode* :: *rel*().

```
void addImplicationRestriction(IntVar & intVar, int result,
                               int caseNumber, BoolVarArray & boolVarArray){
    Gecode::rel(*this, boolVarArray[caseNumber] >> (result == intVar));
}
```

Výpis kódu 2.15 Funkce pro poloviční reifikaci výsledku otočení

2.2.2.5 Omezení vzhledem k velikosti tiskové plochy

Pozice objektu se omezí tak, aby nepřesahoval přes hranice tiskové plochy.

```
for(int i = 0; i < intVarArraysX.size(); i++){
    for(int j = 0; j < intVarArraysX[i].size(); j++){
        Gecode::rel(*this, variables_x_y[2*i] + intVarArraysX[i][j] <
            g_PrintSize.first*g_Ratio -
            (int)(g_Objects.getObjects()[i].m_Circles[j].first*g_Ratio));
        Gecode::rel(*this, variables_x_y[2*i+1] + intVarArraysY[i][j] <
            g_PrintSize.first*g_Ratio -
            (int)(g_Objects.getObjects()[i].m_Circles[j].first*g_Ratio));
        Gecode::rel(*this, variables_x_y[2*i] + intVarArraysX[i][j] >
            (int)(g_Objects.getObjects()[i].m_Circles[j].first*g_Ratio));
        Gecode::rel(*this, variables_x_y[2*i+1] + intVarArraysY[i][j] >
            (int)(g_Objects.getObjects()[i].m_Circles[j].first*g_Ratio));
    }
}
```

Výpis kódu 2.16 Omezení vzhledem k velikosti tiskové plochy v Gecode

2.2.2.6 Omezení žádné dva objekty se neprotínají

Namísto omezení pro násobení dvou libovolných proměnných má Gecode omezení pro umocnění proměnné na její druhou mocninu, jinak logicky totožné s implementací v OR-Tools, pouze se používají funkce a proměnný pro Gecode, namísto pro OR-Tools.

```
for (int i = 0; i < intVarArraysX.size(); i++) {
    for (int j = i + 1; j < intVarArraysX.size(); j++) {
        for(int k = 0; k < intVarArraysX[i].size(); k++) {
            for(int l = 0; l < intVarArraysX[j].size(); l++) {
                IntVar tmpEqualX = IntVar(*this,
                    -g_PrintSize.first*g_Ratio, g_PrintSize.first*g_Ratio);
                IntVar tmpEqualY = IntVar(*this,
                    -g_PrintSize.first*g_Ratio, g_PrintSize.first*g_Ratio);
                IntVar tmpMultiplicationX = IntVar(*this, 0,
                    g_PrintSize.first*g_Ratio*g_PrintSize.first*g_Ratio);
                IntVar tmpMultiplicationY = IntVar(*this, 0,
                    g_PrintSize.first*g_Ratio*g_PrintSize.first*g_Ratio);
                Gecode::rel(*this, tmpEqualX ==
                    variables_x_y[2*i]+intVarArraysX[i][k]-
                    variables_x_y[2*j]+intVarArraysX[j][l]);
                Gecode::rel(*this, tmpEqualY ==
                    variables_x_y[2*i+1]+intVarArraysY[i][k]-
                    variables_x_y[2*j+1]+intVarArraysY[j][l]);
                Gecode::sqr(*this, tmpEqualX, tmpMultiplicationX);
                Gecode::sqr(*this, tmpEqualY, tmpMultiplicationY);
                int radius = (int) (
                    (g_Ratio * g_Objects.getObjects()[i].m_Circles[k].first +
                     g_Ratio * g_Objects.getObjects()[j].m_Circles[l].first) *
                    (g_Ratio * g_Objects.getObjects()[i].m_Circles[k].first +
                     g_Ratio * g_Objects.getObjects()[j].m_Circles[l].first));
                Gecode::rel(*this, tmpMultiplicationX + tmpMultiplicationY > radius);
            }
        }
    }
}
```

Výpis kódu 2.17 Omezení protnutí 2 kružnic, 2 různých objektů o rotaci v Gecode

2.2.2.7 Minimalizace vzdálenosti k prostředku tiskové plochy

Gecode se od OR-Tools liší v minimalizačním omezení, zatímco v OR-Tools je možnost dát omezení $\leq$, v Gecode je potřeba minimalizovanou proměnnou explicitně přiřadit, proměnná tedy musí mít maximálně jednu hodnotu pro každý průběh přiřazení hodnot. To způsobuje problémy pro minimalizaci na spojitých proměnných, který jsou v Gecode implementovaný pomocí intervalů. Minimalizovaná proměnná objective se přiřadí pomocí funkce Gecode::expr, kde hodnota je maximální vzdálenost, za použití manhattanská metriky, nejvzdálenějšího objektu od středu tiskové plochy.

```
IntVarArray tmpAbsArray(*this, intVarArraysX.size(), 0, g_PrintSize.first*g_Ratio);
for(int i = 0; i < intVarArraysX.size(); i++){
    Gecode::rel(*this, tmpAbsArray[i] ==
        (abs(variables_x_y[2*i+1]) + abs(variables_x_y[2*i])));
}
objective = Gecode::expr(*this, max(tmpAbsArray));
```

Výpis kódu 2.18 Omezení pro minimalizaci proměnné objective v Gecode

2.2.2.8 Příprava řešiče

Rozdíl přichází také ke konci implementace formulace, kde v případě OR-Tools se pouze předaly parametry řešiče, minimalizavanou proměnnou pomocí metody a spustil se řešič. Gecode požaduje říct, jaké proměnné budou zahrnuty v prohledávání, jakou proměnnou si má vybrat a jakou hodnotu jí ma přiřadit, popř. pokud má vyhledáváč přiřazovat proměnné bez znalosti předchozích rozhodnutí, nebo pokud s nimi má počítat, Gecode má tak spoustu možností, a pro různé problémy může být vhodné použít jiné nastavení. V případě bool proměnných je výběr přes všechny hodnoty jediná možnost, zatímco proměnné typu int mají možností více, změna tohoto parametru může obrovsky změnit rychlost výpočtu, nebo jeho kvalitu.

```
for(int i = 0; i < boolVars.size(); i++){
    Gecode::branch(*this, boolVars[i], BOOL_VAR_ACTION_MAX(0.95), BOOL_VAL_MIN());
}
Gecode::branch(*this, variables_x_y, INT_VAR_ACTION_MAX(0.95), INT_VAL_MED());
```

Výpis kódu 2.19

2.2.3 Implementace druhé formulace

Druhá formulace je přímo určena pro OR-Tools, kde vysoký počet proměnných způsobuje větší nárok na paměť počítače a v případě nedostatku paměti při prohledávání může OR-Tools řešič spadnout. Implementace se moc od první implementace v OR-Tools neliší, liší se pouze v počtu bool proměnných určený pro rotaci, v reifikaci výsledku rotace a ukládání a vypsání výsledku. Tato část obsahuje výpis z kódu pouze těch částí implementace, které se liší od první formulace, implementované v OR-Tools. Zbytek implementace, tedy ty části, které tu chybí, se shoduje s implementací první formulace v OR-Tools.

```
std::vector<std::vector<BoolVar>> boolVars;
for (int i = 0; i < objects.size(); i++) {                                    1
    for (int j = 0; j < anglesToCodeSize; j++) {
        boolVars[i].push_back(cp_model.NewBoolVar());
    }
}
```

Výpis kódu 2.20 Inicializace bool proměnných určených pro rotaci v implementaci za použití kombinací v OR-Tools

```
int angle = 0;
for(size_t j = boolVars[i].size() - 1; j < boolVars[i].size(); j--){
    angle <<= 1;
    if(SolutionBooleanValue(r, boolVars[i][j])){
        angle |= 1;
    }
}
```

Výpis kódu 2.21 Vypočtení výsledku otočení z kombinací pro objekt při zpracování výsledku

```
void addOnlyIfRestriction(CpModelBuilder &cp_model, IntVar &intVar,
        long int result, int caseNumber, std::vector <BoolVar> &boolVars) {
    switch(boolVars.size()){
        case 0:
            cp_model.AddEquality(intVar, result);
            break;
        case 1:
            cp_model.AddEquality(intVar, result).OnlyEnforceIf({
                    ((caseNumber%2)==1) ? boolVars[0] : Not(boolVars[0])});
            break;
        .
        .
        case 8:
            cp_model.AddEquality(intVar, result).OnlyEnforceIf({
                    ((caseNumber%2)==1) ? boolVars[0] : Not(boolVars[0]),
                    (((caseNumber >> 1)%2)==1) ? boolVars[1] : Not(boolVars[1]),
                    (((caseNumber >> 2)%2)==1) ? boolVars[2] : Not(boolVars[2]),
                    (((caseNumber >> 3)%2)==1) ? boolVars[3] : Not(boolVars[3]),
                    (((caseNumber >> 4)%2)==1) ? boolVars[4] : Not(boolVars[4]),
                    (((caseNumber >> 5)%2)==1) ? boolVars[5] : Not(boolVars[5]),
                    (((caseNumber >> 6)%2)==1) ? boolVars[6] : Not(boolVars[6]),
                    (((caseNumber >> 7)%2)==1) ? boolVars[7] : Not(boolVars[7])});
            break;
        }
    }
}
```

Výpis kódu 2.22 Propagace výsledku rotace do proměnný, za použití poloviční reifikace podmíněnou kombinací bool proměnných

Kapitola 3
Experimentální část

Experimentální část se bude zabývat rychlostí výpočtu, kvalitou výsledku, ale i porovnání samotných řešičů, jejich možností, výhod a nevýhod. Všechny experimenty jsou prováděny na notebooku Acer Nitro AN515-54, 16GiB paměti ram, procesor Intel Core i5-9300H 2.4GHz × 8, na notebooku je operační systém Ubuntu 22.04.3 LTS. Objekty pro řešiče byl vybrán dataset zvířat z českého zdroje `https://www.printables.com/cs/model/465763-animals-wallart-46-models` [27].

V případě experimentů na řešiče je hlavní parametr jejich počet, jdou samozřejmě i upravit parametry jako je počet otočení, měřítko, nebo i větvení v případě Gecode. Začátek experimentování bude probíhat bez otočení, snaha bude mít co nejvíc objektů na tiskové ploše bez otočení, následně se experiment bude opakovat s větším počtem otočení. Následně bude ještě experimentování s kvalitou řešení, kde záleží na výsledné minimalizaci ke středu tiskové plochy za nějaký konkrétní čas.

3.1 Výsledky

První experiment je bez použití žádných rotací, další experimenty už jsou za použití maximálního počtu 256 rotací, kde jedna rotace od té následující se může lišit o 1 až 2 stupně. Pro všechny výpočty jsou oba řešiče nastavený na použití 16 vláken, limitování výpočtu časem je ve všech případech 120 vteřin, pokud to tak není, tak je tak výslovně řečeno. Všechny experimenty jsou prováděny s měřítkem 100. Délka výpočtu k řešení je měřena v reálném čase pomocí třídy $std :: chrono :: system_clock$ [28]. Objekty mají maximální počet kružnic 16, tedy čtyřstrom má maximální hloubku rovnou hodnotě 2. Experimenty na menších datech, neboli na méně objektů jsou opakovány 10×, experimenty na větších datech, kde je limit výpočtu nastaven na víc jak 120 vteřin se poté výpočet opakuje 5×, je to z důvodu, že například výpočet nedojde k výsledku v limitu, nebo řešič najde první výsledek, ale už nenajde další, lepší výsledek. Výsledky v tabulkách jsou pak průměry z výsledných hodnot pro daný test. Experimentální část se prvně zaměří na hodnotu výsledné minimalizace ke středu, což je hlavní objektivní porovnání, v pozdějších experimentech se ještě dostane ke kvalitě rozmístění, kde se zhodnotí na menším počtu objektů (max 5), zda řešič používá otočení objektu, aby efektivněji využil tiskovou plochu. Na většinu testů byl vybrán soubor Chicken.stl z stl datasetu zvířat. Důvod vybrání právě Chicken.stl je tvar objektu, jeho složitost, počet kružnic z objektu je 12, ale hlavně velikost objektu umožňuje rozmístění poměrně velkého množství objektů na tiskovou plochu.

První experiment spočívá v postupným rozmístění 2 až 6 objektů. U každého testu se zapamatuje nejlepší výsledek a kdy byl vypočítán. Čas je uváděn v sekundách a výsledek minimalizace v milimetrech, výsledky jsou zaokrouhleny na 3 desetinná místa. Test pro 6 objektů byl

limitován časem 20 minut. Oba řešiče měli stejné nastavení. V Gecode bohužel nebyl schopen řešič najít řešení v 20 minutách. Výsledek testů naznačuje, že problém roste s počtem objektů, ale hlavně s počtem kružnic, jelikož každý nový objekt přidá kružnice, který se nesmí protínat. Objekt souboru Chicken.stl má 12 kružnic, u každého nového i-tého objektu se pro každou kružnici vytvoří $12 * (i - 1)$ omezení, což znamená, že u třetího objektu je již celkem omezení $144+12*12*(3-1) = 432$, takže problém roste vzhledem k počtu objektů lineárně, ale vzhledem k počtu kružnic exponenciálně. Je důležité podotknout, že i když je uveden pouze jeden výsledek, který je průměrem ze všech testů, tak v každém testu jsou stovky výsledků, který řešič zlepšuje, v případě OR-Tools byl řešič schopen pro 2 a 3 objekty projít celý prostor a výsledek je tedy optimální.

OR-Tools	**čas**	**minimalizace**	Gecode	**čas**	**minimalizace**
2 objekty	1,667s	32,24	2 objekty	0,469s	64,47
3 objekty	12,502s	43,56	3 objekty	2,012s	64,47
4 objekty	26,675s	49,16	4 objekty	5,965s	64,47
5 objektů	83,261s	69,37	5 objektů	16,351s	64,47
6 objektů	508,125	105,60	6 objektů	-	-

Tabulka 3.1 Testy Chicken.stl, bez otáčení objektů

Další testy jsou již s rotacemi, kterých je největší možný počet z formulace, neboli 256, každá rotace se tak od té předchozí liší o 1, nebo 2 stupně úhlu. U 6 a 5 objektů je nastaven delší časový limit, na 20 minut, ani jeden řešič nenašel u 6 objektů výsledek. U 5 objektů je výsledek u OR-Tools spíše informativní, že řešič je schopen najít řešení, ale najití řešení může, ale nemusí dlouho trvat, je tam obrovská odchylka od různých výsledků.

OR-Tools	**čas**	**minimalizace**	Gecode	**čas**	**minimalizace**
2 objekty	41,151s	24,44	2 objekty	0,687s	63,39
3 objekty	117,341s	39,183	3 objekty	3.830s	64,24
4 objekty	113,487s	64,68	4 objekty	9,541s	64,32
5 objektů	567s	228,65	5 objektů	16,197s	64,66
6 objektů	-	-	6 objektů	-	-

Tabulka 3.2 Testy Chicken.stl, s 256 rotacemi pro otáčení objektů

Pokud se ale implementace v Gecode nastaví na jiné vetvění, tak poté dostanem výsledky pro Gecode i pro 18 objektů, problém je v jejich kvalitě, kdy výsledek je rozmístění objektů k levému dolnímu rohu tiskové plochy, ne ke středu. Zatímco při 2 objektech dochází k zlepšování výsledku na úrovni srovnatelné s OR-Tools, při více objektech už není takový úspěch a řešič vyhazuje výsledky malé kvality, střed tiskové plochy je volný a žádný objekt se u něj nenachází. Uvedený výsledek je nejlepší, který dokázal řešič vypočítat, pro méně objektů je horších výsledků stovky až tisíce. Limit byl tentokrát 120 vteřin pro všechny testy, až na 19 objektů, kdy byl 20 minut, tam ovšem nebyl výsledek najit, možný důvod je, že není jisté, zda se 19, nebo 20 objektů Chicken.stl bez otočení vejde na tiskovou plochu.

Poslední testy jsou dělány na implementaci druhé formulace v OR-Tools, všechny parametry jsou stejné jako v minulých případech. Pro 5 objektů najít výsledek v některých případech trvalo déle jak 120 vteřin, v tabulce je uvedený průměr z nejlepších výsledků, který byl řešič schopen najit do 120 vteřin. Rozdíl od testů na první formulaci je ten, že tato formulace dokáže najít výsledek víc konzistentně do 120 vteřin pro 5 objektů. Při testu najití řešení do 120 vteřin našla tato formulace výsledek 6. z 10 pokusů, první formulace našla řešení pouze jedno z deseti pokusů.

Gecode	**čas**	**minimalizace**
2 objekty	93,678s	31,58
3 objekty	28,658s	117,41
4 objekty	96,779s	149,35
5 objektů	20.477s	204,09
6 objektů	0,740	234,82
7 objektů	2,963	234,70
8 objektů	4,919	234,69
9 objektů	2,958	234,78
10 objektů	3,671	234,80
11 objektů	4,706	234,81
12 objektů	23,724	234,77
13 objektů	13,988	234,78
14 objektů	19.346	234,77
15 objektů	14,867	234,79
16 objektů	24,078	234,79
17 objektů	15,734	234,81
18 objektů	19.141	234,81
19 objektů	-	-
20 objektů	-	-

Tabulka 3.3 Testy Chicken.stl, s 256 rotacemi pro otáčení objektů, Gecode přiřazení minimálních hodnot při větvení

OR-Tools	**čas**	**minimalizace**
2 objekty	74,340s	24,762
3 objekty	115,482s	41,84
4 objekty	113,487s	64,68
5 objektů	70,142s	91,17
6 objektů	-	-

Tabulka 3.4 Testy Chicken.stl, s 256 rotacemi pro otáčení objektů

3.2 Srovnání výsledků a rotací

Výsledek dalších testů jsou obrázky výskledku, kde je vidět otočení a snaha řešiče umístit objekty ke středu tiskové plochy, spolu s výsledky pro jednotlivé objekty, výstup je dělán pomocí knihovny SFML v C++ a je součástí kódu s řešiči. Další obrázky jsou v příloze této práce. Zatímco implementace v OR-Tools využívá rotací a snaží se objekty co nejlépe natočit, implementace v Gecode rotací nevyužívá a objekty se snaží dát pouze na osy u středu tiskové plochy. Možný řešení pro Gecode by bylo napsat vlastní omezení, nebo vlastní brancher na větvení proměnných, jelikož je to něco, co Gecode umožňuje. Další možnost je použít náhodné přiřazení hodnot, to ale může způsobit, že řešič se nebude přibližovat k výsledku a prostě náhodně tipovat hodnoty. Pravděpodobně nejlepší řešení je pak napsání vlastních funkcí pro větvění a souvisejícího přiřazování hodnot.

3.3 Srovnání řešičů

Součástí práce je i porovnání řešičů, jejich použitelnosti, funkcí a jednoduchosti použití. To zahrnuje osobní zkušenosti z používání řešičů. Jednoduchost používání, instalace, ale i nesrovnalosti, co se týče padání řešiče.

Výhody OR-Tools

- Intuitivní používání
  Funkce a metody jsou intuitivní, jejich menší počet zajišťuje větší přehlednost pro uživatele.

- Aktivní komunita
  Důkazem je například diskuze na oficiální Github stránce OR-Tools [29], v porovnání s Gecode, kde je diskuze o dost chudší a v poslední době není aktivní [30], dalším důkazem aktivní komunity jsou otázky ohledně fungování OR-Tools na stackoverflow [31], kde je oproti Gecode [32] mnohonásobně více otázek na pracování s knihovnou.

- Aktivní vývoj
  V době psaní práce bylo vydáno za poslední šest měsíců tři nové verze a takřka denně jsou vydávány aktualizace na testovací větev knihovny [29].

- Lepší výsledky
  Výsledný rozmístění objektů na tiskové ploše je za použití uhlů a výsledek se snaží zlepšovat až do optimální vzdálenosti od středové plochy. Gecode má výsledek rychle, ale dále už ho nezlepšuje.

- Rozsáhlá dokumentace
  OR-Tools obsahuje rozsáhlou dokumentaci pro práci s knihovnou, jsou popsány třídy. OR-Tools také obsahuje spoustu příkladů, pro práci s OR-Tools [33].

Nevýhody OR-Tools

- Instalace
  Instalace knihovny a její použití s vývojovým prostředím C-Lion bylo obtížné, hlavně proto, že instalační příručka na oficiálních stránkách, alespoň v mém případě nefungovala, knihovna není malá a každý pokus o instalaci trvá alespoň 30 minut, samozřejmě v závislosti na rychlosti počítače a kvalitě internetu. Příručka na oficiální GitHub stránce projektu již fungovala.

- Málo funkčností
  Narozdíl od řešiče Gecode, má OR-Tools málo funkcí, ale i omezení, možností použití proměnných, důvodem je vnitřní fungování CP-SAT řešiče. V OR-Tools například chybí spojité proměnné, možnost přenásobit proměnné, jediná možnost je pomocí mezivýsledků a použití omezení. V OR-Tools sice má možnosti vytvoření vlastního omezení, ale ani samotný autor a jednoho z hlavních vývojářů Laurent Perron to nedoporučuje, z důvodu absence nástrojů na zkoušení nových omezení [22].

- Neobsahuje spojité proměnné
  Jak již bylo zmíněno u předchozí nevýhody, tato nevýhoda je zvlášť velká, protože problém s omezeními o rozložení objektů na tiskové ploše pracuje v základní formě s spojitými proměnnými, problém bylo nutné překonvertovat do diskrétní podoby.

- Pomalejší výsledky
  Oproti Gecode byl OR-Tools pomalejší, výsledky OR-Tools ale považuji za kvalitnější.

- Při prohledávání může spadnout na nedostatek paměti
  OR-Tools při procházení prostoru, neboli větvení si pamatuje různé části stromu, pokud je

problém dostatečně velký, dostatečně velký záleží na více faktorech, například moc proměnných, velké domény, velký počet řešení. OR-Tools neobsahuje možnost omezit použití paměti, problém lze řešit větší pamětí v počítači, nebo zmenšit samotný problém, počet proměnných, menší rozsah domén.

Výhody Gecode

- Obsahuje spojité proměnné
  Gecode obsahuje spojité proměnné, umožňuje tak implementovat problémy s omezeními, které požadují spojité proměnné, bez větších úprav.
- Možnost vlastní implementace
  Knihovna umožňuje implementovat vlastní proměnný, brancher na větvení, omezení, dokonce i vlastní vyhledávač, otevřenost knihovny umožňuje implementaci takřka čehokoliv.
- Rozsáhlá příručka
  Gecode obsahuje velmi rozsáhlou přiručku na psaní v knihovně Gecode, obsahuje návody na napsání vlastních implementací důležitých částí řešiče, ale i informace k implementaci formulace problému s omezeními v Gecode. Odkaz na online příručku, s názvem "Modeling and Programming with Gecode `https://www.gecode.org/doc-latest/MPG.pdf` [21].
- Rozsáhlá dokumentace
  Stejně jako OR-Tools obsahuje Gecode rozsáhlou dokumentaci pro práci s knihovnou se spoustu příkladů [34].

Nevýhody Gecode

- Horší výsledky
  Gecode výsledky nevypadají tak kvalitně jako v OR-Tools, Gecode najde výsledek rychleji, ale jeho kvalita není na takové úrovni jako v OR-Tools, také ani u například 2 objektů se už řešiči nedaří výsledek vylepšit.
- Obtížnost použití
  Zatímco v OR-Tools funguje řešič jako samotný objekt, v Gecode je potřeba implementace třídy, OR-Tools umožňuje nastavení řešiče v jednom objektu pro parametry, Gecode má nastavení pro samotný řešič a poté i pro samotný vyhledávač řešení, o větvení se OR-Tools postará řešič, Gecode je potřeba větvení zvolit, nebo, pokud nesedí na problém, větvění implementovat.

3.3.1 Vlastní zkušenosti

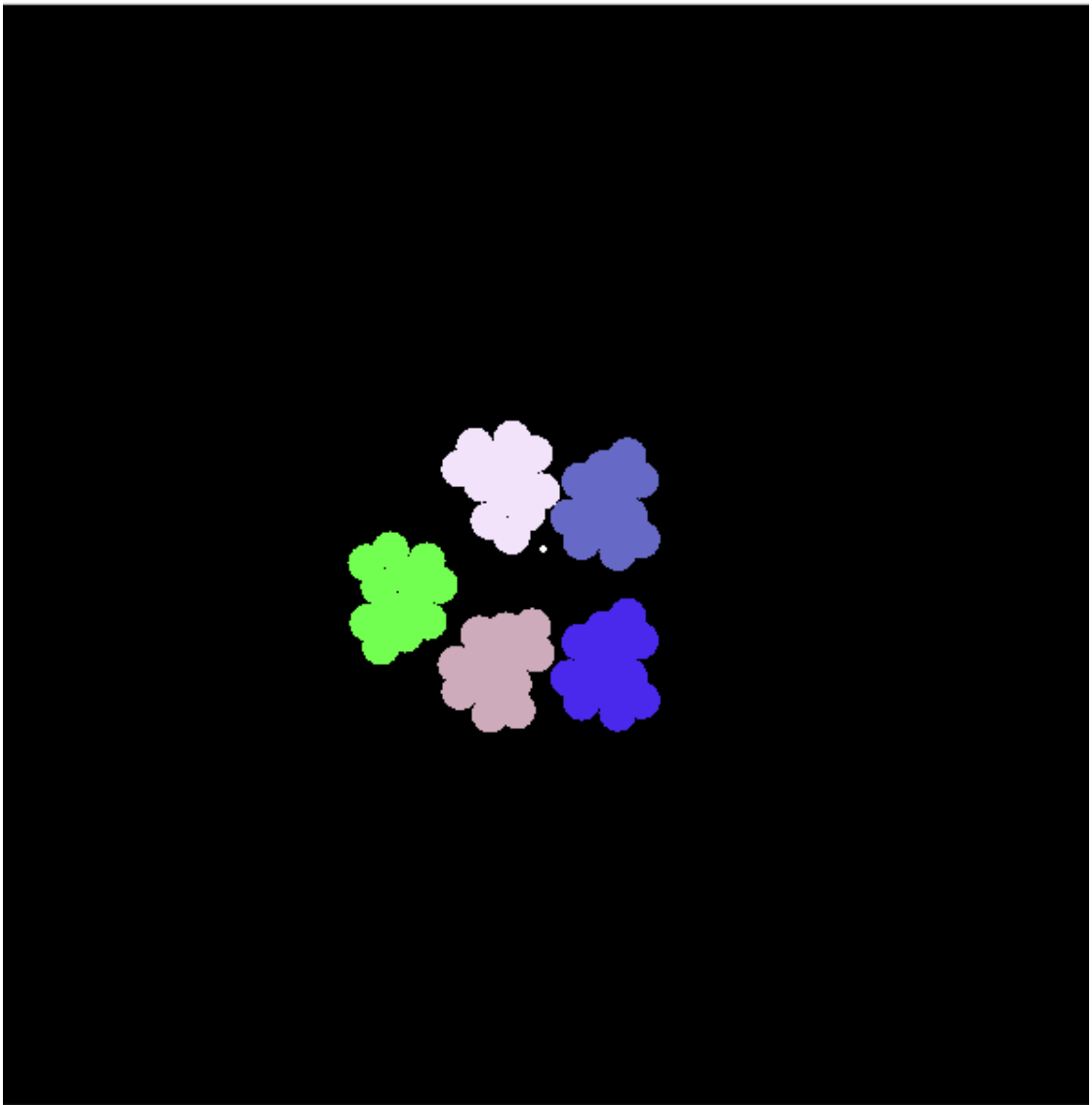
Na závěr hodnocení řešičů bych se chtěl podělit o osobní zkušenosti, protože problém s omezeními je druh řešení problémů, který lze použít na řešení spoustu problémů, proto je volba nástroje na implementace formulací důležitá součást řešení problému s omezeními. V následující části jsou osobní poznatky o používání obou řešičů, ve kterém ještě vyzdvihnu problémy a výhody obou řešičů.

Jedna z nevýhod OR-Tools se projevuje hned při instalaci knihovny, příručky na instalaci na oficiálních stránkách OR-Tools nevedly na úspěsnou instalaci, pouze na možnosti spuštění jednoho souboru přes speciální příkaz. Na možnosti použití OR-Tools s vývojovým prostředí C-Lion nakonec pomohla jiná příručka, ale OR-Tools pak funguje pouze jako poslední testovací verze, která nemusí být stabilní, jednou se dokonce stalo při vytváření práce, že nová verze pokazila kompilaci a knihovnu tak nebylo možné použít na celý víkend, hned v pondělí ale se problém vyřešil, komunita OR-Tools je aktivní a samotný autoři často pomáhají s komunitními dotazy.

Gecode takové problémy nemá, z příruček šel stáhnout a použít v C-Lionu stejně obtížně, nebo v tomhle případě jednoduše, jako každá další knihovna C++. V čem už má ale Gecode nevýhodu je v méně aktivní komunitě, na internetu už není tolik dotazů na Gecode a když už jsou, tak jsou součástí dotazů na řešič MiniZinc, který má jiné prostředí a Gecode se v něm pouze využívá. Navíc Gecode má rozsáhlou, online dostupnou, zdarma příručku, kde jsou návody jak problémy formulovat, ale i jak implementovat nové proměnné, domény, jak implementovat vlastní omezení s propagací ale i vlastní řešič, na internetu existující příručka poradí, jak na implementaci. Gecode také obsahuje spojité proměnné, narozdíl od OR-Tools, pokus o implementaci rozmisťování objektů na tiskové ploše 3D tiskárny jako problém s omezeními byl, ale spojité proměnné v Gecode mají své vlastní problémy a na implementaci by byla potřeba formulace speciálně pro spojité proměnné, stejně jako v případě diskrétních proměnných. Gecode také umožňuje překonvertování nespojité proměnné typu int na spojitou proměnnou typu float a naopak, ale zde nastává problém, když dochází ke konverzi ze spojité na diskrétní proměnnou, spojitá proměnná musí mít tvar celého čísla, jinak řešič spadne, bohužel na to v dokumentaci, nebo příručce upozorněno není [35]. Gecode zároveň neobsahuje možnost desetinnou část odtrhnout, nebo ani omezení, které by číslo zaokrouhlilo na celou hodnotu, to může být způsobeno tím, že spojité proměnné jsou v Gecode implementovány pomocí intervalů.

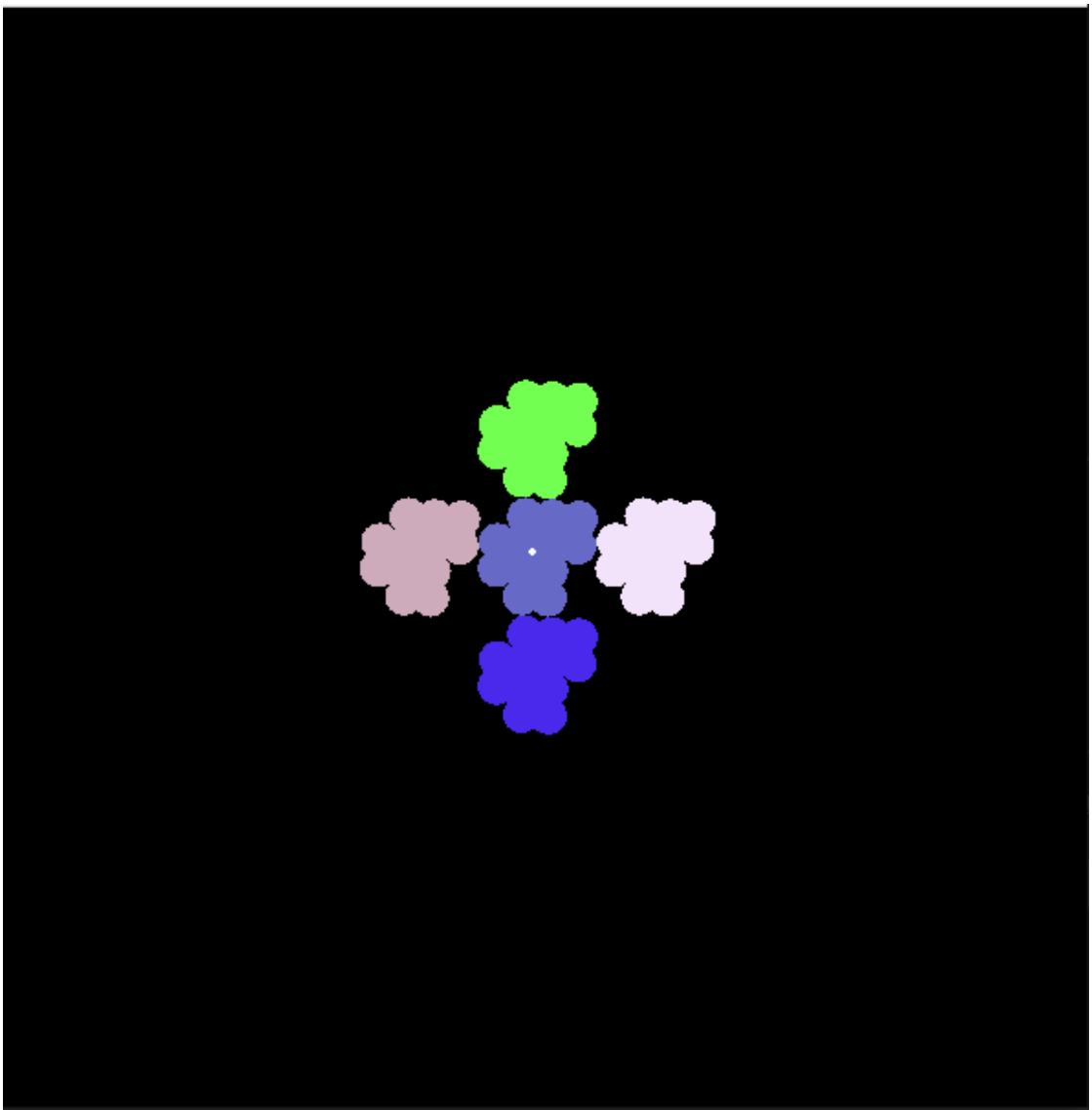
Další problém, na který jsem narazil souvisí také s proměnnými typu float, minimalizovaná proměnná v Gecode musí být přiřazena, to znamená, že někde v implementaci formulace musí mít minimalizovaná proměnná přiřazenou pouze jednu hodnotu, pokud to tak není řešič vyhodí vyjímku, situace je tedy ošetřena, ale v případě formulace pro rozmisťování položek na tiskové ploše a použití spojitých proměnných by musela být formulace předělána, jak již bylo zmíněno, protože přiřazení proměnný typu float jsem byl schopen docílit pouze pomocí větvení na minimalizované proměnné, s tím bohužel ale nepočítala formulace, která tak nedokáže vyhodit řešení. Všechny snahy o použití spojitých proměnných skončily neúspěchem. Věřím, že z důvodu větší možnosti většího zásahu do Gecode může být Gecode rychlejší a výsledky může vracet lepší, pokud by byl implementován vlastní větvení proměnných, jejich přiřazení, to už by ale byla implementace, i když částečná, vlastního řešiče, o čemž práce bohužel není.

Hlavní nevýhoda OR-Tools je nedostatek funkcí, tato skutečnost se hlavně potvrdila u problému, který obsahuje použití omezení jako je kosinus, nebo sinus, či mocnění a umocnění, problémy šlo nakonec obejít, a řešič má podle mě, když dostane dostatek času, šanci najít řešení i pro složitější problémy, než který byly testovány. To ale nemůžu potvrdit, ani vyvrátit, jelikož do řešiče není vidět, jediný co na to ukazuje je snaha řešiče objekty dávat ke středu tiskové plochy s takovým otočením, které benefituje rozložení objektů. OR-Tools také používá při vetvění prostor, neboli graf, který si pamatuje, to má za následek, že OR-Tools potřebuje dostatek paměti. Kolik přesně je potřeba paměti, to záleží na problému, který se implementuje, v případě překročení přidělené paměti řešič spadne. Nastavit limity na použitelnou paměť bohužel nelze a několikrát jsem se setkal s pádem řešiče z důvodu nedostatku paměti, hlavně při zkoušce hodně objektů, na problému s rozmístěním pomáhá zmenšit počet kružnic.



```
object: 0, x = 3392, y = 2400, angle = 28
object: 1, x = -8260, y = -2049, angle = 208
object: 2, x = 3392, y = -6372, angle = 28
object: 3, x = -2767, y = -6438, angle = 8
object: 4, x = -2624, y = 4066, angle = 236
Minimized value = 10309
Time of solution since start: 240.039
////////////////////////////////////////////////////
```

Obrázek 3.1 Výsledek v OR-Tools pro 5 objektů a 256 rotací



```
object: 0, x =  -38, y = 0, angle = 357
object: 1, x =  -38, y = 6428, angle = 357
object: 2, x =  -38, y = -6428, angle = 357
object: 3, x =  -6466, y = 0, angle = 357
object: 4, x =  6390, y = 0, angle = 357
Minimized value = 6466
Time of solution since start: 15.018
////////////////////////////////////////////////////////
```

Obrázek 3.2 Výsledek v Gecode pro 5 objekty a 256 rotací

Kapitola 4
Závěr

Cílem této práce bylo seznámit se s problematikou 3D tisku a technikami splňování omezení. Dále bylo cílem formulovat problém rozmisťování objektů pro 3D tisk jako problém splňování omezení. Nakonec bylo cíl práce otestovat možnosti řešení navržené formulace pomocí současných řešičů problémů s omezeními, jako je Gecode, nebo OR-Tools.

Výsledkem práce předzpracování objektů z 3D formy na formu, která odpovídá 2D reprezentaci objektu z pohledu ze shora, za pomoci knihoven CGAL a Admesh. Objekt ve formátu STL je zpracován do 2D reprezentace, dále se objekt pomocí čtyřstromu jsou z objektů vyjmutý kružnice, které v zjednodušené formě obsahují tvar objektu. Objekt je dále použit pro práci s řešičem, kde je využit v implementaci formulace v řešiči OR-Tools a Gecode. Výsledky obou řešičů jsou dále porovnány, jejich rychlost a kvalita výsledku. Zatímco Gecode dokázal najít výsledek často i rychleji, OR-Tools výsledek byl často kvalitnější a dokázal se zlepšovat, v malých případech dokázal i najít optimální výsledek, protože dokázal projít celý prostor řešení.

Bohužel na více objektech už ani jeden řešič přibližně ve stejný moment nedokázal najít v adekvátním čase řešení, to je za předpokladu, že oba řešiče využívají 16 vláken, možné řešení je změnit reprezentaci objektu na méně kružnic, nebo změnit parametry, či formulovat více omezení, které by dostatečně omezili prostor hledání, případně před spuštěním formulace této práce, spustit řešič na objektech nahrazenými čtverci, či obdelníky a využít již existující omezení na neprotnutelnost objektů. Výsledek poté využít při výpočtu s kružnicemi.

V práci jsem se osobně zabýval i formulováním problému pomocí spojitých proměnných, při implementaci jsem ale narazil na komplikace, který znemožnily implementaci formulace, např. přiřazení hodnoty spojitým proměnným nelze stejně jako u diskrétních proměnných, překonvertování z spojité hodnoty na diskrétní hodnotu musí splňovat, že spojitá proměnná nesmí mít desetinný rozvoj. Rozšíření práce by tedy mohlo být formulovat takovou formulaci, která s tímto bude počítat a půjde implementovat bez problémů.

Program obsahuje i primitivní vizualizaci objektů a jejich rozmístění, to by šlo vylepšit např. implementací UI, které by umožňovalo možnost vybrat objekty, jejich počet, nebo i řešiče opakovaně spouštět na jedno spuštění programu. Program je kompilován a zkoušen na operačním systému Linux, na operačním systému Windows není vyzkoušena jeho funkčnost, další rozšíření programu je zkompilovat program pro operační systém Windows a zajistit na něm i jeho funkčnost.

Bibliografie

1. EDELKAMP, Stefan; WICHERN, Paul. Packing Irregular-Shaped Objects for 3D Printing. *KI 2015: Advances in Artificial Intelligence: 45-58*. 2015.
2. JOSEF PRUSA, Prusa3D by. *Částečně sestavená 3D tiskárna Original Prusa MINI+*. 2024 [cit. 2024-04-25]. Dostupné také z: `https://www.prusa3d.com/cs/produkt/castecne-sestavena-3d-tiskarna-original-prusa-mini-10/`.
3. *Práce s 3D modely ve formě meshí* [`https://courses.fit.cvut.cz/BI-3DT/tutorials/mesh.html`]. 2022 [cit. 2024-04-25].
4. PROJECT, The CGAL. *CGAL/cgal*. 2024. Dostupné také z: `https://github.com/CGAL/cgal`.
5. HRONČOK, Miro; RANELLUCCI, Alessandro; CHVÁTAL, Tomáš; GLADKY, Anton; DOUCETTE, Andy; GOLASOWSKI, Ondřej; FINKELSTEIN, Udi; JIN, Chow Loong; O'BRIEN, Christopher; DEVENDRA; APESTEGUÍA, Fernando; SEGUIN, Guillaume; JONATAN1024; ŽEHRA, Marek; LOWE, Matthew; CEJKA, Tomas; DRONECFD; MNML_; MCORBE; 2BRIGHT. *admesh/admesh*. 2022. Dostupné také z: `https://github.com/admesh/admesh`.
6. *The StL Format Standard Data Format for Fabbers*. Ennex Research Corporation, 1997. Dostupné také z: `https://www.fabbers.com/tech/STL_Format`.
7. BURNS, Marshall. *Automated Fabrication: Improving Productivity in Manufacturing*. Prentice Hall, 1993. ISBN 978-0131194625.
8. SÜZEN, Ahmet Ali. *ASCII and binary STL file*. 2024 [cit. 2024-04-26]. Dostupné také z: `https://www.researchgate.net/figure/ASCII-and-binary-STL-file_fig2_326668533`.
9. CANESSA, E.; FONDA, C.; ZENNARO, M. Low-cost 3D Printing for Science, Education and Sustainable Development. In: 2013, s. 19–20. ISBN 92-95003-48-9. Dostupné také z: `http://sdu.ictp.it/3d/book.html`. Online; 2024-04-26.
10. S.R.O, dk metal prominent. *FUSED DEPOSITION MODELING (FDM)*. 2024 [cit. 2024-04-25]. Dostupné také z: `https://www.dkmp.cz/o-nas/detail/prehled-technologii-3d-tisku`.
11. *Slicing* [`https://courses.fit.cvut.cz/BI-3DT/tutorials/slicing.html`]. 2022 [cit. 2024-04-25].
12. FOWLER, Robert J.; PATERSON, Michael S.; TANIMOTO, Steven L. Optimal packing and covering in the plane are NP-Complete. *Information Processing Letters*. 1981.

13. FRIEDMAN, Erich. Packing unit squares in squares: A survey and new results. *The Electronic Journal of Combinatorics [electronic only]*. 1998, roč. DS07, Research paper DS7, 24 p.–Research paper DS7, 24 p. Dostupné také z: http://eudml.org/doc/231733.
14. DECHTER, Rina. *Constraint processing*. Elsevier Morgan Kaufmann, 2003. ISBN 978-1-55860-890-0.
15. EDELKAMP, Stefan; SCHRÖDL, Stefan. Chapter 13 - Constraint Search. In: EDELKAMP, Stefan; SCHRÖDL, Stefan (ed.). *Heuristic Search*. San Francisco: Morgan Kaufmann, 2012, s. 571–631. ISBN 978-0-12-372512-7. Dostupné z DOI: https://doi.org/10.1016/B978-0-12-372512-7.00013-4.
16. SCHULTE, Christian; TACK, Guido; LAGERKVIST, Mikael Z. Modeling and Programming with Gecode. In: 2019 [cit. 2024-04-26], s. 53. Dostupné také z: https://www.gecode.org/doc-latest/MPG.pdf.
17. GOOGLE. *OR-Tools* [https://developers.google.com/optimization/introduction]. 2024 [cit. 2024-04-25].
18. GOOGLE. *OR-Tools* [https://developers.google.com/optimization]. 2024 [cit. 2024-04-25].
19. *The MiniZinc Challenge* [https://www.minizinc.org/challenge/]. 2024 [cit. 2024-05-13].
20. *a4cp* [http://www.a4cp.org/events/cp-conference-series]. 2024 [cit. 2024-04-28].
21. SCHULTE, Christian; TACK, Guido; LAGERKVIST, Mikael Z. Modeling and Programming with Gecode. In: 2019 [cit. 2024-04-26]. Dostupné také z: https://www.gecode.org/doc-latest/MPG.pdf.
22. PERRON, Laurent. *custom constraint or-tools*. 2018 [cit. 2024-04-28]. Dostupné také z: https://groups.google.com/g/or-tools-discuss/c/qANoFaIsj88.
23. PROJECT, The CGAL. *Computational Geometry Algorithms Library*. 2024 [cit. 2024-04-25]. Dostupné také z: https://www.cgal.org/.
24. THE CGAL PROJECT. *CGAL User and Reference Manual*. 5.6.1. CGAL Editorial Board, 2024. Dostupné také z: https://www.cgal.org/.
25. *Rotation Matrix* [https://mathworld.wolfram.com/RotationMatrix.html]. 2024 [cit. 2024-05-12].
26. SCHULTE, Christian; TACK, Guido; LAGERKVIST, Mikael Z. Modeling and Programming with Gecode. In: 2019 [cit. 2024-04-26], s. 122. Dostupné také z: https://www.gecode.org/doc-latest/MPG.pdf.
27. KORYNTA, Ondřej. *Animals wallart 46 models*. 2023. Dostupné také z: https://www.printables.com/cs/model/465763-animals-wallart-46-models.
28. CPPREFERENCE.COM. *std::chrono::system_clock*. 2023 [cit. 2024-05-12]. Dostupné také z: https://en.cppreference.com/w/cpp/chrono/system_clock.
29. GOOGLE. *Google/or-tools: Google's operations research tools*. 2024 [cit. 2024-05-11]. Dostupné také z: https://github.com/google/or-tools.
30. GECODE. *Gecode*. 2024 [cit. 2024-05-11]. Dostupné také z: https://github.com/Gecode/gecode.
31. *Questions tagged [or-tools]*. 2024 [cit. 2024-05-11]. Dostupné také z: https://stackoverflow.com/questions/tagged/or-tools.
32. *Questions tagged [gecode]*. 2024 [cit. 2024-05-11]. Dostupné také z: https://stackoverflow.com/questions/tagged/gecode.
33. GOOGLE. Google OR-Tools Reference. In: 2024 [cit. 2024-04-27]. Dostupné také z: https://developers.google.com/optimization/reference.

34. GECODE. Gecode: Documentation. In: 2020 [cit. 2024-04-27]. Dostupné také z: https://www.gecode.org/documentation.html.

35. SCHULTE, Christian; TACK, Guido; LAGERKVIST, Mikael Z. Modeling and Programming with Gecode. In: 2019 [cit. 2024-04-26], s. 100. Dostupné také z: https://www.gecode.org/doc-latest/MPG.pdf.

Obsah příloh

- `readme.txt` ..... stručný popis obsahu média
- `projectCSP` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
- `BP_vysledky` ..... adresář s obrázky výsledků práce