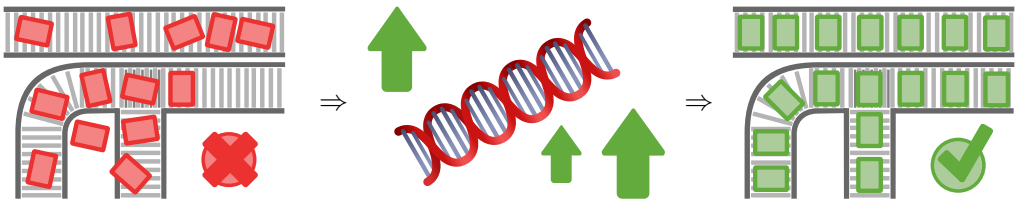
BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2021
http://excel.fit.vutbr.cz

Warehouse Manager: Nástroj pro pokročilou simulaci a optimalizaci skladových operací

Bc. Filip Kočica*



Abstrakt

Tato práce řeší problematiku alokace produktů do lokací ve skladu za pomoci moderních heuristických přístupů a realistické simulace. Práce poskytuje grafický nástroj umožňující sestavení modelu skladu, generování syntetických zákaznických objednávek, optimalizaci alokace produktů za pomoci kombinace *state of the art* technik, simulátor vytvořeného modelu skladu a nakonec nástroj pro hledání nejkratší cesty objednávky skrze sklad. Práce také uvádí porovnání různých přístupů a experimenty s vytvořenými nástroji. Podařilo se optimalizovat propustnost skladu na téměř dvojnásobek ($\sim 57\%$). Se zvyšující se komplexitou skladu se kvalita optimalizace lehce snižuje ($\sim 10\%$). Přínosem této práce je možnost vytvoření modelu plánovaného či již existujícího skladu a jeho simulace i optimalizace, což může značně zvýšit propustnost skladu a pomoci detekovat a odstranit vytížená místa. To může vést k ušetření zdrojů či pomáhat v plánování. Dále tato práce přináší nový způsob optimalizace skladu a nové optimalizační kritérium.

Klíčová slova: Sklad, optimalizace, simulace, generátor, objednávka, produkt, pickování, evoluce

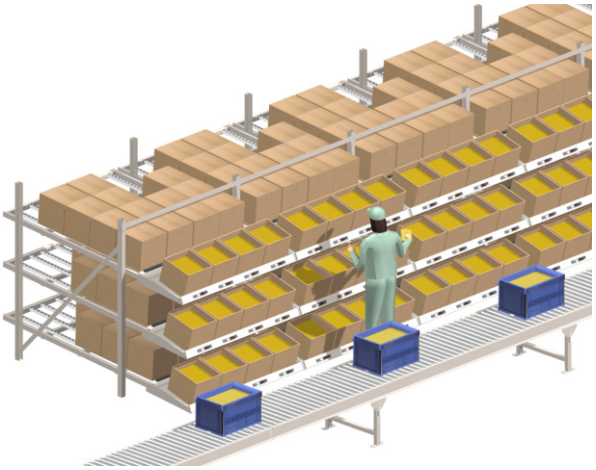
Přiložené materiály: Demonstrační video

*xkocic01@fit.vutbr.cz, *Fakulta informačních technologií, Vysoké učení technické v Brně*

1. Úvod

Sklady jsou velmi důležitá část dodavatelského řetězce, kam jsou dočasně uloženy produkty z výroby, než jsou odeslány k zákazníkům v rámci objednávek. Simulace a automatizace skladů se vzhledem k rostoucím nárokům na jejich výkonnost a propustnost v posledních letech značně rozšířila. Automatizace skladu~~, ač velmi nákladný proces,~~ může společnosti přinést nebývalé zvýšení produktivity, bezpečnosti a v neposlední řadě také kvality, resp. menší chybovosti.

Zákaznická objednávka sestává z několika položek, kde každá položka jednoznačně identifikuje zakoupený produkt a jeho požadované množství. Objednávky jsou typicky vyřizovány ve formě kartonu, do kterého se vloží zakoupené produkty (popřípadě faktura, atp.) a karton se odešle k zákazníkovi. Vyřizování takových objednávek v rámci skladu poté sestává z cestování kartonu mezi lokacemi po dopravnících a vybírání požadovaných produktů ze slotů lokací (úložné prostory, kde jsou produkty dočasně uloženy) do kartonů objednávek. Proces sbírání zakoupených produktů do kartonů je obecně označován jako pickování objednávek, viz 1. Způsob pickování objednávek má zřejmě největší vliv na výkonnost skladu jako celku, a proto je často považován za nejslibnější oblast z hlediska optimalizace skladových operací. Z pohledu optimalizace pickování je velmi důležité rovnoměrné rozložení zatížení mezi jednotlivé oblasti skladu [1]. Tato práce řeší kombinatorický NP-těžký problém, a ~~sice~~ jakým způsobem rozmístit



Obrázek 1. Na obrázku lze vidět lokaci sestávající z jednotlivých slotů a pickera, což je pracovník skladu, který vytahuje produkty ze slotů lokace do kartonů objednávek, které přijíždí po dopravníku[1].

produkty do slotů lokací ve skladu, aby byla propustnost skladu co nejvyšší. Tato problematika se v literatuře označuje jako SLAP (*storage location assignment problem*).

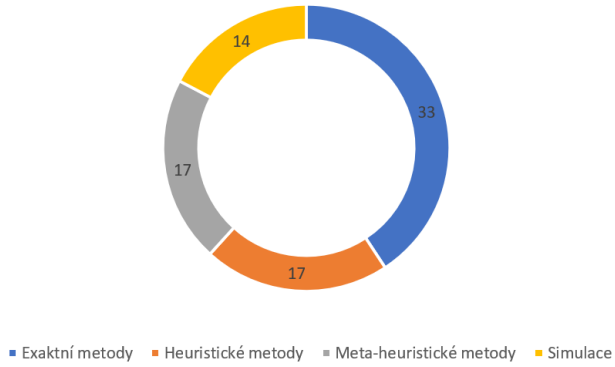
Z množství vědeckých příspěvků, které se v posledních letech zabývaly problematikou SLAP lze usuzovat, že je to velmi aktivní a diskutované téma. Z obsahu těchto příspěvků pak lze usuzovat, že toto téma není zdaleka vyřešené, a existuje zde velký potenciál pro možná zlepšení, a to zejména z pohledu zvýšení výkonnosti skladů. Více v sekci 2.

Přístup k řešení dané komplexní úlohy v této práci spočívá v implementaci sady pěti kooperujících nástrojů:

- **Generátor** – na základě pravděpodobnostních modelů generuje korelující sady zákaznických objednávek pro trénování a testování.
- **Simulátor** – provádí realistickou simulaci běhu vytvořeného skladu na generovaných či importovaných objednávkách.
- **Pathfinder** (česky hledač cest) – umožňuje nalezení optimální cesty objednávky skrze sklad.
- **Optimalizátor rozložení produktů** – provádí optimalizaci rozložení produktů ve skladu za účelem zvýšení propustnosti skladu pomocí evolučních algoritmů.
- **Warehouse manager** (česky skladový manažer) – grafický nástroj, kde má uživatel možnost vytvořit model skladu dle jeho potřeb.

To vše spojené v jedné přehledné a snadno použitelné

[1] Převazato z http://orderpickingfastfetch.blogspot.com/2013/01/what-is-pick-to-light-pick-to-light-or.html



Obrázek 2. Graf udávající přístupy pro řešení problematiky SLAP v odborné literatuře a jejich četnost. Vytvořeno na základě dat z [2].

grafické aplikaci Warehouse manager[2]. Hlavním přínosem tedy bude zvýšení propustnosti skladu a to díky hned několika optimalizačním technikám, které jsou v dokumentu kvantitativně i kvalitativně porovnány.

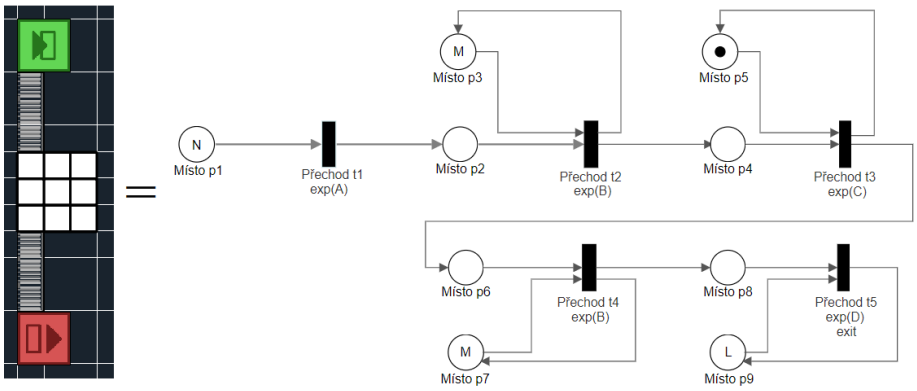
Vytvořené řešení je zcela nezávislé na modelu skladu, ten si lze vytvořit zcela libovolně nebo dle potřeb. Výsledky optimalizace jsou u malých skladů na velmi vysoké úrovni, ačkoli se zvyšující se komplexitou skladu se kvalita optimalizace snižuje. Mimo optimalizaci rozložení produktů ve skladu poskytuje řešení další užitečné funkcionality, jako je například zmíněná identifikace úzkých míst (tzv. *bottlenecků*) či nalezení nejkratší cesty objednávky skrze sklad.

2. Související práce

Na obrázku 2 lze vidět rozdělení metod řešení SLAP do čtyř kategorií. Každá také s četností, s jakou byla v posledních letech využita. Jak lze vidět, nejčastěji se používají exaktní metody. O druhé místo se dělí heuristické a meta-heuristické metody. V poslední řadě se jedná o simulace. Práce řešící tuto problematiku nejčastěji optimalizují kritéria jako jsou čas, prostor a vzdálenost.

V práci [3] autoři implementovali nástroj pro minimalizaci času zpracování objednávek. Práce řeší problém SLAP pomocí seřazení produktů od nejčastěji kupovaného po nejméně kupovaný a slotů lokací od nejbližšího k vchodu a východu ze skladu až po ten nejvzdálenější. Následně provádí namapování produktů do slotů tak, že nejčastěji kupovaný produkt je v nejvýhodnějším slotu, atd. Následně byl nástroj vyhodnocen na modelu existujícího skladu a bylo zjištěno, že časy manipulace s materiálem byly zredukovány o 37.8%. Tento princip byl (pro porovnání) využit i v této práci a jej vidět v grafu 7 jako `SLAP` a dosáhl

[2] Nástroje však poskytují také CLI (*command line interface*).



Obrázek 3. Triviální sklad se vstupem, jednou lokací a výstupem propojených dopravníkem. Vpravo je odpovídající PT síť. Zjednodušený popis: na začátku se nachází místo `p1` s `N` tokeny, kde `N` se rovná počtu objednávek, které chceme ve skladu zpracovat. Tyto objednávky přichází do systému v časových intervalech (`A`) daných exponenciálním rozložením. Po vstupu objednávky musí její karton vjet na dopravník. Ten má však omezenou kapacitu danou výpočtem délka dopravníku děleno velikost jednoho kartonu (`M`). Doba po kterou jede karton po dopravníku je vypočtena jako délka dopravníku děleno rychlost kartonu (`B`). Poté karton vjede do lokace, kde si alokuje pickera na dobu (`C`), která je spočtena jako suma doby pickování všech produktů, které se v této lokaci mají pickovat. Obdobně karton projede další dopravník a nakonec je karton odeslán ze skladu – je spočtena doba odesílání jedné objednávky (`D`), a také kolik jich lze odesílat zároveň (`L`). Vesměs každá hodnota v celém procesu je konfigurovatelná.

zlepšení 33.2%.

Genetický algoritmus pro minimalizaci cestované vzdálenosti ve skladu byl použit v práci [1]. Autoři optimalizovali přesně definovaný model skladu daný zákazníkem popsaný matematickou funkcí, a podařilo se jim snížit cestovanou vzdálenost ve skladu při zpracovávání objednávek o 28%. To vede ke značnému zrychlení pickování a dle jejich výpočtů by to ušetřilo zákazníkovi 2700 eur ročně.

3. Implementace

V rámci této práce bylo vytvořeno pět kooperujících nástrojů, které lze použít jak v textovém, tak grafickém režimu. Všechny nástroje jsou konfigurovatelné pomocí XML souborů či přímo v GUI. Veškeré nástroje byly implementovány v `C++` (standard `c++17`), pro grafickou nástavbu byl použit framework `Qt` verze 5.

3.1 Generátor

Tento nástroj slouží pro vygenerování dvou vzájemně korelujících sad zákaznických objednávek. První sada je použita pro optimalizaci skladu (dále nazývaná jako trénovací sada) a druhá pro vyhodnocení kvality optimalizace (dále nazývána jako testovací sada).

Pravděpodobnostní modely, na základě kterých se generování provádí, jsou Gaussova rozložení – parametry (střední hodnotu a rozptyl) definuje uživatel. Generátor je založen na hodnotách ADU[3] a ADQ[4].

[3] *Average daily units* – průměrný denní počet zakoupení.

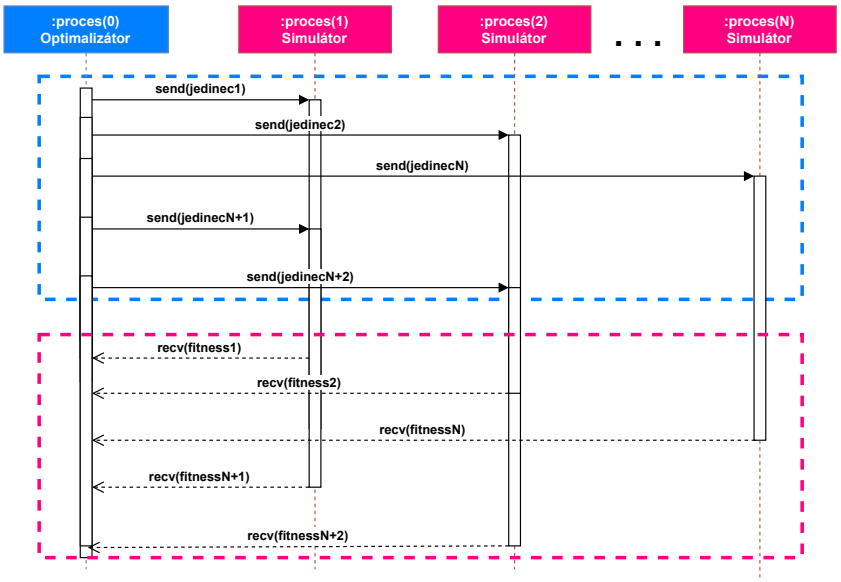
[4] *Average daily quantity* – průměrná denní zakoupená kvantita.

Samotné generování probíhá tak, že se vygeneruje hodnota ADU pro každý produkt a spočtou se pravděpodobnosti zakoupení jednotlivých produktů pomocí rovnice:

$$p_i = \frac{ADU_i}{\sum_{n=1}^{N} ADU_n}, \tag{1}$$

z čehož vznikne diskrétní pravděpodobnostní rozložení. Poté se iteruje přes počet objednávek, které chce uživatel vygenerovat. Pro každou takovou objednávku se z normálního rozložení vygeneruje počet položek, které má tato objednávka mít. Poté je pro každou položku nutno vybrat produkt. To se provádí náhodným výběrem z pravděpodobnostního rozložení z rovnice 1, a tedy čím větší má produkt spočtenou pravděpodobnost zakoupení, tím má vyšší šanci výběru do objednávky. Nakonec se projdou všechny objednávky i jejich položky a pro každou z položek je vygenerována kvantita (zakoupené množství). To se provede tak, že vygenerovaná hodnota z rozložení ADQ pro daný produkt se vydělí počtem zakoupení tohoto produktu, tedy vygenerovaná kvantita se rozdělí mezi zakoupené produkty.

To ve výsledku dává tři Gaussova rozložení, první pro ADU, druhé pro ADQ a třetí pro počet položek objednávky. Vzhledem k tomu, že obě sady objednávek jsou generovány ze stejných pst. rozložení, vzniklé sady jsou různé, avšak spolu korelují. Při použití GUI jsou generovaná data „fitována“ do třech grafů.



Obrázek 4. Sekvenční diagram znázorňující paralelizaci optimalizace (rovnoměrné rozdělení výpočtu simulací všech jedinců populace mezi N procesů). Modrý obdélník označuje odeslání jedinců na simulaci a růžový pak vysbírání výsledků doby simulace od jednotlivých potomků.

3.2 Simulátor

Účelem tohoto nástroje je odsimulování zpracování importovaných či generovaných objednávek na vytvořeném modelu skladu tak, jako by to byl reálný skladový systém. Lze jej použít samostatně (např. pro identifikaci úzkých míst, či pro statistickou analýzu), avšak jeho hlavní účel je aproximace kvality nalezeného řešení v optimalizátoru rozložení produktů – jinými slovy je použit jako objektivní funkce.

Autoři práce [4] zmiňují, že ze všech možných druhů je nejvhodnější a nejpřirozenější simulace skladu pomocí diskrétních událostí, protože sklad je v podstatě kolekce entit, které reagují na fixní diskrétní události. Simulátor je tedy založen na principu diskrétní simulace a při implementaci byla využita knihovna `SIMLIB/C++`[5]. Simulátor poskytuje poměrně komplexní konfiguraci, což umožňuje rozsáhlé možnosti experimentování (od nastavení rychlostí pracovníků a dopravníků až po doplňování produktů viz 3.2.2). Princip je vysvětlen na snímku 3.

3.2.1 Paralelizace

Simulátor je využíván mj. optimalizačním nástrojem pro vyhodnocení kvality řešení. Taková simulace je spouštěna pro každého jedince populace v každém kroku evolučního algoritmu. To v případě velkých populací vede k velmi dlouhému trvání optimalizace. Vzhledem k tomu, že knihovna `SIMLIB/C++` nebyla koncepčně navržena pro účely paralelního zpracování, nebylo možné provést zrychlení použitím více vláken.

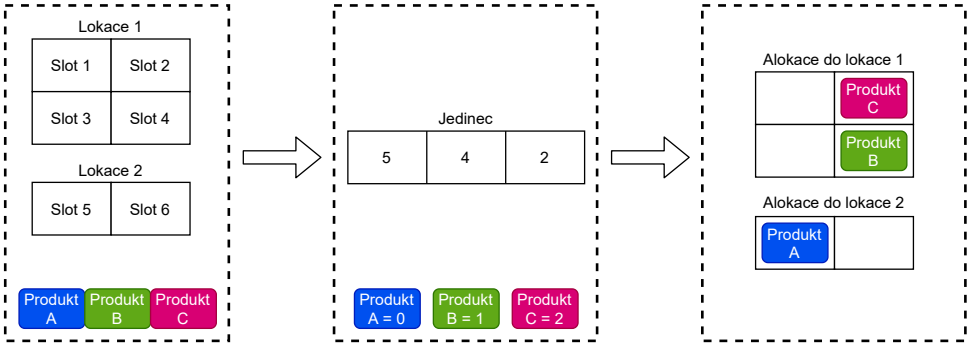
Tento problém byl vyřešen spuštěním několika instancí (procesů) využívající tuto knihovnu, které se nijak neovlivňují a mohou fungovat souběžně. Před začátkem optimalizace je tedy vytvořeno N (konfigurovatelné) takových procesů. Účel takto vytvořených potomků je jednoduchý, a sice provést inicializaci (objednávek, modelu skladu a před-počítání cest), očekávat data, provést simulaci a vrátit výsledek simulace hlavnímu procesu. Po odeslání dat zpět rodiči potomek opět vstupuje do blokujícího čekání na data nebo ukončení komunikace a tedy i samotného potomka. Komunikace mezi rodičem a potomky je následující (viz 4):

- Od rodiče k potomkům se posílá zakódovaná alokace produktů do jednotlivých slotů (celočíselné pole).
- Od potomků zpět k rodiči se posílá výsledek (doba) simulace, a sice hodnota *fitness* (číslo s plovoucí řádovou čárkou).

Jedinci (potažmo jejich simulace) jsou mezi procesy rozděleny zcela rovnoměrně a zrychlení optimalizace pomocí paralelizace simulací bylo velmi značné.

Dále bylo zjištěno, že až 30% všech jedinců v populaci je duplicitních (stejných jako nějaký jiný jedinec). Byl proto implementován mechanismus převodu zakódovaných genů jedince na řetězec, a pomocí hashovací tabulky bylo zajištěno, že se nebudou provádět duplicitní simulace ale jedinci se stejnými geny si pouze překopírují již spočtený výsledek jiného jedince. To vedlo k ještě většímu zrychlení celého procesu a optimalizace i opravdu komplexního skladu bylo možné počítat v řádu maximálně několika dní.

[5] *Simulation Library for C++* – `http://www.fit.vutbr.cz/~peringer/SIMLIB`



Obrázek 5. Triviální příklad pro navržené kódování pro alokaci tří produktů do šesti slotů.

3.2.2 Doplňování produktů

Doplňování produktů (ang. *replenishment*) je proces, při kterém se ze zásobníku produktů doplňují produkty do slotů, ze kterých se pickují zákaznické objednávky. Toto se typicky provádí ve chvíli, kdy množství produktů ve slotu klesne pod určitou (konfigurovatelnou) úroveň. Tento mechanismus byl implementován za účelem zvýšení realističnosti simulace skladu.

3.3 Pathfinder

Nástroj pro optimalizaci cesty byl implementován skrze evoluční algoritmus $\mathscr{MAX}$–$\mathscr{MIN}$ mravenčí systém, a nazývá se `pathfinder`. Cílem tohoto nástroje je nalézt optimální cestu objednávky skrze sklad, tak, aby urazila co nejkratší možnou vzdálenost. Vzhledem k tomu, že každá objednávka potřebuje navštívit jiné lokace, optimální cesta skrze sklad se zpravidla liší, a proto je nutné optimální cestu hledat pro každou objednávku samostatně. Grafická nástavba dokáže mimo tvorbu grafu také zvýraznit aktuálně nejlepší nalezenou cestu vybrané objednávky skrze sklad a očíslovat grafické prvky, aby bylo zřejmé, v jakém pořadí je objednávka navštíví.

3.4 Optimalizátor rozložení produktů

Automatizované sklady jsou mnohdy velmi komplexní systémy a mají mnohá omezení daná jejich layoutem, způsobem manipulace s produkty, úložnými a pickovacími politikami atd. Optimalizace výkonnosti takovýchto skladů často vyžaduje přesnou definici jejich modelu a nelze jej jednoduše převést na matematický výraz. Vzhledem k tomu, a také k možnosti uživatele si vlastnoručně vytvořit model skladu, by bylo velmi obtížné takto obecně vytvořit matematický popis skladu, proto tato práce pro vyhodnocení kvality používá simulaci, která odpovídá reálnému fungování skladu. Myšlenka je taková, že se optimalizací minimalizuje objektivní funkce, a tou je simulovaná doba (tzn. snižuje se doba potřebná ke zpracování všech objednávek). Doba zde představuje simulační čas, nikoli reálný.

Pro nalezení optimální distribuce produktů do slotů lokací byl jako nejvhodnější přístup vybrán GA (genetický algoritmus), a to protože v podstatě nepotřebuje znát matematický popis problému, pouze problém zakódovat jako sekvenci čísel. Dále byly však pro porovnání implementovány další tři evoluční algoritmy, a sice: DE (diferenční evoluce), ABC (algoritmus umělých včelstev) a PSO (optimalizace rojem částic). Všechny tyto algoritmy pracují standardně ve spojitém prostoru, a tedy bylo potřeba je na základě odborných prací [5, 6, 7, 8] redefinovat pro diskrétní prostor, např. pro GA implementovat tzv. „seřazené“ genetické operátory, které neprodukují duplicity genů [8]. K tomu pomohla zejména velká podobnost problematiky SLAP a TSP (problému obchodního cestujícího), pro který byly tyto redefinice v odborných článcích popsány). Trénování optimalizátoru probíhalo na metacentru[6] a optimalizátor podporuje ukládání a načítání vah modelu.

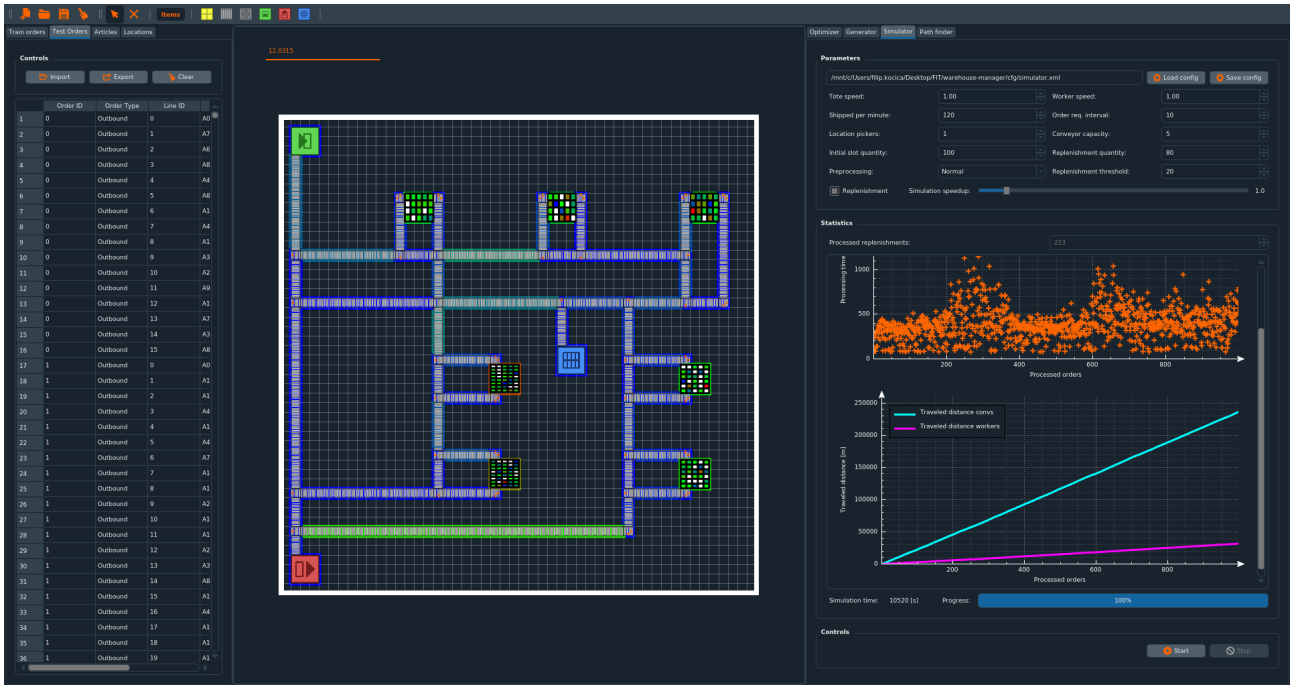
3.4.1 Kódování

Vzhledem k tomu, že byly pro řešení problematiky použity evoluční algoritmy, bylo třeba vymyslet kódování dané problematiky do sekvence binárních, reálných, či celých čísel. Výsledný princip kódování je pro všechny čtyři evoluční algoritmy stejný a sice zakódovaný jedinec je reprezentován jednoduše jako vektor celých čísel. Index ve vektoru identifikuje produkt a hodnota na daném indexu reprezentuje slot, do kterého je daný produkt alokován. Příklad znázorňující toto kódování lze najít na obrázku 5.

3.5 Warehouse manager

Layout grafické aplikace byl vytvořen za pomoci aplikace `Qt Designer`: v horní liště lze najít tlačítka pro ovládání, jako např. načtení již existujícího modelu ze souboru, atd. Dále se v této liště nachází jednotlivé zařízení skladu, které může uživatel využít pro vytvoření modelu skladu. Na levé straně aplikace jsou záložky, které slouží pro import, export a investigaci

[6] https://metavo.metacentrum.cz/cs



Obrázek 6. Grafická aplikace, ve výsledku nazvaná Warehouse Manager, disponuje mimo původní účel (tj. tvorba modelu skladu) také veškerými funkcionalitami implementovanými v rámci této práce. To znamená veškerou kontrolu nad simulátorem, `pathfinder`-em, generátorem i optimalizátorem rozložení. To umožňuje plné využití této práce i např. logistickým manažerům, zcela bez nutnosti využít příkazovou řádku. Na snímku je zachycena simulace skladu na základě modelu skladu, importovaných objednávek, aktuální alokace produktů do slotů a nastavených parametrů. Jednotlivé prvky/zařízení i sloty lokací jsou doplněny o barevnou *heatmap*-u, která u prvků znázorňuje jejich vytížení (červená značí maximální vytížení, modrá malé vytížení) a u slotů jak často je obsažený produkt kupován. V pravé části jsou vytvořeny grafy a statistiky s podrobnějšími informacemi.

Tabulka 1. Nejlepší konfigurace optimalizátoru GA.

Parametr	Hodnota
Selekce	Turnaj
Mutace	Uspořádaná [8]
Křížení	Uspořádané [8]
Hodnota *trial*	10
Pravděpodobnost křížení	0.6
Pravděpodobnost mutace jedince	0.4
Pravděpodobnost mutace genu	0.2

dat využívaných jednotlivými nástroji, jako jsou (zleva): objednávky pro trénování, testování, produkty a lokace se sloty, které mohou obsahovat i aktuální alokaci produktů do daných slotů. Vpravo jsou opět záložky, které slouží pro přepínání mezi jednotlivými nástroji. Každý z těchto čtyř nástrojů má ve své záložce následující části:

- **Konfigurace** – Zde jsou veškeré parametry, které daný nástroj poskytuje a lze je předvyplnit konfiguračním souborem, či je vy-exportovat.
- **Statistiky** – Zde jsou obsaženy grafy, do kterých jsou postupně doplňovány získané hodnoty.
- **Řízení** – Poskytuje tlačítka pro kontrolu jednotlivých nástrojů, jako je spuštění či zastavení.

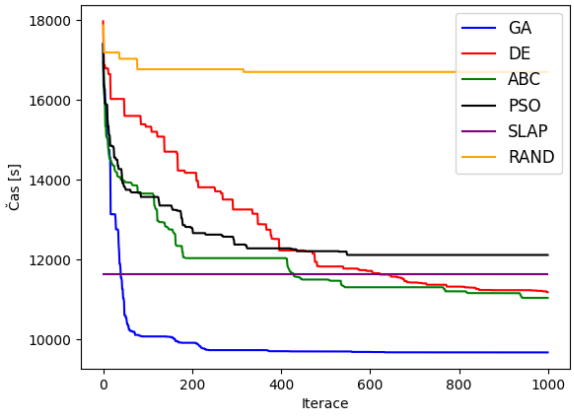
Uprostřed aplikace je plocha určená pro tvorbu a manipulaci s modelem skladu. Cílem této plochy je poskytnout úplný a intuitivní 2D editor poskytující různé druhy skladových prvků a manipulaci s nimi. Ve výsledku editor (mimo jiné) umožňuje:

- Přiblížení a oddálení scény/modelu skladu.
- Měřítko vůči reálnému světu.
- Změnu velikosti, pozice a rotaci prvků.
- Propojování skladových prvků pomocí portů.
- Hromadnou selekci a kopírování prvků.
- Zobrazení podrobných informací o prvku.
- Uložení modelu skladu do souboru a načtení ze souboru.

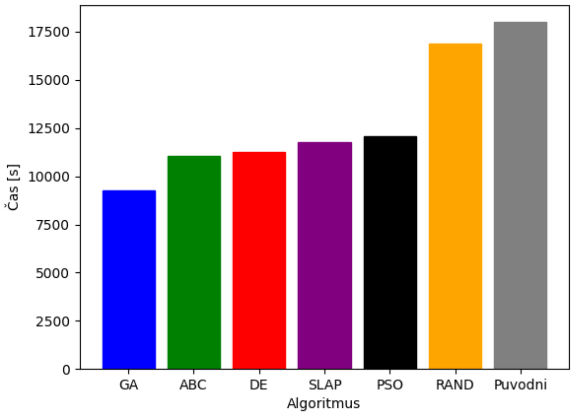
Pro implementaci plochy pro tvorbu modelu skladu byla využita grafická scéna (`QGraphicsScene`) a grafický pohled. Do grafické scény jsou postupně umisťovány objekty odvozené z grafických prvků, které je rozšiřují o specifické vlastnosti vhodné pro daný *use-case*.

4. Experimenty

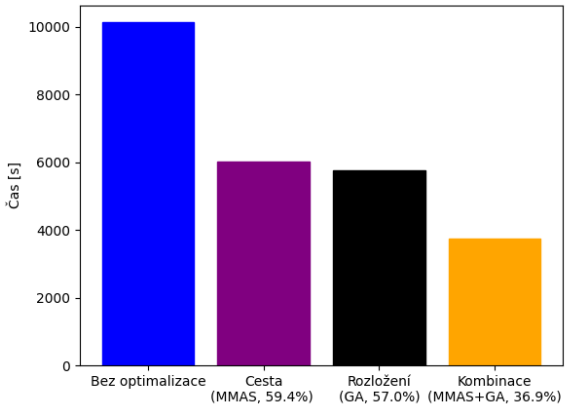
Pomocí experimentů bylo zjištěno, že nejlepších výsledků dosahuje algoritmus GA a byla nalezena jeho



Obrázek 7. Graf průběhu trénování čtyř evolučních algoritmů, metodiky SLAP a náhodného prohledávání. Ve všech případech byl použit stejný model skladu a stejné trénovací objednávky. Na vodorovné ose jsou iterace algoritmu, zatímco na svislé ose je doba potřebná pro zpracování veškerých trénovacích objednávek v sekundách (neboli doba simulace).



Obrázek 8. Sloupcový graf vyhodnocení nejlepších natrénovaných modelů na testovací sadě objednávek, a to pro každou z metod. Pro porovnání je přidán poslední sloupec udávající jak dlouho zhruba trvalo testovací objednávky zpracovat před optimalizací. Barevné schéma metod je ponecháno stejné pro zachování konzistence.



Obrázek 9. Sloupcový graf porovnávající optimalizaci cesty pomocí $\mathscr{MM}$AS, optimalizaci rozložení produktů pomocí genetického algoritmu a nakonec jejich kombinaci.

optimální konfigurace, kterou lze vidět v tabulce 1.

Nejúspěšnější experiment vznikl kombinací algoritmu GA a vlastnosti algoritmu ABC. Algoritmus ABC pro každou včelu (řešení problému) udržuje hodnotu *trial*, která udává počet kroků algoritmu, ve kterých se daná včela nezlepšila. Pokud tato hodnota dosáhne před-definované hodnoty, je tato včela nahrazena novou náhodně vygenerovanou včelou. Problém u GA byl, že se zasekával v lokálních minimech a nedokázal najít příliš dobré řešení, proto jsem jej zkusil doplnit o tuto vlastnost a nalezené řešení bylo vždy lepší o v průměru o cca. 5.11%.

5. Vyhodnocení

Vyhodnocení je rozděleno na tři části: optimalizaci rozložení produktů, cesty a nakonec jejich kombinaci.

5.1 Optimalizátor rozložení produktů

Jak již bylo zmíněno, při optimalizaci tohoto problému se minimalizuje doba potřebná ke zpracování sady objednávek. Na grafu 7 lze vidět průběh trénování evolučních algoritmů na problému rozřazení 150 produktů do 200 slotů, což lze vyjádřit jako kombinatorický problém:

$$V_{150}(200) = \frac{200!}{50!} = 2.593067\mathrm{e}+310.$$

Dále je na grafu vidět také průběh náhodného prohledávání prostoru (`RAND`) a klasické metodiky (`SLAP`). Tato metodika je jednokrokový výpočet mapující nejčastěji kupované produkty do nejvýhodnějších slotů, a proto je konstantní. Na grafu 8 lze poté vidět vyhodnocení modelů na testovací sadě objednávek. Nejlépe si vedl genetický algoritmus, kterému se povedlo snížit dobu potřebnou ke zpracování 1000 objednávek téměř na polovinu. Se zvětšujícími se problémy (sklady) se kvalita optimalizace zhoršuje (cca. o 10%), zejména kvůli obrovskému nárůstu komplexity.

5.2 Pathfinder

Úkolem tohoto doplňkového nástroje je nalezení co nejkratší (optimální) cesty objednávky skrze sklad. Největší problém, na kterém byl tento algoritmus

testován sestával z 200 lokací, což už je z hlediska skladových politik obrovský sklad a tedy nemá cenu řešit tento problém pro větší modely skladu. Algoritmus v závislosti na konfiguraci dokáže nalézt optimální řešení tohoto problému do 300 iterací. U skladů typických velikostí je optimální cesta nalezena maximálně do 100 iterací algoritmu.

5.3 Kombinace

Nástroj `pathfinder` lze použít v rámci simulátoru pro hledání optimálních cest pro objednávky. Vzhledem k tomu, že optimalizátor rozložení používá simulátor pro aproximaci kvality řešení, lze použít všechny tři nástroje zároveň a optimalizovat jednak rozložení produktů a zároveň délku cesty. Na grafu 9 lze vidět porovnání: neoptimalizovaný sklad (200 produktů, 400 slotů), nejlepší dosažené výsledky samostatných optimalizací a nakonec kombinace těchto optimalizací. Jak lze vidět, kombinací těchto optimalizátorů lze dosáhnout ještě lepších výsledků, avšak za velice vysokou cenu doby trénování, která i pro menší sklad dosahuje velmi vysokých hodnot, což není v souladu s fungováním skladů, které musí být schopny rychle reagovat na změny požadavků.

6. Shrnutí

Práce měla za úkol vytvořit nástroj, který bude schopen optimalizovat fungování skladu za účelem zvýšení jeho propustnosti. Důležitou podmínkou byla nezávislost optimalizace na modelu skladu (tj. uživatel jej může vytvořit dle svých potřeb), čehož bylo dosaženo za pomoci grafického editoru a realistické simulace. Dále bylo třeba vytvořit generátor syntetických dat kvůli citlivosti zákaznických dat a nakonec kvantitativně vyhodnotit dosažené výsledky.

Nástroj `pathfinder` dokáže nalézt optimální cestu skrze sklad v relativně malém počtu iterací algoritmu a zrychlení zpracování objednávek je téměř dvojnásobné – **59.4%**. Stejně tak optimalizátor rozložení produktů dokáže téměř dvojnásobně zrychlit zpracování všech objednávek – **57%**, avšak doba pro natrénování je zde značně delší. Kombinací těchto dvou přístupů zároveň pak lze dosáhnout ještě lepších výsledků, avšak za cenu velmi dlouhé optimalizace. Při značném zvýšení komplexity (velikosti problému) se kvalita optimalizace zhoršuje – cca. o **10%**.

Přínosem této práce je úplný grafický nástroj, který dosahuje velmi dobrých výsledků, poskytuje nespočet funkcí a který je vyvíjen pro společnost SEACOMP s.r.o, kde snad nalezne reálné využití. Dále tato práce přináší novou metodu k řešení problematiky SLAP a sice kombinaci dvou *state of the art* technik a nakonec přináší nové optimalizační kritérium v kontextu SLAP.

V budoucnu by bylo možné rozšířit práci o nástroj schopný generovat optimální rozložení skladu na základě uživatelem definovaných podmínek, a to za pomoci CGP (kartézského genetického programování).

Poděkování

Zde bych rád poděkoval svému vedoucímu práce Ing. Oldřichu Kodymovi, konzultantu Ing. Danielu Chalupovi a kolegu Bc. Davidu Vosolovi za cenné rady a podnětné připomínky. Výpočetní zdoje byly poskytnuty projektem „e-Infrastruktura CZ“ (e-INFRA LM2018140) v rámci programu „Projects of Large Research, Development and Innovations Infrastructures“.

Literatura

[1] E. Bottani, M. Cecconi, G. Vignali, and R. Montanari. Optimisation of storage allocation in order picking operations through a genetic algorithm. *International Journal of Logistics Research and Applications*, 15(2):127–146, 2012.

[2] J. R. Montoya-Torres J. J. R. Reyes, E. L. Solano-Charris. The storage location assignment problem: A literature review. *International Journal of Industrial Engineering Computations*, 2019.

[3] Claudia Battista, A. Fumi, Francesco Giordano, and Massimiliano Schiraldi. Storage location assignment problem: implementation in a warehouse design optimization tool. 01 2011.

[4] Valentina Colla and Gianluca Nastasi. *Modelling and Simulation of an Automated Warehouse for the Comparison of Storage Strategies*. 02 2010.

[5] Indadul Khan and Manas Kumar Maiti. A swap sequence based artificial bee colony algorithm for traveling salesman problem. *Swarm and Evolutionary Computation*, 44:428 – 438, 2019.

[6] T. Liu and M. Maeda. An algorithm of set-based differential evolution for traveling salesman problem. In *2014 Joint 7th International Conference on Soft Computing and Intelligent Systems (SCIS) and 15th International Symposium on Advanced Intelligent Systems (ISIS)*, pages 81–86, 2014.

[7] K. Borna, R. Khezri, and C. Yiu. A combination of genetic algorithm and particle swarm optimization method for solving traveling salesman problem. *Cogent Mathematics*, 2, 12 2015.

[8] Matthew Caryl. Travelling salesman problem. `http://www.permutationcity.co.uk/projects/mutants/tsp.html`, Naposledy navštíveno 3. 1. 2021. [online].