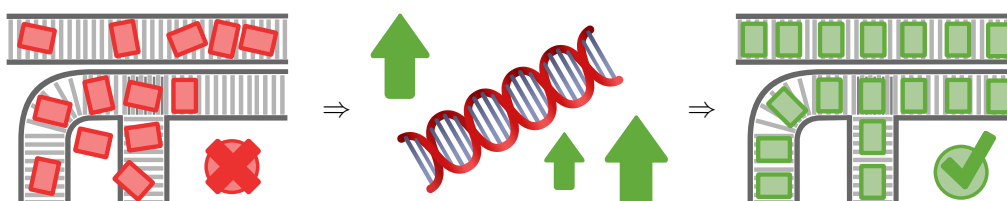


# Warehouse Manager: Nástroj pro pokročilou simulaci a optimalizaci skladových operací

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

**Příložené materiály:** [Demonstrační video](#)

\*[xkocic01@fit.vutbr.cz](mailto:xkocic01@fit.vutbr.cz), Fakulta informačních technologií, Vysoké učení technické v Brně

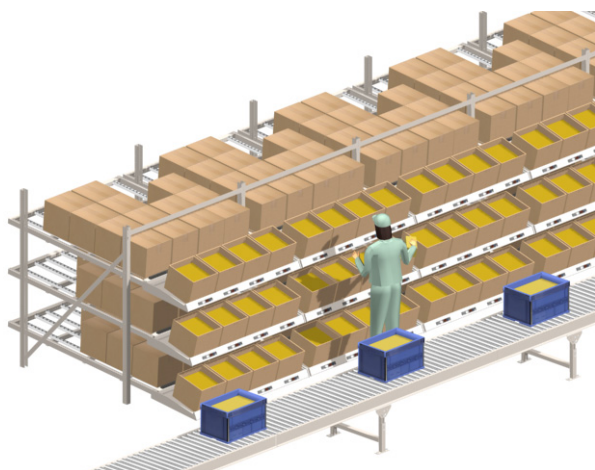
## 1. Úvod

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

16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30



**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<sup>1</sup>.

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

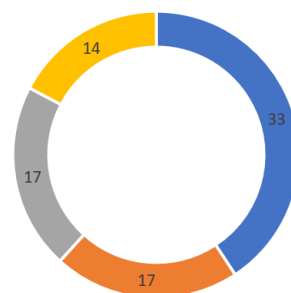
Z množství vědeckých příspěvků, které se v posledních letech zabývaly problematikou SLAP lze usuzovat, že je to velmi aktivní a diskutované téma. Z obsahu těchto příspěvků pak lze usuzovat, že toto téma není zdaleka vyřešené, a existuje zde velký potenciál pro možná zlepšení, a to zejména z pohledu zvýšení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

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



■ Exaktní metody ■ Heuristické metody ■ Meta-heuristické metody ■ Simulace

**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<sup>2</sup>. Hlavním přínosem tedy bude zvýšení propustnosti skladu a to díky hned několika optimalizačním technikám, které jsou v dokumentu kvantitativně i kvalitativně porovnány.

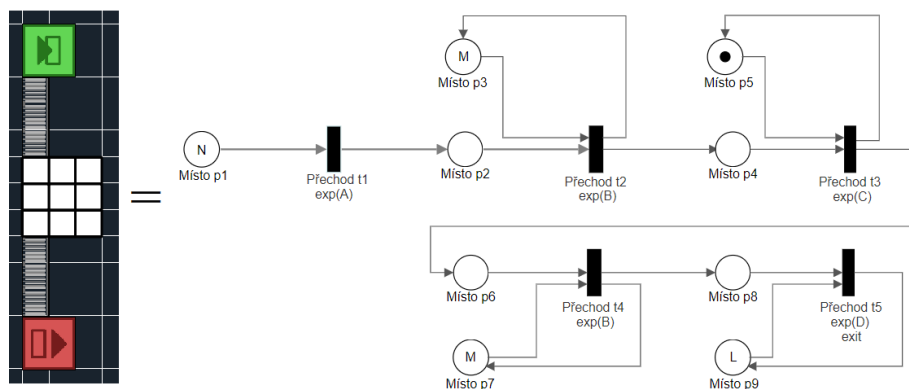
Vytvořené řešení je zcela nezávislé na modelu skladu, ten si lze vytvořit zcela libovolně nebo dle potřeb. Výsledky optimalizace jsou u malých skladů na velmi vysoké úrovni, ačkoli se zvyšující se komplexitou skladu se kvalita optimalizace snižuje. Mimo optimalizaci rozložení produktů ve skladu poskytuje 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 lze jej vidět v grafu 7 jako SLAP a dosáhl

<sup>2</sup>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.

97 zlepšení 33.2%.

98 Genetický algoritmus pro minimalizaci cestované  
99 vzdálenosti ve skladu byl použit v práci [1]. Autoři  
100 optimalizovali přesně definovaný model skladu daný  
101 zákazníkem popsany matematickou funkcí, a podařilo  
102 se jim snížit cestovanou vzdálenost ve skladu při zpra-  
103 covávání objednávek o 28%. To vede ke značnému  
104 zrychlení pickování a dle jejich výpočtů by to ušetřilo  
105 zákazníkov **2700** eur ročně.

### 106 3. Implementace

107 V rámci této práce bylo vytvořeno pět kooperujících  
108 nástrojů, které lze použít jak v textovém, tak grafickém  
109 režimu. Všechny nástroje jsou konfigurovatelné po-  
110 mocí XML souborů či přímo v GUI. Veškeré nástroje  
111 byly implementovány v C++ (standard c++17), pro  
112 grafickou nastavbu byl použit framework Qt verze 5.

#### 113 3.1 Generátor

114 Tento nástroj slouží pro vygenerování dvou vzájemně  
115 korelujících sad zákaznických objednávek. První sada  
116 je použita pro optimalizaci skladu (dále nazývaná jako  
117 trénovací sada) a druhá pro vyhodnocení kvality opti-  
118 malizace (dále nazývána jako testovací sada).

119 Pravděpodobnostní modely, na základě kterých  
120 se generování provádí, jsou Gaussova rozložení –  
121 parametry (střední hodnotu a rozptyl) definuje uživatel.  
122 Generátor je založen na hodnotách **ADU<sup>3</sup>** a **ADQ<sup>4</sup>**.

<sup>3</sup>Average daily units – průměrný denní počet zakoupení.

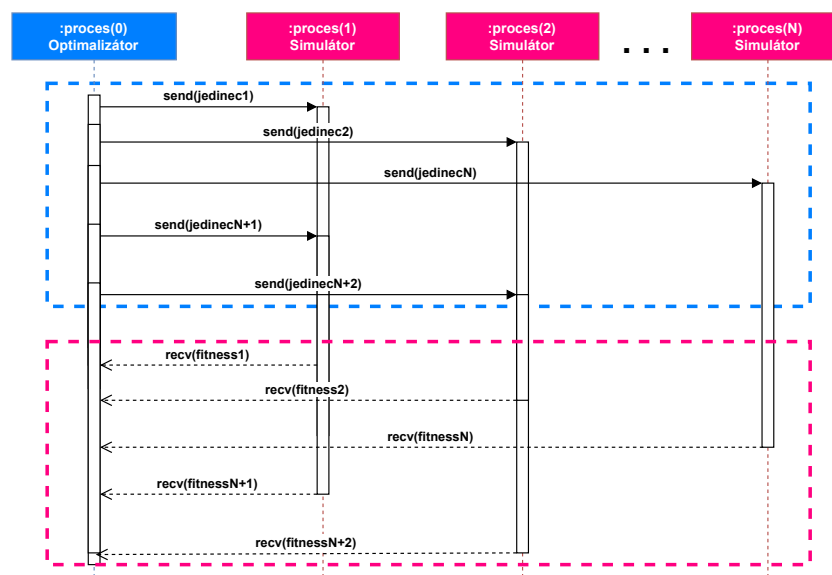
<sup>4</sup>Average daily quantity – průměrná denní zakoupená kvantita.

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

$$p_i = \frac{ADU_i}{\sum_{n=1}^N ADU_n}, \quad (1)$$

z čehož vznikne diskrétní pravděpodobnostní ro-  
zložení. Poté se iteruje přes počet objednávek, které  
chce uživatel vygenerovat. Pro každou takovou ob-  
jednávku 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 pro-  
dukt 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 hod-  
nota 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 ob-  
jednávky. 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.

## 145 3.2 Simulátor

146 Účelem tohoto nástroje je odsimulování zpracování  
147 importovaných či generovaných objednávek na  
148 vytvořeném modelu skladu tak, jako by to byl reálný  
149 skladový systém. Lze jej použít samostatně (např.  
150 pro identifikaci úzkých míst, či pro statistickou  
151 analýzu), avšak jeho hlavní účel je aproximace kval-  
152 ity **nalezeného řešení** v optimalizátoru rozložení pro-  
153 ductů – jinými slovy je použit jako objektivní funkce.

154 Autoři práce [4] zmiňují, že ze všech možných  
155 druhů je nejvhodnější a nejpřirozenější simulace  
156 skladu pomocí diskretních událostí, protože sklad  
157 je v podstatě kolekce entit, které reagují na fixní  
158 diskretní události. Simulátor je tedy založen na  
159 principu diskretní simulace a při implementaci byla  
160 využita knihovna SIMLIB/C++<sup>5</sup>. Simulátor posky-  
161 tuje poměrně komplexní konfiguraci, což umožňuje  
162 rozsáhlé možnosti experimentování (od nastavení  
163 rychlostí pracovníků a dopravníků až po doplňování  
164 produktů viz 3.2.2). Princip je vysvětlen na snímku 3.

### 165 3.2.1 Paralelizace

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

Tento problém byl vyřešen spuštěním několika in- 174  
stancí (procesů) využívající tuto knihovnu, které se 175  
nijak neovlivňují a mohou fungovat souběžně. Před 176  
začátkem optimalizace je tedy vytvořeno  $N$  (konfig- 177  
urovatelné) takových procesů. Účel takto vytvořených 178  
potomků je jednoduchý, a sice provést inicializaci 179  
(objednávek, modelu skladu a před-počítání cest), 180  
očekávat data, provést simulaci a vrátit výsledek sim- 181  
ulace hlavnímu procesu. Po odeslání dat zpět rodiči 182  
potomek opět vstupuje do blokujícího čekání na data 183  
nebo ukončení komunikace a tedy i samotného po- 184  
tomka. Komunikace mezi rodičem a potomky je 185  
následující (viz 4): 186

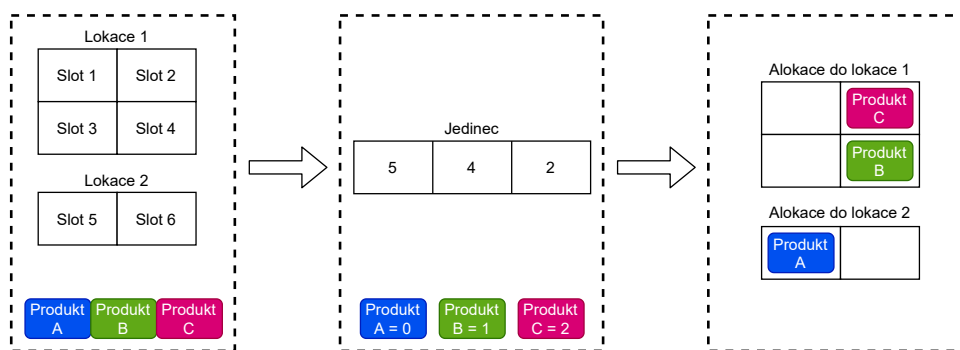
- Od rodiče k potomkům se posílá zakódovaná 187  
alokace produktů do jednotlivých slotů 188  
(celočíslné pole). 189
- Od potomků zpět k rodiči se posílá výsledek 190  
(doba) simulace, a sice hodnota *fitness* (číslo s 191  
plovoucí řádovou čárkou). 192

Jedinci (potažmo jejich simulace) jsou mezi pro- 193  
cesy rozděleny zcela rovnoměrně a zrychlení optimal- 194  
izace pomocí paralelizace simulací bylo velmi značné. 195

Dále bylo zjištěno, že až 30% všech jedinců v 196  
populaci je duplicitních (stejných jako nějaký jiný 197  
jedinec). Byl proto implementován mechanismus 198  
převodu zakódovaných genů jedince na řetězec, a po- 199  
mocí hashovací tabulky bylo zajištěno, že se nebudou 200  
provádět duplicitní simulace ale jedinci se stejnými 201  
geny si pouze překopírují již spočtený výsledek jiného 202  
jedince. To vedlo k ještě většímu zrychlení celého 203  
procesu a optimalizace i opravdu komplexního skladu 204  
bylo možné počítat v řádu maximálně několika dní. 205

<sup>5</sup>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.

### 206 3.2.2 Doplnování produktů

207 Doplnování produktů (ang. *replenishment*) je proces,  
208 při kterém se ze zásobníku produktů doplňují produkty  
209 do slotů, ze kterých se pickují zákaznické objednávky.  
210 Toto se typicky provádí ve chvíli, kdy množství pro-  
211 ductů ve slotu klesne pod určitou (konfigurovatelnou)  
212 úroveň. Tento mechanismus byl implementován za  
213 účelem zvýšení realističnosti simulace skladu.

### 214 3.3 Pathfinder

215 Nástroj pro optimalizaci cesty byl implemen-  
216 tován skrze evoluční algoritmus *MAX-MIN*  
217 mravenčí systém, a nazývá se *pathfinder*. Cílem  
218 tohoto nástroje je nalézt optimální cestu objednávky  
219 skrze sklad, tak, aby urazila co nejkratší možnou  
220 vzdálenost. Vzhledem k tomu, že každá objednávka  
221 potřebuje navštívit jiné lokace, optimální cesta skrze  
222 sklad se zpravidla liší, a proto je nutné optimální cestu  
223 hledat pro každou objednávku samostatně. Grafická  
224 nastavení dokáže mimo tvorbu grafu také zvýraznit  
225 aktuálně nejlepší nalezenou cestu vybrané objednávky  
226 skrze sklad a očíslovat grafické prvky, aby bylo zřejmé,  
227 v jakém pořadí je objednávka navštíví.

### 228 3.4 Optimalizátor rozložení produktů

229 Automatizované sklady jsou mnohdy velmi komplexní  
230 systémy a mají mnohá omezení daná jejich *layoutem*,  
231 způsobem manipulace s produkty, úložnými a pick-  
232 ovacími politikami atd. Optimalizace výkonnosti  
233 takovýchto skladů často vyžaduje přesnou definici je-  
234 jich modelu a nelze jej jednoduše převést na matem-  
235 atický výraz. Vzhledem k tomu, a také k možnosti  
236 uživatele si vlastnoručně vytvořit model skladu, by  
237 bylo velmi obtížné takto obecně vytvořit matematický  
238 popis skladu, proto tato práce pro vyhodnocení kvality  
239 používá simulaci, která odpovídá reálnému fungování  
240 skladu. Myšlenka je taková, že se optimalizací mini-  
241 malizuje objektivní funkce, a tou je simulovaná doba  
242 (tzn. snižuje se doba potřebná ke zpracování všech ob-  
243 jednavek). Doba zde představuje simulační čas, nikoli  
244 reálný.

Pro nalezení optimální distribuce produktů do slotů  
lokací byl jako nejvhodnější přístup vybrán GA (genet-  
ický 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 algo-  
ritmy, a sice: DE (diferenční evoluce), ABC (algo-  
ritmus umělých včelstev) a PSO (optimalizace ro-  
jem částic). Všechny tyto algoritmy pracují stan-  
dardně 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 ob-  
chodního cestujícího), pro který byly tyto redefinice  
v odborných článcích popsány). Trénování opti-  
malizátoru probíhalo na metacentru<sup>6</sup> a optimalizátor  
podporuje ukládání a načítání *vah* modelu.

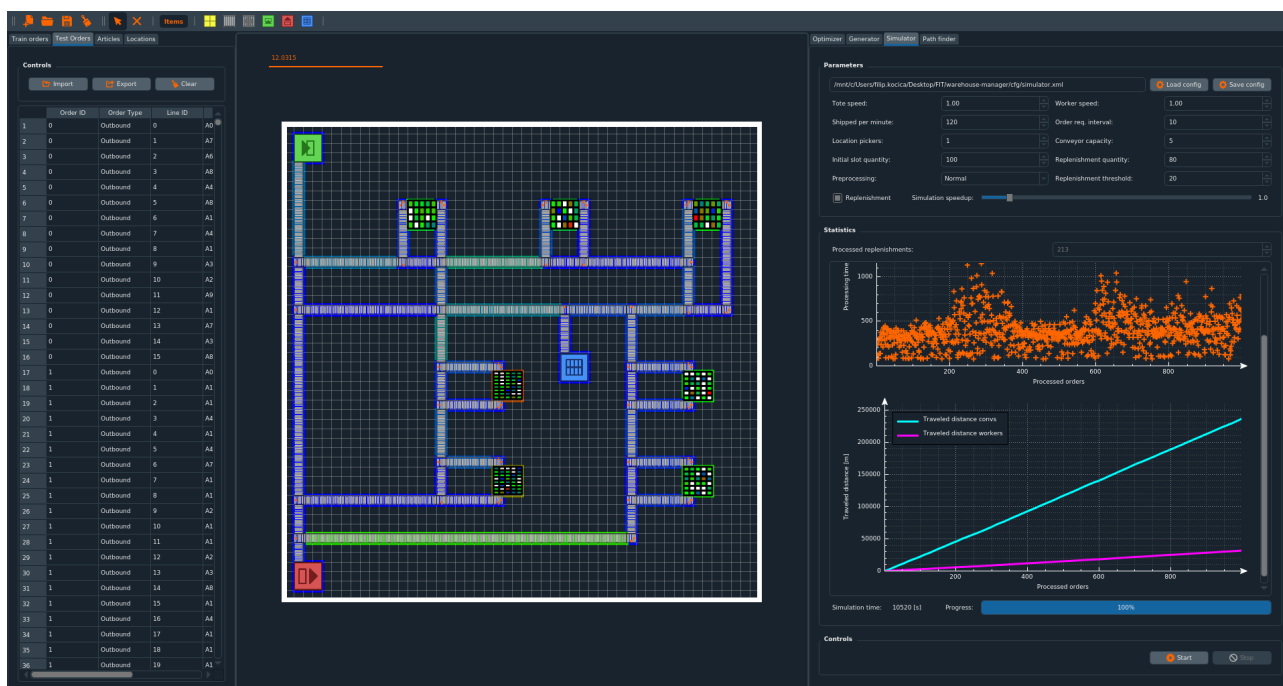
#### 3.4.1 Kódování

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 pro-  
dukt a hodnota na daném indexu reprezentuje slot, do  
kterého je daný produkt alokovan. Příklad znázorňující  
toto kódování lze najít na obrázku 5.

### 3.5 Warehouse manager

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

<sup>6</sup><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 <i>trial</i>	10
Pravděpodobnost křížení	0.6
Pravděpodobnost mutace jedince	0.4
Pravděpodobnost mutace genu	0.2

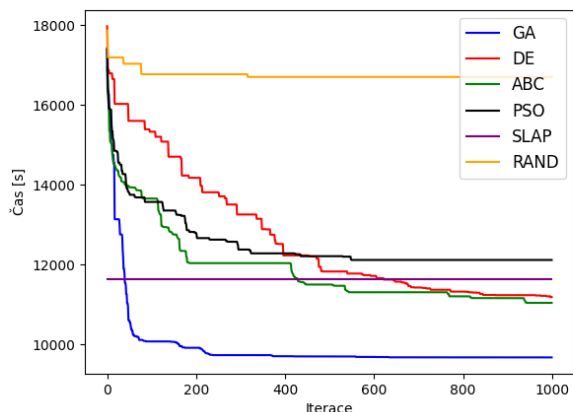
Uprostřed aplikace je plocha určená pro tvorbu a manipulaci s modelem skladu. Cílem této plochy je poskytnout úplný a intuitivní 2D editor poskytující různé druhy skladových prvků a manipulaci s nimi. Ve výsledku editor (mimo jiné) umožňuje:

- 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.

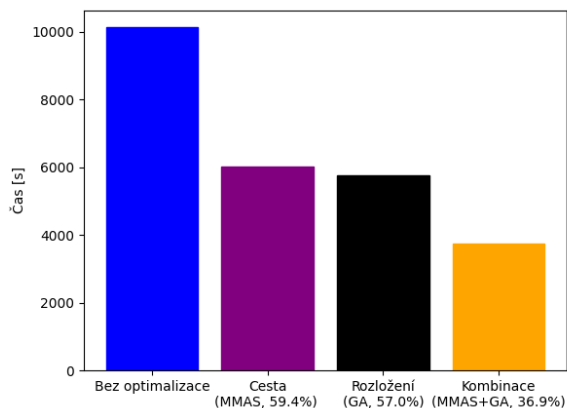
Pro implementaci plochy pro tvorbu modelu skladu byla využita grafická scéna (QGraphicsScene) a grafický pohled. Do grafické scény jsou postupně umísťovány objekty odvozené z grafických prvků, které je rozšiřují o specifické vlastnosti vhodné pro daný use-case.

## 4. Experimenty

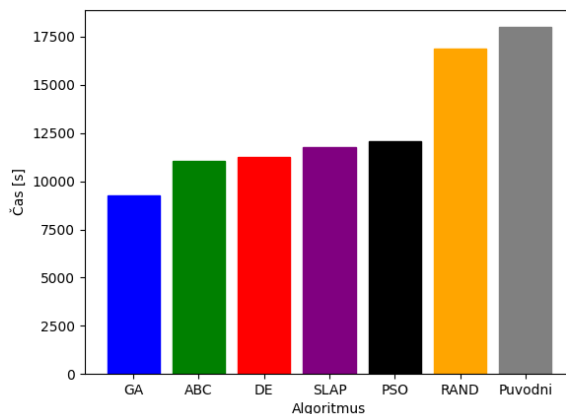
Pomocí experimentů bylo zjištěno, že nejlepších výsledků dosahuje algoritmus GA a byla nalezena jeho



**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 9.** Sloupkový graf porovnávající optimalizaci cesty pomocí MMAS, optimalizaci rozložení produktů pomocí genetického algoritmu a nakonec jejich kombinaci.



**Obrázek 8.** Sloupkový graf vyhodnocení nejlepších trénovaných 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.

## 5. Vyhodnocení

331

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

332

333

### 5.1 Optimalizátor rozložení produktů

334

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:

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

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.

335

336

337

338

339

340

341

342

343

344

345

346

### 5.2 Pathfinder

347

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

348

349

350

319 optimální konfigurace, kterou lze vidět v tabulce 1.

320 Nejúspěšnější experiment vznikl kombinací algo-  
321 ritmu GA a vlastnosti algoritmu ABC. Algoritmus  
322 ABC pro každou včelu (řešení problému) udržuje  
323 hodnotu *trial*, která udává počet kroků algoritmu, ve  
324 kterých se daná včela nezlepšila. Pokud tato hod-  
325 nota dosáhne před-definované hodnoty, je tato včela  
326 nahrazena novou náhodně vygenerovanou včelou.  
327 Problém u GA byl, že se zasekával v lokálních min-  
328 imech a nedokázal najít příliš dobré řešení, proto jsem  
329 jej zkusil doplnit o tuto vlastnost a nalezené řešení  
330 bylo vždy lepší o v průměru o cca. 5.11%.

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, jenž 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 zdroje 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].