



**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**

BRNO UNIVERSITY OF TECHNOLOGY

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**

FACULTY OF INFORMATION TECHNOLOGY

**ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ**

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

**SIMULACE SKLADU A OPTIMALIZACE ROZMÍSTĚNÍ  
PRODUKTŮ ZA ÚČELEM ZVÝŠENÍ PROPUSTNOSTI  
SKLADU**

WAREHOUSE SIMULATION AND PRODUCT DISTRIBUTION OPTIMIZATION FOR INCREASED  
THROUGHPUT

**DIPLOMOVÁ PRÁCE**

MASTER'S THESIS

**AUTOR PRÁCE**

AUTHOR

**Bc. FILIP KOČICA**

**VEDOUCÍ PRÁCE**

SUPERVISOR

**Ing. OLDŘICH KODYM**

BRNO 2021

## Zadání diplomové práce



Student: **Kočica Filip, Bc.**  
Program: Informační technologie Obor: Kybernetická bezpečnost  
Název: **Simulace skladu a optimalizace rozmístění produktů za účelem zvýšení propustnosti skladu**  
**Warehouse Simulation and Product Distribution Optimization for Increased Throughput**

Kategorie: Umělá inteligence

Zadání:

1. Proveďte literární rešerši optimalizačních metod vhodných pro řešení úlohy rozmístění produktů v lokacích ve skladu (storage location assignment problem).
2. Navrhněte a implementujte simulátor skladu včetně generování produkčních dat.
3. Navržený simulátor otestujte na ukázkové úloze.
4. Navrhněte a implementujte jednu nebo více optimalizačních metod vhodných pro řešení dané úlohy.
5. Implementované metody experimentálně otestujte s využitím simulátoru z hlediska propustnosti skladu.
6. Diskutujte dosažené výsledky a možné budoucí rozšíření projektu.

Literatura:

- Zelinka, I. a kol.: Evoluční výpočetní techniky. Principy a aplikace. BEN, Praha, 2009 (CS)
- M. Kosfeld, Warehouse design through dynamic simulation, *1998 Winter Simulation Conference. Proceedings (Cat. No.98CH36274)*, Washington, DC, USA, 1998, pp. 1049-1053 vol.2, doi: 10.1109/WSC.1998.745852.
- Nastasi, G., Colla, V., Cateni, S. et al. Implementation and comparison of algorithms for multi-objective optimization based on genetic algorithms applied to the management of an automated warehouse. *J Intell Manuf* 29, 1545-1557 (2018).  
<https://doi.org/10.1007/s10845-016-1198-x>

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování práce viz <https://www.fit.vut.cz/study/theses/>

Vedoucí práce: **Kodym Oldřich, Ing.**

Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.

Datum zadání: 1. listopadu 2020

Datum odevzdání: 19. května 2021

Datum schválení: 30. října 2020

## Abstrakt

Tato práce řeší problematiku alokace produktů do lokací ve skladu za pomoci moderních meta-heuristických přístupů v kombinaci s realistickou simulací skladu. 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 experimentálního skladu na téměř dvojnásobek – **57%**. 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.

## Abstract

This thesis focuses on the storage location assignment problem using modern meta-heuristic techniques combined with realistic simulation. A graphical tool implemented as part of this work is capable of warehouse model creation, generation of synthetic customer orders, optimization of product allocation using state of the art techniques, extensive warehouse simulation, and a pathfinder capable of finding the shortest path for orders going through the system. The work presents the comparison between different approaches based on many parameters to reach the most efficient allocation of products to warehouse slots. The author conducted tests on an experimental warehouse featuring almost twice the throughput – **57%**. The benefit of this work is a possibility to create model of an already built warehouse and its simulation and optimization, driving impact on the throughput of the warehouse, saving the user's resources, or helping him in planning and bottle-neck identification. Furthermore, this thesis introduces a new approach to warehouse optimization and new optimization criteria.

## Klíčová slova

Sklad, optimalizace, simulace, generátor, objednávka, produkt, pickování, evoluce

## Keywords

Warehouse, optimization, simulation, generator, order, product, picking, evolution

## Citace

KOČICA, Filip. *Simulace skladu a optimalizace rozmístění produktů za účelem zvýšení propustnosti skladu*. Brno, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Oldřich Kodým

# Simulace skladu a optimalizace rozmístění produktů za účelem zvýšení propustnosti skladu

## Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Oldřicha Kodyma. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....  
Filip Kočica  
27. dubna 2021

## Poděkování

Zde bych rád poděkoval svému vedoucímu práce Ing. Oldřichu Kodymovi, konzultantovi Ing. Danielu Chalupovi a 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“.

# Obsah

<b>1</b>	<b>Úvod</b>	<b>3</b>
<b>2</b>	<b>Problematika rozmístění produktů do lokací ve skladu</b>	<b>5</b>
2.1	Motivace pro efektivní optimalizaci skladu . . . . .	5
2.2	Strategie uložení produktů . . . . .	7
2.3	Nástroje pro optimalizaci skladu . . . . .	7
2.4	Simulace skladu . . . . .	8
2.5	Související práce . . . . .	10
<b>3</b>	<b>Evoluční algoritmy</b>	<b>12</b>
3.1	Genetické algoritmy . . . . .	12
3.1.1	Struktura genetických algoritmů . . . . .	13
3.2	Diferenční evoluce . . . . .	17
3.3	Algoritmus umělých včelstev . . . . .	18
3.4	Optimalizace rojem částic . . . . .	19
3.5	<i>MA<math>\chi</math>-MZN</i> mravenčí systém . . . . .	20
3.6	Redefinice jednotlivých metod pro diskrétní prostor . . . . .	21
3.6.1	Problém obchodního cestujícího . . . . .	21
3.6.2	Redefinice genetických algoritmů pro diskrétní prostor . . . . .	22
3.6.3	Redefinice diferenční evoluce pro diskrétní prostor . . . . .	22
3.6.4	Redefinice algoritmu umělých včelstev pro diskrétní prostor . . . . .	23
3.6.5	Redefinice optimalizace rojem částic pro diskrétní prostor . . . . .	25
<b>4</b>	<b>Návrh nástrojů pro simulaci a optimalizaci skladu</b>	<b>26</b>
4.1	Generátor produkčních dat . . . . .	26
4.2	Simulátor skladu . . . . .	27
4.3	Pathfinder . . . . .	27
4.4	Optimalizátor rozložení produktů . . . . .	27
4.5	Warehouse Manager . . . . .	27
<b>5</b>	<b>Implementace nástrojů pro simulaci a optimalizaci skladu</b>	<b>28</b>
5.1	Generátor produkčních dat . . . . .	28
5.1.1	Konfigurace . . . . .	28
5.1.2	Princip fungování a implementace . . . . .	29
5.2	Simulátor skladu . . . . .	30
5.2.1	Konfigurace . . . . .	30
5.2.2	Princip fungování a implementace . . . . .	31
5.2.3	Paralelní simulace . . . . .	31

5.2.4	Řešení problému duplicitních jedinců . . . . .	32
5.2.5	Doplňování produktů . . . . .	32
5.3	Pathfinder . . . . .	33
5.3.1	Konfigurace . . . . .	34
5.3.2	Princip fungování a implementace . . . . .	35
5.4	Optimalizátor rozložení produktů . . . . .	36
5.4.1	Kódování . . . . .	36
5.4.2	Grafická teplotní mapa . . . . .	38
5.4.3	Konfigurace . . . . .	38
5.4.4	Experimenty . . . . .	40
5.4.5	Průběh optimalizace . . . . .	41
5.5	Warehouse Manager . . . . .	42
5.5.1	Layout aplikace . . . . .	43
5.5.2	Tvorba modelu skladu . . . . .	43
5.6	Moduly programu . . . . .	46
<b>6</b>	<b>Vyhodnocení účinnosti vytvořených nástrojů</b>	<b>48</b>
6.1	Optimalizace rozložení produktů . . . . .	48
6.2	Optimalizace cesty . . . . .	49
6.3	Kombinace optimalizací . . . . .	49
<b>7</b>	<b>Závěr</b>	<b>51</b>
	<b>Literatura</b>	<b>52</b>
<b>A</b>	<b>Obsah přiloženého paměťového média</b>	<b>54</b>
<b>B</b>	<b>Manuál</b>	<b>55</b>
<b>C</b>	<b>Plakát</b>	<b>56</b>

# Kapitola 1

## Úvod

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

Sklady se zpravidla optimalizují za účelem rychlejšího odbavování zákaznických objednávek. Každá 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 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. Plynulost a efektivita pickování objednávek, které jsou ovlivněny mimo jiné rozložením produktů, mají zásadní vliv na výkonnost skladu jako celku, a proto jsou často považovány za nejslibnější oblast z hlediska optimalizace skladových operací [5]. Tato práce řeší primárně kombinatorický NP-těžký problém, a sice jakým způsobem rozmístit produkty do slotů lokací ve skladu, aby bylo dosaženo co nejvyšší propustnosti. 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.

V navrženém řešení uživatel nejprve provede vytvoření modelu skladu v grafickém 2D editoru dle jeho potřeb. Poté si vygeneruje data pomocí **generátoru objednávek**, který na základě pravděpodobnostních modelů generuje sady zákaznických objednávek pro trénování a testování. Na základě modelu skladu a vygenerovaných objednávek může následně uživatel provést realistickou **simulaci skladu**, nebo pomocí nástroje **hledáč cest** (dále jako pathfinder) nalézt optimální cestu objednávky skladem pomocí mravenčího algoritmu. Dále je uživatel schopen provést **optimalizaci rozložení produktů** ve skladu za účelem zvýšení propustnosti skladu pomocí čtyř evolučních algoritmů. Veškeré zmíněné funkcionality reprezentované jednotlivými nástroji jsou jednak použitelné samostatně skrze příkazovou řádku, ale také spojeny do jednoho grafického nástroje, nazvaného **skladový manažer**

(dále jako Warehouse Manager). Hlavním přínosem tedy bude zvýšení propustnosti skladu, a to díky hned několika optimalizačním technikám, které jsou v dokumentu porovnány.

Vytvořené řešení je zcela nezávislé na modelu skladu, ten si lze vytvořit zcela libovolně, narozdíl od velké části existujících řešení. Výsledky optimalizace jsou u menších až středních skladů na velmi vysoké úrovni, ačkoli se zvyšující se komplexitou skladu se kvalita optimalizace lehce snižuje. Mimo optimalizaci rozložení produktů ve skladu poskytuje řešení další užitečné funkcionality, jako je například identifikace úzkých míst (tzv. *bottlenecků*) či nalezení nejkratší cesty objednávky skrze sklad.

Kapitola 2 se zabývá obecným popisem problematiky alokace produktů do lokací a souhrnem existujících přístupů. V kapitole 3 lze nalézt popis evolučních algoritmů a jejich redefinici pro diskrétní prostor. Kapitoly 4 a 5 popisují návrh a implementaci pěti nástrojů vytvořených v rámci této práce. Kapitola 6 poté shrnuje dosažené výsledky práce.



## Kapitola 2

# Problematika rozmístění produktů do lokací ve skladu

Tato kapitola pojednává o problematice rozmístění produktů do lokací, anglicky zvané *Storage Location Assignment Problem*, dále pouze zkratka SLAP. V odborné literatuře se lze však setkat s různými variacemi označeními této problematiky, avšak se stejným významem (např.: *Storage Assignment*, *Product allocation/location*, *Slotting*, a tak dále). Při studiu této problematiky jsem vycházel z [3, 5, 8, 9, 11, 17, 18].

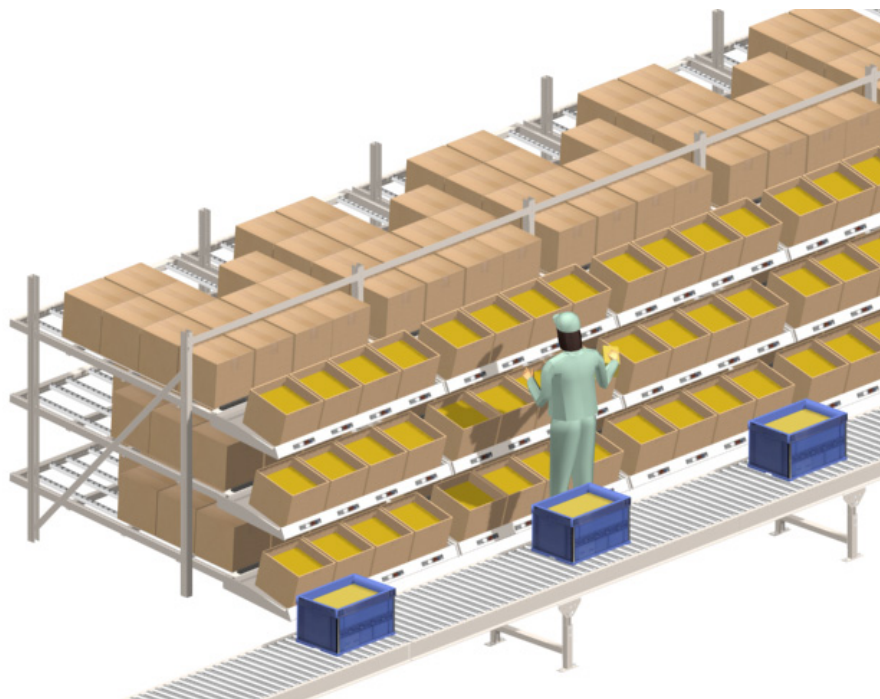
### 2.1 Motivace pro efektivní optimalizaci skladu

Sklady slouží pro dočasné uložení produktů (typicky z produkce), které jsou následně postupně získávány za účelem vyřízení zákaznických objednávek. „Pickování“ zákaznických objednávek je proces při kterém jsou takto uložené produkty systematicky hledány a přemístovány z úložných prostor (přesněji slotů lokací), ve kterých jsou dočasně uloženy, do kartonů, ve kterých jsou následně odeslány k zákazníkům. Zákaznické objednávky typicky sestávají z jednotlivých položek, anglicky zvaných *order line*, vždy představující jeden produkt (v anglické literatuře často označován jako *article*, *product*, *code* či *stock keeping unit*) a jeho požadovanou kvantitu (množství). Způsob organizace produktů a jejich pickování značně ovlivňuje výkonnost celého skladu a tedy ovlivňuje efektivnost celého dodavatelského řetězce. To znamená, že čím rychleji budou produkty napickovány, tím rychleji se zákaznická objednávka vyřídí [5].

Pickování objednávek je jedna z časově nejnáročnějších operací prováděných ve skladu a je spojena s 55% celkových nákladů skladu, a proto se tedy dle vědců jeví jako nevhodnější oblast pro optimalizace skladu [5].

Proces průmyslové automatizace způsobil požadavky na automatizované skladové systémy v mnoha různých odvětvích. Typické činnosti v takovýchto skladech jsou: příjem zákaznických objednávek a jejich seskupování, třídění a plánování v závislosti na aktuálním množství jednotlivých produktů ve skladu, postupné spouštění objednávek a jejich pickování, až po proces odesílání ze skladu k zákazníkům. Proces pickování objednávek sestává z několika částí, které značně závisí na typu skladu, jeho komplexitě a v neposlední řadě na použitém informačním systému. Příklad takového pickování společně s popisem terminologie lze vidět na snímku 2.1 [9, 5].

Z pohledu složitosti se SLAP klasifikuje jako NP-těžký problém, a to vzhledem k množství variací způsobených množstvím produktů a úložných prostor. Vzhledem k tomu, mnoho



Obrázek 2.1: Příklad pickování objednávek pro vysvětlení terminologie<sup>1</sup>. Na obrázku lze vidět lokaci a pickera (pracovník skladu, který vytahuje produkty ze slotů lokace do kartonů objednávek). Způsob fungování je následující: po dopravníku přijede karton, který reprezentuje nějakou objednávku. Uživatel tento karton naskenuje. Systém následně zjistí, které produkty tato objednávka potřebuje, a ukazuje uživateli které produkty (z jakých slotů) má do tohoto kartonu vkládat. Po každém vložení produktu do kartonu uživatel tuto akci potvrdí a pokračuje dalším produktem. Až nasbírá všechny produkty, které jsou na této lokaci potřeba napickovat, systém uživateli řekne, ať karton vloží na dopravník (odkud pokračuje dál), a aby naskenoval další karton, u kterého bude postupovat podobným způsobem.

meta-heuristických a heuristických metod bylo použito za účelem řešení této problematiky. Pokud je počet produktů roven počtu úložných prostor, jedná se o problém QAP<sup>2</sup>. Pokud počet produktů převyšuje počet úložných prostor, jedná se o problém Knapsack [11].

Strategie uložení produktů hraje důležitou roli z pohledu skladových informačních systémů (ang. *warehouse management system*). Tato operace je mnohdy mylně považována za snadnou, ale ve skutečnosti kvůli nejistotě požadavků, variabilitě druhů produktů a potřebě okamžité reakce na změnu na trhu je velmi komplexní a vyžaduje složitá rozhodnutí. Je nutné, aby sklady byly navrženy a řízeny tak, aby byly nákladově efektivní. Náklady na fungování skladu jsou do značné míry určovány již ve fázi návrhu a proto je vhodné před samotnou stavbou skladu věnovat značnou pozornost studiu a analýze budoucího skladu pro dosažení co nejoptimálnějšího skladu pro daný *use-case* (česky případ užití) [3, 11].

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

<sup>2</sup>QAP – *Quadratic assignment problem* – česky problematika kvadratického rozřazení.

Problematika rozmístění produktů do úložných lokací ve skladu se snaží o nalezení co nejefektivnějšího rozřazení jednotlivých produktů do lokací ve skladu za účelem snížení doby potřebné k vyřízení zákaznických objednávek. To následně velmi ovlivňuje KPI<sup>3</sup> [18].

## 2.2 Strategie uložení produktů

Vzhledem k variabilitě parametrů produktů, které mohou být ve skladu ukládány, existují různé strategie (někdy také zvané jako politiky) uložení produktů do úložných prostor skladu. Při použití skladového informačního systému se tyto možnosti ještě rozšiřují, protože informační systém umožňuje značně větší kontrolu nad celým skladem. Základní strategie pro ukládání produktů jsou následující [3][18]:

- **Fixed slot** politika – Každý slot každé lokace skladu má pevně přiřazený produkt, který obsahuje. Toto nastavení je neměnné. Pro tuto politiku není třeba žádný informační systém a je nejvhodnější z pohledu optimalizací (použito v této práci).
- **Random** politika – Jak z jména vyplývá, produkty jsou rozřazeny do slotů náhodně, což z této politiky dělá jednu z nejlhčích metod. Je velmi rozšířená, protože často vyžaduje méně místa než ostatní metody a umožňuje efektivnější využití úložných prostor. Je také často použita pro srovnání při vyhodnocování výkonnosti ostatních politik.
- **Frequency-based politika** – Přiřazuje nejčastěji pickované/kupované produktu do slotů, které jsou nejbližší ke vstupu a výstupu ze skladu, za účelem snížení celkové doby zpracování objednávek (použito v této práci pro porovnání).
- **Class-based politika** – Je kompromis mezi jednoduchostí náhodné politiky a přesností (komplexností) politiky založené na frekvenci pickování.

Toto je výčet pouze základních politik pro ukládání produktů. Tyto politiky lze mezi sebou dále kombinovat a vytvářet tak složitější, ale účinnější politiky.

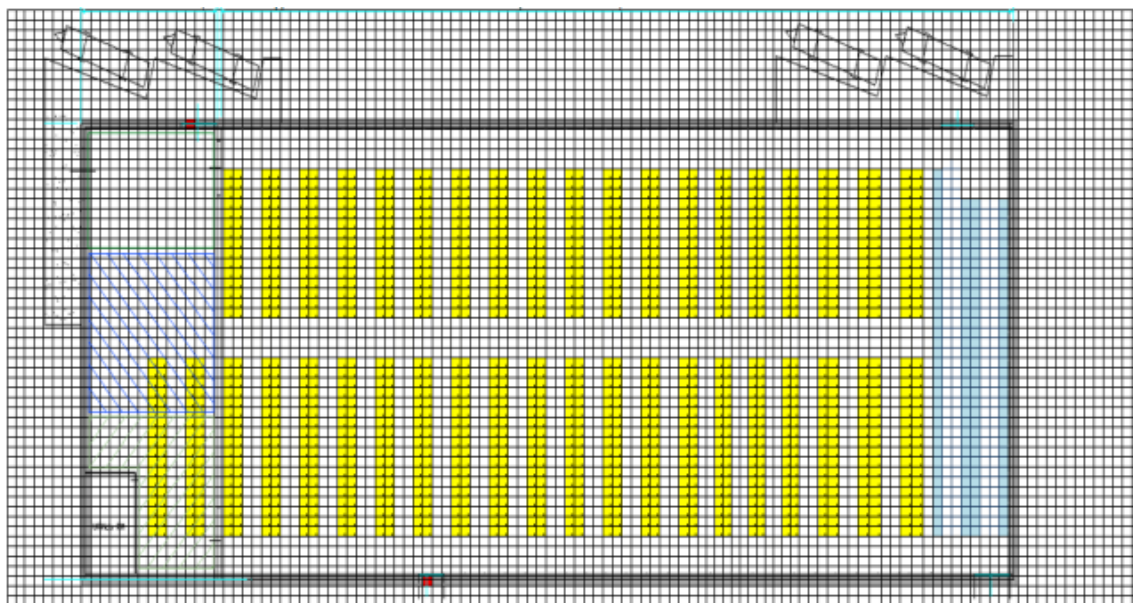
## 2.3 Nástroje pro optimalizaci skladu

Při studiu této podkapitoly jsem vycházel z práce [3]. Nástroj pro optimalizaci skladu by měl být schopen plnit tyto úlohy:

- Vytvoření modelu skladu – Tj. pozice vstupů, výstupů, jednotlivých lokací (včetně množství obsažených slotů), apod. Způsob, jakým jsou jednotlivé zařízení ve skladu rozmístěny se označuje jako layout skladu.
- Datovou analýzu – Na základě (typicky historických) dat – např. objednávek zákazníků – poskytovat agregované či na pouhý pohled neviditelné informace, typicky využité jako podpora při rozhodování.
- Podporu při rozhodování – Pomocť uživatelům nástroje při rozhodnutích, jako například jak efektivně rozmístit jednotlivé prvky skladu či produkty do lokací.

---

<sup>3</sup>KPI – *Key performance indicators* – česky výkonnostní klíčové indikátory



Obrázek 2.2: Na obrázku je možné vidět příklad jednoduchého nástroje pro tvorbu modelu skladu, a také v něm vytvořeného modelu skladu. Obrázek převzat z práce [3].

- Kontrola a řízení zboží na skladě – sem spadá mj. také doplňování produktů (ang. *replenishment*).

Snadno upravitelný layout skladu pak může uživateli pomoci při takzvaných *what-if* (v překladu co-kdyby) a *as-is* (jak-je) analýzách [3]:

- Analýza *what-if* – Může být provedena na základě virtuálního přerovnění layoutu skladu pomocí konfiguračního nástroje a opětovného vyhodnocení a porovnání KPI.
- Analýza *as-is* – Může být provedena na základě **již existujícího** layoutu skladu za účelem vyhodnocení KPI.

Příklad toho, jak může vypadat takový nástroj pro vytvoření virtuálního modelu skladu lze vidět na obrázku 2.2. V závislosti na možnostech nástroje je uživatel schopen si sám jednoduše vytvořit virtuální reprezentaci svého existujícího skladu (měřítko vůči reálnému světu, pozice lokací a slotů, vstupů a výstupů skladu, či dopravníků propojující jednotlivé komponenty). Tato reprezentace může uživateli pomoci při rozhodování za pomoci zmíněných analýz.

## 2.4 Simulace skladu

Automatizované sklady jsou velmi komplexní systémy. Často mají mnohá omezení daná jejich layoutem, způsobem manipulace s produkty (dopravníky, vysokozdvíhné vozíky, ...), úložnými a pickovacími politikami, a tak dále. Optimalizace výkonnosti takovýchto skladů často vyžaduje přesnou definici jejich modelu. Zmíněné praktiky jako ukládání či pickování produktů nelze jednoduše převést na matematické výrazy, které by šly optimalizovat standardním způsobem. Optimalizace takovýchto systémů je navíc často více-objektivní (např. co nejkratší doba pickování a zároveň co nejvyšší úspora místa), kdy se optimalizuje

více parametrů zároveň a hledá se množina možných řešení která představuje kompromisy (ang. *tradeoff*) mezi různými požadavky. Z toho vyplývá, že lepší způsob jak se vypořádat s řešením tohoto problému je pomocí programovacích technik, což de-facto znamená vytvořit simulátor skladu, odpovídající pokud možno co nejvíce reálnému modelovanému systému [9].

Skladové systémy jsou vzhledem k jejich důležitosti v dodavatelském řetězci a potřebě optimalizace (resp. velké úspoře prostředků při jejich optimalizaci) velmi často předmětem simulování. V nejedné práci zabývající se problematikou SLAP se objevují nástroje pro simulaci skladu (příjem, pickování a odesílání objednávek, atp.). Tyto nástroje jsou nejčastěji založeny na diskretních událostech. V posledních letech se však také objevují řešení založená na agentech. Simulace bývá zpravidla využívána za účelem sledování změn chování při použití různých strategií či konfigurací. Simulátor skladu byl využit například za účelem porovnání ujetých vzdáleností při použití různých strategií uložení produktů v automatizovaném prostředí skladu či vlivu použité alokace slotů na proces manuálního pickování objednávek [11].

Práce [9] definuje důležitost simulace skladu následujícími úkoly, jež dokáže plnit:

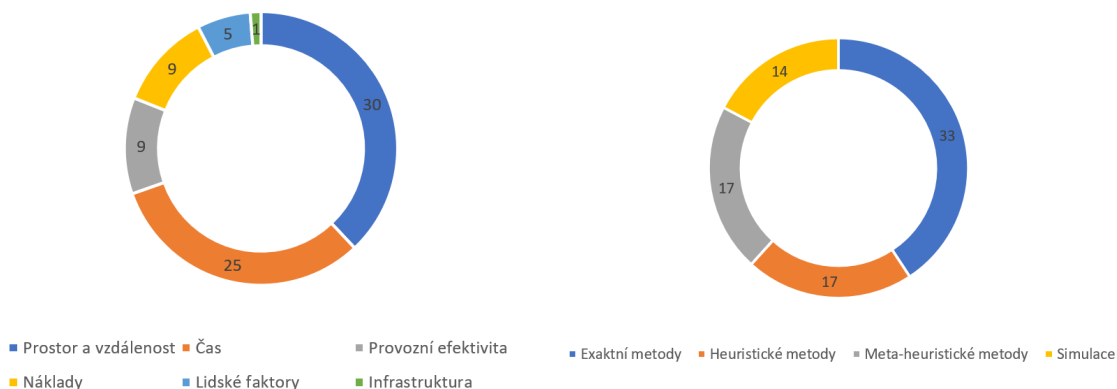
- Poskytnutí tzv. *proof-of-concept* (návrh konceptu).
- Analýza dopadů na potencionální změny v existujícím systému.
- Evaluace výkonnosti skladu již ve fázi návrhu, a to při různém zatížení.
- Optimalizace parametrů skladu (layout, úložné a pickovací politiky, ...).
- Analýza možných „úzkých bodů“ (ang. *bottleneck*).
- Odhad neměřitelných proměnných a kvantit.
- Průzkum a testování nových skladových politik.
- Plánování kapacit skladu.
- Zodpovězení *what-if* otázek (analýza).

Přístupy k simulaci skladu jsou tři. Buď lze využít existujících grafických aplikací, frameworků/knihoven či specializovaných programovacích jazyků. Grafické aplikace sice nevyžadují nutnost psaní kódu a vytvoření systémů je poměrně jednoduché, avšak jsou zpravidla velmi komplexní, drahé a nemusí poskytovat všechny potřebné funkcionality. Patří sem např. AutoMod<sup>TM</sup> nebo Siemens Tecnomatix<sup>TM</sup>. Softwarové knihovny poskytují programátorům před-připravené třídy a funkce, které jim pomohou k vytvoření vlastního simulátoru a jsou zpravidla založeny na simulaci diskretními událostmi. Nevýhodou těchto knihoven je, že většina z nich je orientovaná na simulaci počítačových sítí a ve většině případů je nelze použít pro simulaci skladu (např. OMNeT++ a Ns2). V poslední řadě se používají specializované programovací jazyky mezi které patří např. SIMSCRIPT a SIMULA. Tyto jazyky ulehčují modelování problému pomocí sady před-připravených konstrukcí a instrukcí a jsou často zakomponovány v grafických aplikacích zmíněných výše. Mají však jistá omezení podobně jako softwarové knihovny [9].

V této práci je využit druhý přístup a sice softwarová knihovna, která je zdarma k použití, kompatibilní se zbytkem C++ programu a flexibilní.

		Použitá metoda			
		Exaktní	Heuristická	Meta-heuristická	Simulace
<b>Optimaliz. kritérium</b>	Vzdálenost, prostor			[17, 5]	[9]
	Čas	[3]	[8]		
	Provozní efektivita			[18]	
	Náklady				
	Lidské faktory				
	Infrastruktura				

Tabulka 2.1: Veškeré studované práce klasifikované podle použité metody a optimalizovaného kritéria. Práce by se daly klasifikovat na základě mnoha dalších kritérií, ale pro přehlednost byly zvoleny pouze dvě hlavní, a sice metoda a optimalizované kritérium.



Obrázek 2.3: Graf udávající optimalizační kritéria pro řešení SLAP v odborné literatuře a jejich četnost. Vytvořeno na základě dat z [11].

Obrázek 2.4: Graf udávající optimalizované výkonnostní indikátory (KPI) skladu v odborné literatuře a jejich četnost. Vytvořeno na základě dat z [11].

## 2.5 Související práce

Současná odborná literatura zabývající se problematiku SLAP lze rozdělit do různých kategorií. Nejdůležitější jsou však dvě, a sice použitý přístup a optimalizovaná kritéria, viz 2.3 a 2.4. Tato práce kombinuje meta-heuristiku se simulací a optimalizovaným kritériem je čas zpracování sady objednávek. V tabulce 2.1 lze vidět veškeré studované práce zabývající se problematikou SLAP klasifikované dle těchto kritérií [11].

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% oproti původnímu stavu. Tento princip byl pro porovnání využit i v této práci a lze jej vidět v grafu 6.1 jako Battista a spol. Při experimentech v této práci dosáhl zlepšení 33.2%.

Genetický algoritmus pro minimalizaci ujeté vzdálenosti ve skladu byl použit v práci [5]. Autoři optimalizovali přesně definovaný model skladu daný zákazníkem popsany matema-

tickou funkcí, a podařilo se jim snížit cestovanou vzdálenost ve skladu při zpracovávání objednávek o 28%. To vede ke značnému zrychlení pickování.

Řešení problematiky SLAP pomocí meta-heuristického přístupu založeného na genetických algoritmech se objevilo také v práci [18]. Byla zde vytvořena matematická objektivní funkce, která přesně popisovala rozložení skladu. V této práci autoři počítali také s doplňováním produktů a soustředili se na zjištění důležitosti rozložení zátěže mezi jednotlivé pickery, které je dle nich esenciální pro správnou optimalizaci.

Autoři práce [9] se kvůli složitosti a možnému zavedení chyb či ignorování přepisu skladu na matematickou funkci rozhodli využít pro vyhodnocení simulátor skladu. Autoři zmiňují, že ze všech možných kombinací druhů simulace skladu je nejvhodnější a nejpřirozenější simulace pomocí diskrétních událostí, protože sklad je v podstatě kolekce entit, které reagují na fixní události (jako je například pickování objednávek). Dále byla v rámci práce provedena případová studie existujícího skladu a implementován simulátor daného skladu. Výsledkem každé simulace byl soubor obsahující veškeré KPI skladu, které mohli být dále jednoduše použity pro srovnávání a podařilo značně zvýšit množství uložených produktů.

Heuristický přístup byl použit v práci [8]. Byl vytvořen komplexní matematický model skladu a pomocí celo-číselného programování byl optimalizován čas cesty po skladu. V práci byl implementován heuristický vyhledávací algoritmus tabu a pro malé problémy našel algoritmus vždy optimální řešení.

Podobně jako ve zmíněných pracích je i zde experimentováno se základním i genetickým přístupem. Navíc je práce doplněna i o jiné optimalizace a umožňuje uživateli nastavit si vlastní sklad a strukturu objednávek, všechno v rámci grafické aplikace.

## Kapitola 3

# Evoluční algoritmy

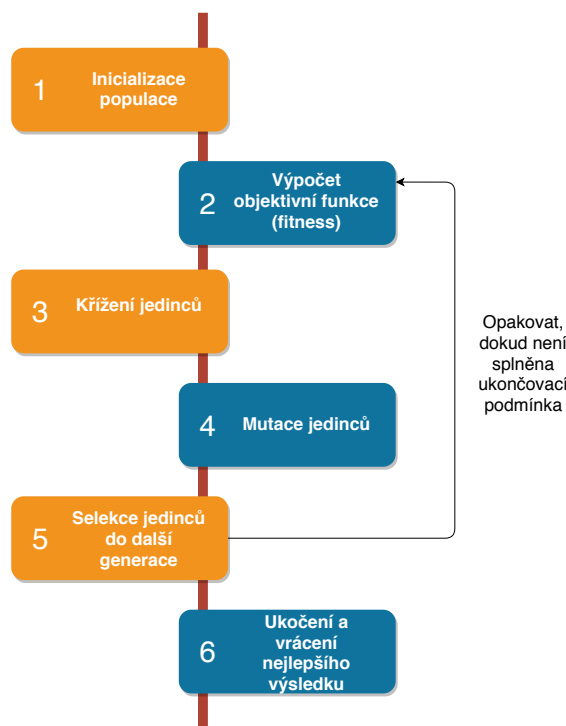
Tato kapitola pojednává o obecném popisu heuristických vyhledávacích a optimalizačních algoritmů inspirovaných přírodou. Jmenovitě se jedná o genetické algoritmy, diferenční evoluci, optimalizaci rojem částic a algoritmus umělých včelstev. Na konci je v krátkosti popsán známý kombinatorický problém obchodního cestujícího a jsou popsány redefinice výše zmíněných evolučních algoritmů pro tento problém řešený v diskrétním prostoru s několika omezujícími podmínkami.

### 3.1 Genetické algoritmy

Při studiu genetických algoritmů (ang. *Genetic Algorithms* – zkratka GA) jsem vycházel z prací [1, 14, 2, 5]. Genetické algoritmy jsou heuristické vyhledávací nebo optimalizační algoritmy inspirované Darwinovským principem evoluce skrze přirozený výběr. Genetický algoritmus však funguje na vysoké úrovni abstrakce evolučních procesů za účelem „evoluce“ řešení pro zadané problémy. Snaží se optimalizovat (tedy minimalizovat či maximalizovat) objektivní funkci, a to za pomoci přechodu z jedné sady chromozomů na novou a to za pomoci genetických operátorů. Genetické algoritmy byly vytvořeny Johnem Hollandem v roce 1970 za účelem nalezení řešení problémů, které byly výpočetně neřešitelné. Po uvedení se dočkaly velké popularity a rychlého vývoje a byly aplikovány k řešení celé řady problému v různých odvětví jako je věda či průmysl. V dnešní době jsou genetické algoritmy stále velmi aktivní a rostoucí oblast výpočetní inteligence, kam patří mimo jiné například také umělé neuronové sítě [1].

Každý genetický algoritmus pracuje nad populací umělých chromozomů (chromozom je také označován jako jedinec populace), kde každý chromozom představuje jedno řešení problému. Chromozom je konečný řetězec (často binární) a má hodnotu *fitness*, což je desetinné číslo, které udává jak dobrý daný chromozom je. Na začátku výpočtu se provádí inicializace populace, kde se pomocí uniformní náhodné funkce vygenerují hodnoty pro každého jednotlivce v populaci. Dále genetický algoritmus provádí selekci a kombinaci jednotlivců, založenou na velikosti jejich hodnoty *fitness* (dle toho zda se jedná o minimalizační či maximalizační problém), za účelem vytvoření nové generace populace, která bude poskytovat lepší řešení daného problému. Tento proces přirozeného výběru je opakován a nové generace jsou vytvářeny až do chvíle, kdy je splněno kritérium pro zastavení (nalezení optimálního řešení nebo dosažení jistého počtu iterací) [1].





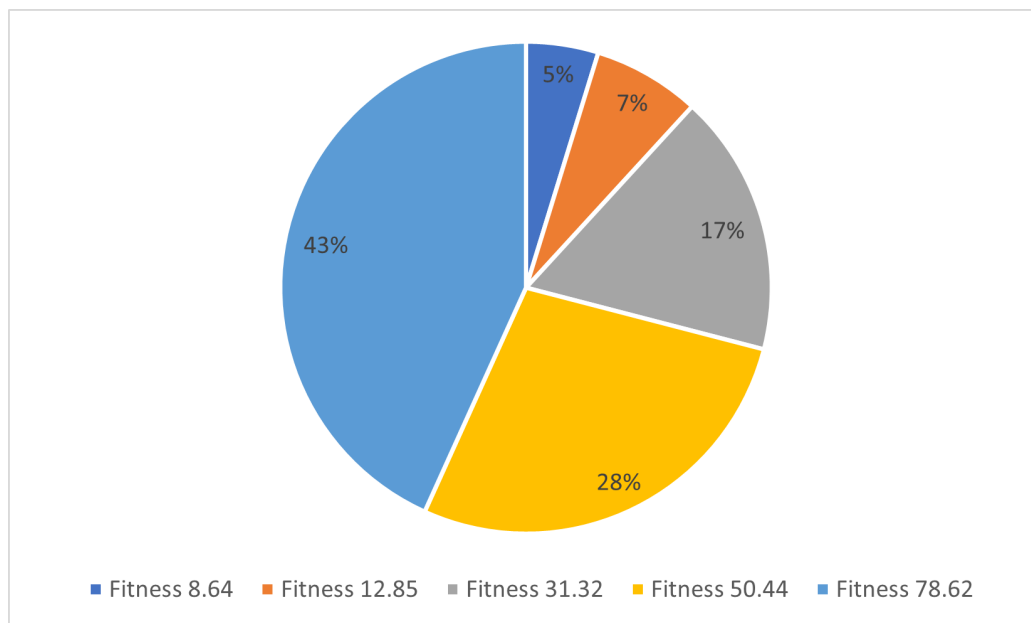
Obrázek 3.1: Princip výpočtu genetického algoritmu. Vytvořeno na základě dat z [1].

### 3.1.1 Struktura genetických algoritmů

Genetický algoritmus je velmi modulární (skládá se z několika odlišných částí), což umožňuje znovupoužití jednotlivých částí v jiných genetických algoritmech, usnadňující jejich implementaci. Hlavní části genetického algoritmu jsou kódování genů a chromozomů, fitness funkce (objektivní funkce), křížení jedinců (ang. *crossover*), mutace (ang. *mutation*) a selektce do další generace na základě fitness funkce, jak lze vidět na obrázku 3.1. Za pomoci těchto operátorů genetický algoritmus hledá vhodnější řešení pomocí evoluce a to skrze navazující iterace zvané jako generace [1, 2].

#### Kódování chromozomů a genů

Jak již bylo zmíněno, genetický algoritmus modifikuje populaci chromozomů za účelem dosažení lepších jedinců. Chromozomy jsou abstrakcí DNA chromozomů a jsou reprezentovány jako řetězce (často binární), které představují řešení daného problému. Jednotlivé části chromozomu jsou v literatuře nazývány geny a chromozom je tedy řetězec genů určité (konečné) délky. Pozice genu v chromozomu je typicky označována jako *locus*. Hodnota, které gen nabývá se nazývá „*allele*“ hodnota. Řešený problém je vždy takzvaně „zakódován“ do chromozomů a interpretace této kódované reprezentace se liší problém od problému. Toto je jedna z hlavních výhod genetických algoritmů, a to sice že podobné reprezentace lze použít pro množství různých problémů, umožňující vývoj společných modulů a usnadňující aplikaci genetických algoritmů na nové, zatím neřešené problémy [1, 2].



Obrázek 3.2: Vizualizace příkladu selekce pomocí metody ruleta s pěti jedinci, každý s různou hodnotou fitness. Největší šanci (43%) na výběr do další generace populace má jedinec s hodnotou fitness 78.62.

### Fitness funkce

Fitness funkce (objektivní funkce) je výpočetní prostředek, který slouží pro ohodnocení kvality chromozomu vzhledem k jeho schopnosti řešit daný problém, což mimo jiné určuje také pravděpodobnost, s jakou bude daný chromozom vybrán do další generace. Optimální hodnota této funkce se bude lišit v závislosti na řešeném problému, zda se jedná o minimalizační či maximalizační úlohu. Fitness funkce je opět závislá na řešeném problému a úzce spojena s kódováním chromozomu, protože značná část interpretace (významu) chromozomu je zakódovaná právě do této fitness funkce [1].

### Selekce jedinců

Selekční operátor slouží pro usměrňování evoluce chromozomů. Do další generace je z populace vybrán jen stanovený počet jedinců, a to, kteří jedinci budou vybráni, závisí zejména na hodnotě fitness jednotlivých prvků (což je diskriminátor kvality jedinců) a z části také na zvoleném selekčním algoritmu. Za účelem zachování diverzity nejsou z populace zpravidla vybráni jen ti nejvhodnější jedinci, ale i jedinci s horší hodnotou fitness (tzn. ne tak dobrá řešení). Jedinci s lepší fitness však mohou být zvýhodňováni, aby nebyl výběr čistě náhodný, ale do další generace se tak dostaly pravděpodobněji ta lepší řešení. Selektce se často provádí s nahrazováním, tzn. lepší chromozomy mohou vybrány vícekrát a dokonce být kombinovány sami se sebou. Zde je přehled několika nejznámějších selekčních metod [1, 2, 14]:

- **Roulette wheel method** (metoda rulety) – Je to jedna z nejpoužívanějších metod pro selekci v genetických algoritmech. Zde pravděpodobnost, že bude chromozom vybrán do další generace odpovídá proporčně hodnotě fitness tohoto chromozomu. Tzn. že pravděpodobnost je vypočtena jako hodnota fitness daného chromozomu dělena suma hodnoty fitness všech chromozomů v populaci. V případě, že se jedná o minimalizační

problém, by se častěji vybíraly horší řešení, proto je nutné před provedením této metody invertovat hodnoty fitness jednotlivých chromozomů (viz příklad 3.2).

- **Rank method** (metoda „pořadí“) – Tato metoda v podstatě funguje na stejném principu jako metoda rulety, avšak odstraňuje její největší problém. Metoda rulety má problémy v případě, že hodnota fitness jednotlivých prvků se hodně liší (tzn. pokud jeden chromozom bude zabírat většinu rulety, ostatní nemají téměř žádnou šanci na výběr). Metoda řeší zmíněný problém tak, že seřadí všechny chromozomy na základě jejich fitness, a postupně jim přiřazuje čísla od 1 po  $N$ . Tzn. nejvhodnější chromozom bude mít hodnotu  $N$ , zatímco nejméně vhodný chromozom bude mít hodnotu 1. Vzhledem k tomu, že tato metoda nezvýhodňuje lepší řešení tak výrazně, má pomalejší konvergenci.
- **Elitism method** („elitní“ metoda) – V době, kdy se vytváří nová generace populace, rovnou překopíruje nejlepší (nebo několik) řešení. To zaručí že algoritmus neztratí nejlepší nalezené řešení, což výrazně zrychluje konvergenci genetického algoritmu.
- **Tournament method** (metoda turnaje) – Metoda vybere dva (obecně  $N$ ) chromozomů a vybere ten s nejvyšší hodnotou fitness.
- **Truncation method** (metoda „zkrácení“) – Tato metoda vybere z populace náhodně jeden chromozom (s uniformním rozložením pravděpodobnosti).

### Křížení jedinců

Vytváření nových jedinců (následníci rodičů) do další generace je v genetických algoritmech prováděno za pomoci sady před-definovaných operátorů. Tyto operátory na vstupu přijímají dva jedince z aktuální generace (rodiče) a produkují na výstup dva nové jedince (potomky). Biologická analogie je kombinování genetického materiálu (DNA), jež se projevuje při reprodukci živočichů. Vzhledem k tomu, že ve většině případů jsou jedinci pro křížení vybíráni na základě jejich fitness hodnoty, je pravděpodobné že jejich kombinací vznikne lepší chromozom. Operátor křížení je ve své podstatě nedeterministický, protože se provádí pouze s určitou pravděpodobností. Také výsledek tohoto operátoru je nedeterministický, protože je založen na stochastické funkci. Operátor křížení má jeden parametr, a to je tzv. *crossover rate*, reprezentující jak často se bude křížení provádět. Při potencionálním křížení se tedy vygeneruje číslo v intervalu  $[0, 1]$  s uniformním rozložením, pokud je hodnota *crossover rate* vyšší než vygenerované číslo, tak se křížení provede, jinak nikoli. Zde je přehled několika nejznámějších operátorů křížení [1, 14]:

- **One-point crossover** (operátor křížení jednoho bodu) – S uniformní pravděpodobností se náhodně vybere číslo  $k$  z intervalu 1 až  $N$ , kde  $N$  je délka chromozomu. Poté se vytvoří potomci takto: První potomek obsahuje řetězec genů 0 až  $k$  z prvního rodiče a zbytek z druhého rodiče. Druhý potomek obsahuje řetězec genů 0 až  $k$  z druhého rodiče a zbytek z prvního rodiče.
- **Two-point crossover** (operátor křížení dvou bodů) – Funguje na stejném principu jako předchozí operátor, ale místo jednoho bodu vybírá dva náhodné body. Potomci poté nesestávají z jedné části z jednoho rodiče a z jedné části z druhého rodiče, nýbrž ze dvou částí z jednoho rodiče a jedné (prostřední) části z druhého rodiče. Tento operátor lze obecně použít na  $N$  bodů (*N-point crossover*).

- **Uniform crossover** (uniformní operátor křížení) – Vytváří potomky tak, že prochází rodiče a na každé pozici uniformně vybírá rodiče, ze kterého se použije hodnota allele.

## Mutace jedinců

Oproti operátoru křížení, který na základě dvou rodičů vytváří dva potomky, operátor mutace upravuje pouze jednoho jedince. Tyto změny jsou typicky malé, jsou prováděny náhodně a jsou typicky prováděny až po procesu křížení jedinců. Podobně jako u křížení, je i zde definován operátor udávající pravděpodobnost, s jakou bude jedinec mutován (ang. *mutation rate*). Tento operátor funguje stejně jako operátor uvedený u křížení jedinců. Dále je ale definována pravděpodobnost mutace jednotlivých genů. Pravděpodobnost mutace bývá však relativně malá, obzvláště ve srovnání s pravděpodobností křížení. Lze provést například pomocí invertování bitů v binárním řetězci [1, 14].

## Proces evoluce

Jak již bylo zmíněno, na počátku evoluce je celá populace (všichni jedinci) náhodně inicializovaná. Následně je vyhodnocena fitness funkce pro každý chromozom. Následně se pomocí genetických operátorů vytváří nová generace následníků. Tento proces se skládá z několika částí, které jsou detailněji popsány výše. První je proveden proces výběru jedinců pro mutování. Takto vybraní jedinci jsou kříženi mezi sebou. Z tohoto procesu vznikne stejný počet potomků, jako do něj vstupuje rodičů a následně jsou někteří z jedinců mutováni. Tito jedinci vstupují do další iterace jako nová generace. Takovýto proces vytváření nových generací je opakován až do doby, kde je buďto splněna podmínka optimalizace/hledání (např. splnění všech podmínek), nebo je dosažen maximální počet generací. Existuje množství různých schémat, jak může genetický algoritmus fungovat. Zde je příklad několika nejběžnějších [1, 2]:

- **Úplné nahrazení** – V tomto schématu jsou po každé iteraci nahrazeni pomocí genetických operátorů všichni jedinci svými následníky.
- **Ustálené schéma** – Nová generace je vytvořena generováním jednoho nového následníka každou generaci, který nahrazuje nejméně vhodného jedince předchozí populace.
- **Nahrazení s elitní částí** (nejrozšířenější použití) – Téměř úplné nahrazení jedinců do další generace (skrze genetické operátory), ale jeden nebo dva jedinci (s nejlepší hodnotou fitness) jsou přesunuti beze změny. Toto schéma zaručuje, že nebudou ztraceny doposud nejlepší výsledky skrze nedeterministický (náhodný) výběr.

Tabulka 3.1: Shrnutí genetických algoritmů. Převzato z [2].

Reprezentace	Různé druhy, např. bitové řetězce
Křížení	Různé druhy, např. jednobodové křížení
Mutace	Různé druhy, např. inverze bitů
Selekce rodičů	V závislosti na fitness (např. metoda rulety)
Selekce následníků	Generační (následník nahrazuje rodiče)

## 3.2 Diferenční evoluce

Při studiu diferenční evoluce (ang. *Differential Evolution* – zkratka DE) jsem vycházel z [15, 2]. Diferenční evoluce v teorii vychází z genetického algoritmu. Je založena na populaci jedinců a umožňuje provádět tři typy operací s jedinci, a sice: křížení, mutaci a selekci do další generace. Tato metoda je používána k optimalizaci nelineárních funkcí ve spojitém prostoru a pracuje tedy s reálnými čísly. Populace je tvořena jedinci (kandidátní řešení), reprezentovanými vektory reálných čísel  $\bar{x} \in \mathbb{R}^n$ , v diferenční evoluci také nazývaných jako cílové vektory. Populace je na začátku náhodně inicializována, je vypočtena hodnota fitness pro každého jedince a poté je prováděna optimalizace. Odlišujícím faktorem je že pořadí jedinců v populaci nezávisí na jejich hodnotě fitness a odklonění od použití klasických operátorů evolučních algoritmů [15, 2].

### Mutace

Algoritmus používá tzv. diferenční mutaci, kde nové kandidátní řešení  $\bar{v}'$  je získáno přičtením vektoru váženého rozdílu dvou náhodně vybraných jedinců populace:

$$\bar{v}' = \bar{v} + F(\bar{a} - \bar{b}), \quad (3.1)$$

kde  $F$  reprezentuje faktor škálování, což je reálné číslo, které slouží pro kontrolu rychlosti evoluce. Hodnoty  $\bar{a}$  a  $\bar{b}$  jsou již zmínění dva náhodně vybraní jedinci populace. Dalším operátorem je operátor křížení, který slouží ke zvýšení diverzity populace [2].

### Křížení

V diferenční evoluci se používá uniformní křížení s parametrem  $C_r \in [0, 1]$ , který udává pravděpodobnost, s jakou bude (pro jakoukoli pozici v aktuálním rodiči) hodnota *allele* rodiče zahrnuta do potomka, oproti křížení v genetických algoritmech, kde pravděpodobnost udává zda budou daní rodiče vůbec kříženi, či nikoli. Nový cílový vektor  $\bar{x}'$  je získán křížením následovně:

$$\bar{x}'_k = \begin{cases} \bar{v}_k, & \text{pokud } (U \leq C_r) \vee (j = k) \\ \bar{x}_k, & \text{jinak.} \end{cases} \quad (3.2)$$

Hodnota  $U \in [0, 1]$  označuje náhodně vygenerované reálné číslo. Hodnota  $j$  je parametr, který zaručuje, že se potomek bude lišit alespoň jedním *allele* a nakonec hodnota  $k$  slouží pro iterování skrze jednotlivé hodnoty *allele* [15].

### Selekce

Selekce porovnává fitness nového cílového vektoru (mutovaného a kříženého) s původním. Do další generace je vybrán ten vektor, který má lepší hodnotu fitness:

$$\bar{x} \leftarrow \bar{x}', \text{ pokud } f(\bar{x}') \geq f(\bar{x}), \quad (3.3)$$

kde  $f$  představuje objektivní funkci pro maximalizační optimalizační problém [15].

Diferenční evoluce má spoustu variací, např. se liší způsoby jak lze vybírat rodiče. V literatuře se používá zavedená notace DE/x/y/z, kde  $x$  reprezentuje výběr rodiče (např. „rand“ – náhodný rodič, „best“ – nejlepší rodič),  $y$  je počet diferenčních vektorů a  $z$  značí použité schéma křížení (např. „bin“ je uniformní křížení) [2].

Tabulka 3.2: Shrnutí diferenční evoluce. Převzato z [2].

Reprezentace	Vektor reálných čísel
Křížení	Uniformní křížení
Mutace	Diferenční mutace
Selekce rodičů	DE/rand/1/bin – náhodně, DE/best/1/bin – nejlepší, ...
Selekce následníků	Deterministické nahrazení (rodič vs. potomek)

### 3.3 Algoritmus umělých včelstev

Při studiu algoritmu umělých včelstev (ang. *Artificial Bee Colony* – zkratka ABC) jsem vycházel z [12, 13]. Tento algoritmus analogicky odpovídá chování včelího hnízda a tedy používá včelařskou terminologii. V tomto algoritmu se objevují tři druhy umělých včel, a sice: „zaměstnané“ (ang. *employed*), „pozorovací“ (ang. *onlooker*) a „průzkumné“ (ang. *scout*). Zaměstnaná včela navštívuje zdroje potravy, kdežto pozorující včela vyčkává za účelem výběru zdroje potravy. Průzkumné včely provádí náhodné hledání zdroje potravy v okolí. V tomto algoritmu spadá první polovina včel do kategorie zaměstnaných, kdežto druhá polovina do kategorie pozorujících a každému zdroji potravy připadá právě jedna zaměstnaná včela. Z toho plyne, že počet zdrojů potravy je přesně roven počtu zaměstnaných včel. Průzkumnými včelami se stávají zaměstnané včely, jejichž zdroj potravy je vyčerpán zaměstnanými a pozorujícími včelami. Proces optimalizace je tedy následovný [12]:

1. Náhodná inicializace zdrojů potravy.
2. Umístění zaměstnaných včel na zdroje potravy v okolí.
3. Umístění pozorujících včel na zdroje potravy v okolí.
4. Vyslání průzkumných včel za účelem nalezení nových zdrojů potravy.
5. Pokud není splněna ukončovací podmínka, vrácení na bod 2.

Každá iterace algoritmu sestává ze třech fází: vyslání zaměstnaných včel na zdroje potravy v okolí a určení množství nektaru na těchto zdrojích, výběr zdrojů potravy pozorujícími včelami po určení množství nektaru jednotlivých zdrojů a určení průzkumných včel a jejich vyslání na možné zdroje potravy. V tomto algoritmu je kandidátní řešení problému reprezentováno jako zdroj potravy (potažmo zaměstnaná včela) a kvalitu řešení reprezentuje množství nektaru. To znamená, že čím více nektaru na zdroji potravy je, tím lepší toto řešení je. Zaměstnané a pozorující včely stochasticky modifikují zdroje potravy (řešení) a vyhodnocují množství nektaru (fitness). Nový zdroj potravy  $v_{ij}$  je ze starého zdroje  $x_{ij}$  a sousedního zdroje  $x_{kj}$  vytvořen následujícím způsobem:

$$v_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj}), \quad (3.4)$$

kde  $k \in 1, 2, \dots, BN$  a  $j \in 1, 2, \dots, D$  ( $BN$  je počet zdrojů potravy a  $D$  je dimenze řešeného problému) jsou náhodně vybrané indexy, kde však musí platit:  $i \neq k$ . Číslo  $\phi_{ij} \in [-1, 1]$  je také generováno náhodně a slouží k určení dalšího zdroje potravy v okolí, což lze při analogii na reálnou včelu chápat jako vizuální porovnávání zdrojů potravy [12, 13].

Každý zdroj potravy si udržuje hodnotu počtu pokusů. Pokud se množství nektaru v tomto zdroji zvýší (větší fitness hodnota), tento čítač se vynuluje. Pokud je tomu ovšem naopak a množství nektaru je nižší, tento čítač se zvýší o jedna. Pokud čítač dosáhne předem stanovené hodnoty (tzn. zdroj potravy je vyčerpán), stává se z něj průzkumná včela (tzn. zdroj je znovu náhodně inicializován a je určena jeho hodnota fitness) [12].

Tabulka 3.3: Shrnutí algoritmu umělých včelstev [12].

Reprezentace	Vektor reálných čísel
Křížení	—
Mutace	Modifikace zdrojů potravy včelami dle rovnice 3.4
Selekce rodičů	Náhodně vybrané zdroje potravy s omezujícími podmínkami
Selekce následníků	Náhodná inicializace zdroje potravy po vyčerpání nektaru

### 3.4 Optimalizace rojem částic

Při studiu optimalizace rojem částic (ang. *Particle Swarm Optimization* – zkratka PSO) jsem vycházel z [4, 2]. Tento algoritmus je inspirovaný sociálním chováním ptačího hejna, zatímco jméno a použitá terminologie odpovídá spíše fyzickým částicím. Podobně jako u diferenční evoluce, odlišujícím faktorem je odklonění od použití klasických operátorů evolučních algoritmů a pořadí jedinců v populaci nezávisí na jejich hodnotě fitness. Optimalizace rojem částic nepoužívá operátor křížení a mutace je definována jako sčítání vektorů. Celkový přehled lze nalézt v tabulce 3.4. Každé možné řešení (tzn. každý jedinec populace)  $\bar{x} \in \mathbb{R}^n$  s sebou nese také vektor „rychlostí“ (ang. *velocities*)  $\bar{v} \in \mathbb{R}^n$ . To znamená, že každé řešení je dvojice  $\langle \bar{x}, \bar{v} \rangle$ . Vektor rychlostí je později použit pro nalezení nového řešení, tzn. vektoru  $\bar{x}$ . Hlavní myšlenka algoritmu je tedy v tom, že nová dvojice  $\langle \bar{x}', \bar{v}' \rangle$  je vytvořena z aktuální dvojice  $\langle \bar{x}, \bar{v} \rangle$ , a to tak, že první se vypočte nový vektor rychlostí, a poté je přičten k řešení [2]:

$$\bar{x}' = \bar{x} + \bar{v}'. \quad (3.5)$$

Každé z řešení je tedy reprezentováno jako bod v prostoru s jistou pozicí a rychlostí, kde rychlost je použita pro výpočet následné pozice. Výpočet rychlosti samotné je v podstatě vážený součet třech komponentů, a to aktuální rychlosti a dvou vektorových rozdílů:

$$\bar{v}_i = w\bar{v}_i + \phi_1 U_1(\bar{b}_i - \bar{x}_i) + \phi_2 U_2(\bar{c} - \bar{x}_i), \quad (3.6)$$

kde váhy  $w$ ,  $\phi_1$  a  $\phi_2$  jsou označovány jako (zleva): *inertia*, faktor učení osobní změny a faktor učení sociální změny. Hodnoty  $U_1$  a  $U_2$  jsou náhodně vygenerované z uniformního rozložení v intervalu  $[0, 1]$ . Index  $i$  značí pozici v populaci/roji. Dále hodnota  $\bar{b}$  označuje dosavadní nejlepší výsledek jedince populace, zatímco  $\bar{c}$  označuje dosavadní nejlepší výsledek celé populace, kde první z hodnot je třeba uchovávat v paměti pro každého jedince

zatímco druhou pouze pro celou populaci. Tento mechanismus tedy vyžaduje aby jedinci byli jednoznačně identifikovatelní a měli tedy identitu, aby bylo možné upravovat jejich vlastní paměť (nejlepší dosavadní výsledek). Každého jedince populace lze tedy považovat jako trojici  $(\bar{x}, \bar{v}, \bar{b})$ , značící řešení (pozici), aktuální rychlost a dosavadní nejlepší výsledek jedince. Postup výpočtu je tedy následující [4, 2]:

1. Náhodná inicializace velikosti roje, pozic částic a rychlosti částic.
2. Výpočet nové rychlosti částice za pomoci rovnice (3.6).
3. Výpočet nové pozice částice za pomoci rovnice (3.5).
4. Aktualizace osobních a globálního nejlepšího výsledku.
5. Pokud není splněna ukončovací podmínka, vrácení na bod 2.

Tabulka 3.4: Shrnutí optimalizace rojem částic. Převzato z [2].

Reprezentace	Vektor reálných čísel
Křížení	—
Mutace	Přičtení vektoru rychlostí
Selekce rodičů	Deterministická (rodič vytvoří jednoho následníka pomocí mutace)
Selekce následníků	Generační (následník nahrazuje rodiče)

### 3.5 $MA\mathcal{X}$ – $MIN$ mravenčí systém

Při studiu  $MA\mathcal{X}$ – $MIN$  mravenčího systému (ang. *MA $\mathcal{X}$ –MIN Ant System* – zkratka *MMAS*) jsem vycházel z prací [6, 20]. Tento algoritmus je, jak název napovídá, inspirovaný chováním mravenčích kolonií. Hlavní princip spočívá v nepřímé komunikaci jedinců populace reprezentujících řešení problému, zvaných mravenci. Komunikace je zajištěna analogií chemické substance zvané feromon, která je periodicky upravována jednotlivými mravenci, reprezentující kvalitu řešení. Algoritmus *MMAS* je odvozen od původního algoritmu mravenčího systému (ang. *Ant System*), jehož cílem je dosažení lepší výkonnosti zejména při hledání řešení komplexnějších úloh.

Jedná se o variantu mravenčího systému, která na začátku výpočtu všechny hrany inicializuje hodnotou maximální hodnotou feromonu, tedy  $\tau_{max}$ , což umožňuje lepší prozkoumání řešení na začátku algoritmu. Během aktualizace jsou upraveny pouze hrany, které využilo nejlepší řešení a to buď nejlepší řešení v rámci celého výpočtu nebo pouze v rámci aktuální iterace. Hodnota feromonu každé hrany  $\tau_{ij}$  při aktualizaci feromonů je spočtena pomocí rovnice:

$$\tau_{ij}(t+1) = \rho\tau_{ij}(t) + \Delta\tau_{ij}^{best}, \quad (3.7)$$

kde  $\Delta\tau_{ij}^{best} = 1/f(s^{best})$  a  $f(s^{best})$  značí cenu nejlepšího (globálního či iteračního) řešení. Množství feromonu je omezeno následovně:  $\tau_{min} \leq \tau_{ij} \leq \tau_{max}$ . Toto omezení je zavedeno kvůli relativně vysoké pravděpodobnosti brzké stagnace vyhledávání způsobené značným důrazem na prohledávání založeném zejména na nejlepším řešení, tzn. aby rozdíly mezi



množstvím feromonů jednotlivých hran nebyly příliš obrovské. Dále tento algoritmus často obsahuje opakovanou inicializaci feromonů všech hran hodnotou  $\tau_{max}$  ve chvíli, kdy je detekována stagnace vyhledávání. Tento krok do procesu vyhledávání zavede dostatečné množství diverzity, aby se předcházelo stagnaci vyhledávání.

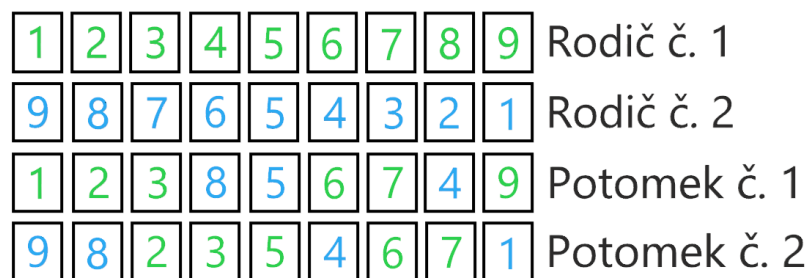
Pokud je pro aktualizaci feromonů použito vždy globální nejlepší řešení, hledací proces se může velmi rychle zaseknout v lokálním minimu a limitovat prohledávání lepších řešení, což může vést ke špatným výsledkům. Riziko uváznutí v lokálním minimu je sníženo, pokud se pro aktualizaci používá nejlepší řešení v rámci iterace, protože to se může značně lišit každou iteraci a tedy lze dosáhnout diverzity a lepšího prohledávání prostoru. Tyto dva přístupy je ovšem možné také kombinovat.

## 3.6 Redefinice jednotlivých metod pro diskrétní prostor

Téměř všechny evoluční algoritmy popsané v této kapitole pracují implicitně ve spojitém prostoru – tzn. počítají s reálnými čísly. Některé problematiky, jako je například problém obchodního cestujícího nebo problém distribuce a alokace produktů do lokací ve skladu pracují v diskrétním prostoru a tedy počítají s celými čísly. Aby bylo možné evoluční algoritmy na tyto problémy aplikovat, je potřeba provést redefinici algoritmu a s tím související redefinici jeho operátorů (mutace, křížení, ...). V této kapitole bude krátce popsán problém obchodního cestujícího a redefinice jednotlivých metod (vyjma mravenčího algoritmu), aby byly schopny řešit tento problém v diskrétním prostoru. V následujících sekcích je typicky popsáno jen nutné minimum pro fungování v diskrétním prostředí a k řešení problému obchodního cestujícího – velká část z algoritmů zůstává stejná jako u spojitých verzí algoritmů popsaných dříve v kapitole a tedy zde už není popisována. Důvod, proč je zde tato podkapitola je, že problém obchodního cestujícího je v literatuře velmi často řešeným problémem a má mnoho společného s problémem řešeným v rámci této práce – problematikou distribuce a alokace produktů do lokací ve skladu, která je v této práci řešena právě pomocí těchto redefinic evolučních algoritmů.

### 3.6.1 Problém obchodního cestujícího

Tento problém by se dal popsat následovně: pokud chce obchodní cestující navštívit každé z  $m$  měst právě jednou (kde délka cesty z města  $i$  do města  $j$  je označována jako  $c_{ij}$ ) a následně se vrátit do počátečního města, jaká je nejkratší taková cesta? Problém obchodního cestujícího spadá do třídy NP-úplných problémů. To znamená, že pokud existuje efektivní řešení (takové, že řeší daný problém v polynomiálním čase) problému obchodního cestujícího, pak existuje efektivní řešení pro každý problém z třídy NP-úplných problémů. Pro formulaci tohoto problému se používá graf  $G = (V, E)$ , kde  $V \in 1, 2, \dots, N$  jsou uzly reprezentující města a  $E$  hrany reprezentující cesty mezi nimi. Každá hrana má cenu, která reprezentuje vzdálenost mezi těmito městy, tzn. čím větší cena hrany, tím delší je cesta mezi těmito městy. Pokud se obchodní cestující může dostat z každého města do kteréhokoli jiného města, pak je graf úplný. Okružní výlet se skládá z jisté podmnožiny uzlů a hran, zvaný obecně jako cesta či v teorii grafů jako Hamiltonova kružnice. Výsledkem algoritmů řešící tento problém je tedy sekvence délky počtu měst s nejnižší cenou (nejkratší cestou), kde se žádné z měst neopakuje a každé sousední město musí být dosažitelné skrze jednu hranu [10, 13].



Obrázek 3.3: Graficky znázorněný příklad uspořádaného křížení dvou rodičů [7].

### 3.6.2 Redefinice genetických algoritmů pro diskrétní prostor

Při studiu jsem vycházel z [19, 7]. Řešení problému obchodního cestujícího (individuum z populace nebo také chromozom) lze chápat jako cestu složenou z pole genů, kde každý gen reprezentuje jedno město. Město lze reprezentovat například pomocí celočíselného identifikátoru města nebo jeho souřadnic ve formě  $(x, y)$ . Populace, selekce a fitness fungují na stejném principu jako ve spojitém prostoru, není je tedy potřeba předefinovávat. Je však nutné redefinovat operátory křížení a mutace tak, aby neprodukovaly v řešení duplicitní hodnoty.

**Ordered crossover** (česky uspořádané křížení) je křížení, které neporušuje omezení problému obchodního cestujícího specifikované v části 3.6.1. To znamená že po zkřížení dvou rodičů, potomek bude stále validním řešením a tedy bude obsahovat všechna města a žádné z nich se nebude opakovat. Princip je takový, že pro vytvoření potomka se vybere jedna či více částí cesty z prvního z rodičů. Následně jsou doplněny chybějící města z druhého rodiče a to v takovém pořadí, v jakém se nachází v druhém rodiči. Druhý potomek je vytvořen úplně stejným principem, pouze se role rodičů obrací. Příklad aplikace tohoto operátoru je graficky znázorněn na obrázku 3.3.

**Ordered mutation** (česky uspořádaná mutace) je mutace, která obdobně jako uspořádané křížení neporušuje žádné z omezení problému obchodního cestujícího. Funguje na jednoduchém principu přemístění prvku z pozice  $A$  na pozici  $B$ , kde platí, že  $A \neq B$ . Na tuto mutaci existují různé variance, jako například prohození prvků na indexech  $A$  a  $B$ , či inverze části cesty.

### 3.6.3 Redefinice diferenční evoluce pro diskrétní prostor

V této sekci je popsána tzv. SBDE – *step-based differential evolution* (česky volně přeloženo jako „diferenční evoluce založená na krocích“). V této verzi jsou všechny potřebné aritmetické operátory redefinovány následovně [15]:

**Element** Element  $(x, y)$  reprezentuje hranu  $(x, y)$  mezi městy  $x$  a  $y$ .

**Jedinec** Kandidátní řešení, skládá se z elementů a představuje Hamiltonskou kružnici.

**Operátor násobení**  $rand \in [0, 1] \times \text{element} = rand \in [0, 1] \times (x, y)$  udává, že element  $(x, y)$  bude pro konstrukci nového kandidátního řešení (individua) použit s pravděpodob-

ností *rand*. Při řešení problému obchodního cestujícího výraz  $0.3 \times (1, 2)$  říká, že hrana  $(1, 2)$  bude při konstrukci nové cesty navštívena s pravděpodobností 0.3.

**Operátor odečítání** Pokud  $x_1$  a  $x_2$  jsou jedinci populace, potom výraz  $x_1 - x_2 = e \mid e \in x_1 \wedge e \notin x_2$ , tzn. výsledkem jsou všechny hrany, které jsou v  $x_1$  a zároveň nejsou v  $x_2$ .

**Operátor sčítání** Pokud se element nachází v obou jedincích, element je použit ve výsledku, a je použita vyšší z pravděpodobností daných elementů se vyskytovat v kandidátním řešení. Pokud se prvek nachází pouze v jednom z dvou řešení, je použit ve výsledku a to se stejnou pravděpodobností.

**Výpočet** Redefinice aritmetických operátorů pro diskrétní prostor s jistým významem pro řešení problému obchodního cestujícího umožňuje použití binomického křížení, které se vyskytuje v klasické diferenční evoluci. Operátor mutace je třeba trochu upravit, a to následovně:

$$\bar{v} = \omega \times x_a + r_{rand} \times (x_b - x_c), \quad (3.8)$$

kde  $\omega$  a  $r_{rand}$  jsou pravděpodobnosti a jsou generovány náhodně pro každý element. V každé iteraci je pro každého jedince náhodně vygenerováno  $\alpha \in [0, 1]$ . Každý element v jedinci, který nemá pravděpodobnost nižší než je hodnota  $\alpha$ , je umístěn do „sady zbytků“. Z této sady zbytků se poté berou elementy pro konstrukci kandidátního řešení, Hamiltonské kružnice. V případě, že kandidátní řešení ještě není úplné a sada zbytků je již prázdná, je provedeno heuristické vyhledání k doplnění kandidátního řešení (např. hledání nejbližšího souseda). Zbytek je již stejný jako v klasické verzi diferenční evoluce.

### 3.6.4 Redefinice algoritmu umělých včelstev pro diskrétní prostor

V této sekci je popsáno řešení problému obchodního cestujícího pomocí algoritmu umělých včelstev založeného na *swap sequence* (česky volně přeloženo jako „sekvence výměny“) a *swap operators* (česky operátory výměny). Ty jsou definovány následovně [13]:

**Operátor výměny** Pokud  $X$  je možné řešení problému, tedy sekvence měst, kterými musí obchodní cestující projít – Hamiltonská kružnice  $X = (x_1, x_2, \dots, x_n, x_1)$  s množinou měst  $V = 1, 2, \dots, N$ , kde  $x_i \in V$  a  $x_i \neq x_j, \forall i \neq j$ , pak je operátor výměny  $SO(i, j)$  definován jako výměna měst/uzlů  $x_i$  a  $x_j$  v možném řešení problému  $X$ . Symbol  $\diamond$  značí binární operaci výměny a jeho výsledkem obecně je  $X' = X \diamond SO(i, j)$  – nové možné řešení problému. Příklad: Necht  $X = (x_1, x_2, x_3, x_4, x_5) = (1, 2, 3, 4, 5)$  je možné řešení problému obchodního cestujícího a  $SO(1, 4)$  operátor výměny. Poté  $X' = X \diamond SO(1, 4) = (1, 2, 3, 4, 5) \diamond SO(1, 4) = (4, 2, 3, 1, 5)$ , a tedy města na indexech 1 a 4 jsou prohozena a vzniká nové možné řešení problému  $X'$ .

**Sekvence výměny** Sekvence dvou a více operátorů výměny se nazývá sekvence výměny. Taková sekvence se značí jako  $SS = (SO_1, SO_2, \dots, SO_n)$ , kde  $SO_{1,2,\dots,n}$  jsou operátory výměny. Na pořadí jednotlivých operátorů v rámci sekvence záleží a jsou v tomto pořadí aplikovány na možné řešení následovně:

$$X' = X \diamond SS = X \diamond (SO_1, SO_2, \dots, SO_n) = (\dots((X \diamond SO_1) \diamond SO_2) \dots \diamond SO_n).$$

Takovéto sekvence výměny lze poté spojovat pomocí operátoru  $\oplus$  následovně:

$$\begin{aligned} SS' &= SS_a \oplus SS_b = (SO_{a1}, SO_{a2}, \dots, SO_{an}) \oplus (SO_{b1}, SO_{b2}, \dots, SO_{bn}) \\ &= (SO_{a1}, SO_{a2}, \dots, SO_{an}, SO_{b1}, SO_{b2}, \dots, SO_{bn}) \end{aligned}$$

Pokud jsou daná možná řešení  $X_1$  a  $X_2$  a je potřeba najít sekvenci výměny takovou, kterou lze aplikovat na  $X_2$  a získat tím  $X_1$ , tedy rozdíl těchto řešení z pohledu operátorů výměny, lze toho docílit pomocí operátoru  $\ominus$ .

**Výpočet** Na začátku jsou náhodně inicializovány všechny zdroje potravy, přiřazeny zaměstnaným včelám a vypočtena fitness stejně, jako ve spojitě verzi algoritmu. V první fázi zvané *employed*, každá zaměstnaná včela se pokouší zlepšit své řešení náhodným výběrem a aplikací jedné z následujících rovnic:

$$Y_i = X_j \diamond (r \odot (X_i \ominus X_k)), \quad (3.9)$$

$$Y_i = X_i \diamond (r \odot (X_j \ominus X_k)), \quad (3.10)$$

$$Y_i = X_{best} \diamond (r \odot (X_i \ominus X_k)), \quad (3.11)$$

$$Y_i = X_i \diamond (r \odot (X_i \ominus X_{best})), \quad (3.12)$$

$$Y_i = X_{best} \diamond (r \odot (X_{best} \ominus X_k)), \quad (3.13)$$

$$Y_i = X_i \diamond (r \odot (X_{best} \ominus X_{worst})), \quad (3.14)$$

$$Y_i = X_i \diamond (r_1 \odot (X_{best} \ominus X_k) \oplus r_2 \odot (X_k \ominus X_i)), \quad (3.15)$$

$$Y_i = X_j \diamond (r \odot (X_{best} \ominus X_i)), \quad (3.16)$$

kde  $Y_i$  je nové řešení (zdroj potravy) získané z  $X_i$  zaměstnanou včelou.  $X_j$  a  $X_k$  jsou náhodně vybrané zdroje potravy, pro které platí  $j \neq k$ . Doposud nejlepší řešení je reprezentováno jako  $X_{best}$  a podobně, nejhorší řešení jako  $X_{worst}$ . Výraz  $r \odot SS$  udává, že jednotlivé operátory výměny  $SO \in SS$  v sekvenci zůstanou s pravděpodobností  $r \in [0, 1]$ . To, která z rovnic 3.9 – 3.16 bude včelou použita, je definováno následovně: každé z rovnic je přiřazen čítač a inicializován na 1. Pokud je rovnice vybrána, je její čítač zvýšen o 1. Pravděpodobnost výběru  $i$ -té rovnice je dána touto rovnicí:

$$\rho_i = \frac{v_i}{\sum_{j=1}^{N_{eq}} v_j}, \quad i = 1, 2, \dots, N_{eq}, \quad (3.17)$$

kde  $N_{eq}$  značí počet rovnic, a  $v_i$  značí rovnici na indexu  $i$ . Pro každou z rovnic je tedy náhodně vygenerováno  $r \in [0, 1]$  a spočteno  $\rho_i$ , pokud platí, že  $r < \rho_i$ , je vybrána rovnice  $v_i$  a provedena její aplikace. Poté se znovu vyhodnotí fitness – tj. množství nektaru a algoritmus se opakuje. Zbytek je již stejný jako v klasické verzi algoritmu.

### 3.6.5 Redefinice optimalizace rojem částic pro diskrétní prostor

V této sekci je popsán algoritmus pro řešení problému obchodního cestujícího pomocí optimalizace rojem částic (v kombinaci s genetickými algoritmy) využívající heuristické křížení jedinců, jehož algoritmus lze vidět v pseudokódu 1. Postup výpočtu redefinovaného algoritmu je následující: Na začátku je inicializován roj o velikosti  $s$  náhodně. To znamená, že každá částice má náhodnou pozici a pro každou částici se nastaví osobní nejlepší výsledek na aktuální náhodně inicializovanou hodnotu. Následně je pro každou částici vypočtena hodnota fitness pomocí objektivní funkce a je vybráno globální maximum. Pro každou částici je vypočtena nová pozice takto:  $x = b_p^x \otimes b_g$ , kde  $b_p^x$  značí dosavadní osobní maximum prvku  $x$ ,  $b_g$  značí dosavadní globální maximum všech částic a operátor  $\otimes$  značí heuristické křížení popsané v pseudokódu 1. Poté je pro každou nově spočtenou částici znovu vyhodnocena objektivní funkce, a pokud je nová hodnota lepší než hodnota osobního nejlepšího výsledku, je osobní nejlepší výsledek přepsán touto novou hodnotou. Po takovémto přepočítání všech částic je znovu vyhodnoceno globální maximum. Takové přepočítávání pozic a znovu vyhodnocování objektivní funkce se opakuje až do doby, kdy je splněna některá z ukončovacích podmínek [4].

---

**Algoritmus 1** Heuristické křížení jedinců [4].

---

**Vstup:** Dva jedinci  $x_1$  a  $x_2$

**Výstup:** Jedinec  $x$

**Kroky:**

```
1: Výběr náhodného města  $v$ 
2: Přesun města  $v$  na začátek  $x_1$  a  $x_2$ 
3: Inicializace  $x$  pomocí  $v$ 
4: for  $i, j = 2, \dots, n$  do
5:   if  $x_1[i] \in x$  and  $x_2[j] \in x$  then
6:      $i = i + 1$ 
7:      $j = j + 1$ 
8:   else if  $x_1[i] \in x$  then
9:     Konkatenace  $x_2[j]$  k  $x$ 
10:     $j = j + 1$ 
11:  else if  $x_2[j] \in x$  then
12:    Konkatenace  $x_1[i]$  k  $x$ 
13:     $i = i + 1$ 
14:  else
15:    Necht  $u$  je poslední město v  $x$ 
16:    if vzdálenost( $u, x_1[i]$ ) < vzdálenost( $u, x_2[j]$ ) then
17:      Konkatenace  $x_1[i]$  k  $x$ 
18:       $i = i + 1$ 
19:    else
20:      Konkatenace  $x_2[j]$  k  $x$ 
21:       $j = j + 1$ 
22:    end if
23:  end if
24: end for
```

---

## Kapitola 4

# Návrh nástrojů pro simulaci a optimalizaci skladu

Tato kapitola se zabývá návrhem nástrojů potřebných pro optimalizaci skladových operací. Výsledné řešení se bude skládat z pěti „kooperujících“ nástrojů. Veškeré nástroje budou implementovány v C++ a pro grafickou nastavbu bude použit framework Qt verze 5.

- **Generátor produkčních dat** – Vzhledem k nemožnosti použití zákaznických objednávek za účelem udržení obchodního tajemství je potřeba vytvořit generátor dat založený na jistém matematickém modelu, aby data nebyla čistě náhodná, nýbrž spolu korelovala.
- **Simulátor skladu** – Nástroj, který na základě vygenerovaných objednávek a vytvořeného layoutu skladu dokáže odsimulovat pickování vygenerovaných objednávek a poskytnout statistiky, zejména pak dobu potřebnou k pickování a zatížení lokací a dopravníků.
- **Pathfinder** (česky hledač cest) – Nástroj, který dokáže nalézt optimální cestu objednávky skrze sklad.
- **Optimalizátor rozložení produktů** – Nástroj, který dokáže optimalizovat alokaci jednotlivých produktů do lokací skladu za účelem snížení doby potřebné k napickování veškerých zákaznických objednávek.
- **Warehouse Manager** (česky skladový manažer) – Grafická aplikace, která uživateli umožní vytvořit model skladu. Dále bude také obsahovat všechny čtyři zmíněné nástroje v grafické podobě.

### 4.1 Generátor produkčních dat

Generátor bude mít za úkol generovat dvě sady syntetických objednávek. Jedna sada bude určena pro trénování genetického algoritmu (reprezentující historické data společnosti) a druhá pro vyhodnocení natrénovaného modelu (reprezentující „budoucí“ data společnosti). Tyto dvě sady spolu musí korelovat, a proto bude vytvořen jeden matematický model (pravděpodobnosti jednotlivých produktů vypočteny na základě ADU – *Average daily units* a množství na základě ADQ – *Average daily quantity*). Pro generování ADU i ADQ pro jednotlivé produkty se využije Normální (Gaussovo) rozdělení.

## 4.2 Simulátor skladu

Bude implementován pomocí vhodné knihovny umožňující simulaci za pomoci diskretních událostí. Zejména bude sloužit pro vyhodnocení kvality jednotlivých řešení v optimalizačním nástroji, ale bude také možné jeho samostatné využití např. pro identifikaci úzkých míst či získání statistik zpracování objednávek ve skladu.

## 4.3 Pathfinder

Tento nástroj bude sloužit pro nalezení optimální cesty objednávky skladem, tak, aby urazila co nejmenší možnost vzdálenost. Toho bude dosaženo pomocí mravenčího algoritmu popsaném v 3.5.

## 4.4 Optimalizátor rozložení produktů

Řešený optimalizační problém (alokace produktů do slotů lokací) má velmi mnoho společného s problémem obchodního cestujícího. Stejně jako obchodní cestující musí projít všechna města, je zde potřeba alokovat všechny produkty do slotů. A stejně tak jako obchodní cestující nesmí navštívit žádné město vícekrát, ani žádný z produktů nelze alokovat do dvou či více slotů zároveň. Budou tedy implementovány čtyři evoluční algoritmy, jež budou implementovány podle redefinic jednotlivých algoritmů pro diskretní prostor a pro řešení problému obchodního cestujícího popsaných v 3.6.

## 4.5 Warehouse Manager

Aplikace bude založena na grafickém frameworku Qt a pro tvorbu modelu budou využity grafické prvky, které budou zasazovány do grafické scény. Dále bude grafické rozhraní doplněno o veškeré nástroje popsané výše, pro jednoduchost použití a pro konzistentnost.

## Kapitola 5

# Implementace nástrojů pro simulaci a optimalizaci skladu

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.

### 5.1 Generátor produkčních dat

Pro optimalizaci a vyhodnocení bylo nutno vytvořit generátor syntetických zákaznických objednávek. Není však možné, aby tyto generované objednávky byly čistě nahodilé, je třeba je generovat na základě nějakého matematického principu, a to z důvodu potřeby alespoň dvou sad objednávek, které budou různé ale budou spolu korelovat. První sada je použita pro optimalizaci skladu (dále nazývaná jako trénovací sada) a druhá pro vyhodnocení kvality optimalizace (dále nazývána jako testovací sada). Myšlenka je taková, že se vyhodnotí simulace skladu na testovací sadě objednávek a uloží se simulovaná doba. Poté se provede optimalizace na trénovací sadě objednávek a na tomto optimalizovaném modelu skladu se opět provede simulace na testovací sadě objednávek. Poté se porovná výsledná doba obou běhů simulace (před a po optimalizaci).

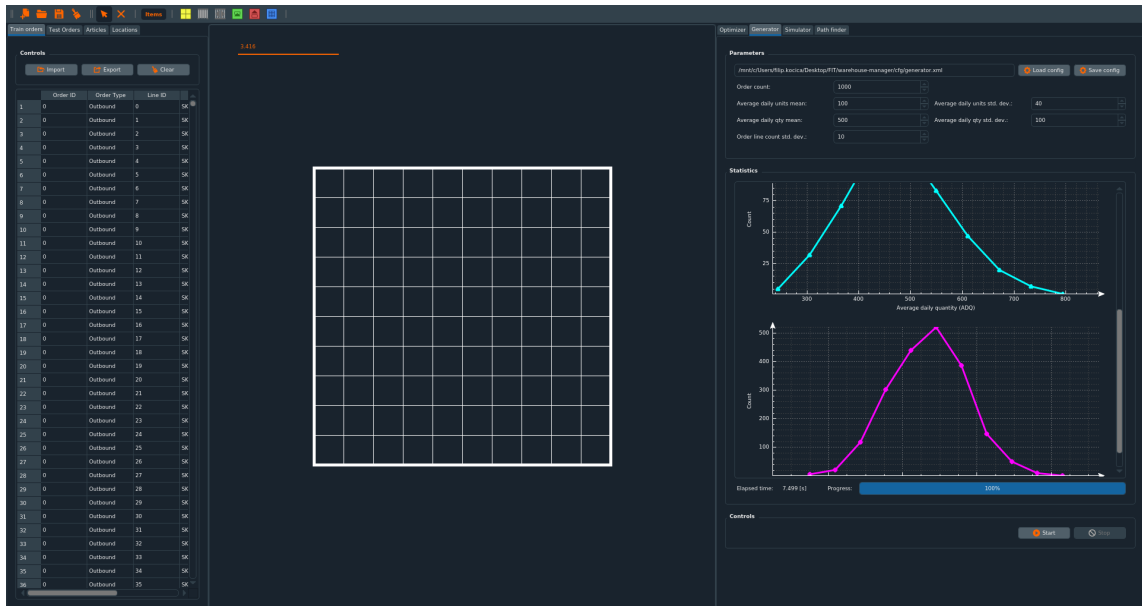
#### 5.1.1 Konfigurace

Generátor objednávek, stejně jako ostatní nástroje, je závislý na vstupu od uživatele, a proto bylo nutné implementovat způsob zadávání potřebných hodnot. K tomu byly využity konfigurační soubory ve formátu XML. Při použití grafického rozhraní je možné nastavit všechny parametry přímo tam, jak lze vidět na snímku 5.1.

Jednotlivé položky které je uživatel schopen nastavit a jejich významy jsou následující (podrobnější vysvětlení lze najít v sekci 5.1.2):

- `orderCount` – Počet generovaných objednávek.
- `miAdu` – Střední hodnota normálního rozložení pro generování ADU produktů.
- `sigmaAdu` – Rozptyl normálního rozložení pro generování ADU produktů.
- `miAdq` – Střední hodnota normálního rozložení pro generování ADQ produktů.





Obrázek 5.1: Grafická implementace generátoru objednávek zakomponovaná do aplikace Warehouse Manager. Uprostřed je plocha pro tvorbu modelu skladu. Vpravo je kontrolní panel, kde je možné načíst a nastavit veškeré parametry generování, sledovat tři grafy normálních rozložení (vytvořených z generovaných dat) a spustit či zastavit generování. Vygenerovaná data jsou uložena do panelu s daty vlevo, kde je lze použít pro optimalizátor (první záložka s trénovacími daty), pro simulátor a pathfinder (druhá záložka s testovacími daty), nebo je možné tyto objednávky serializované exportovat do souboru na disk a opětovně je poté načíst.

- $\sigma_{Adq}$  – Rozptyl normálního rozložení pro generování ADQ produktů.
- $\sigma_{Lines}$  – Rozptyl počtu položek objednávek.

### 5.1.2 Princip fungování a implementace

Parametry generovaných sad objednávek, což jsou jejich množství, obsah a velikost, jsou dány uživatelem definovanými pravděpodobnostními rozděleními. Pravděpodobnostní modely, na základě kterých se generování provádí, jsou Gaussova rozložení – parametry (střední hodnotu a rozptyl) definuje uživatel. Generátor je založen na hodnotách  $ADU^1$  a  $ADQ^2$ .

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

$$p_i = \frac{ADU_i}{\sum_{n=1}^N ADU_n}, \quad (5.1)$$

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

<sup>1</sup> *Average daily units* – průměrný denní počet zakoupení.

<sup>2</sup> *Average daily quantity* – průměrná denní zakoupená kvantita.

rozložení z rovnice 5.1, 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 pravděpodobnostních rozložení, vzniklé sady jsou různé, avšak spolu korelují.

## 5.2 Simulátor skladu

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. Grafickou reprezentaci simulátoru lze vidět na snímku 5.2.

Autoři práce [9] zmiňují, že ze všech možných druhů je nejvhodnější a nejpřirozenější simulace skladu pomocí diskretních událostí, protože sklad je v podstatě kolekce entit, které reagují na fixní diskretní události. Simulátor je tedy založen na principu diskretní simulace a při implementaci byla využita knihovna SIMLIB/C++<sup>3</sup>. Simulátor poskytuje poměrně komplexní konfiguraci, což umožňuje rozsáhlé možnosti experimentování (od nastavení rychlostí pracovníků a dopravníků až po doplňování produktů, viz 5.2.5).

### 5.2.1 Konfigurace

Stejně jako předchozí nástroj, je i simulátor konfigurovatelný, a to buď přímo v grafickém rozhraní, nebo XML souborem při použití textového rozhraní. Jednotlivé položky, které je uživatel schopen nastavit a jejich významy jsou následující:

- `toteSpeed` – Tato hodnota udává, jak rychle jezdí kartony po dopravnících a udává se v metrech za sekundu.
- `workerSpeed` – Udává rychlost pracovníka, to znamená jak rychle se dokáže pohybovat a vytahovat produkty ze slotů lokací. Opět v metrech za sekundu.
- `totesPerMin` – Udává, kolik kartonů/objednávek dokáže sklad odeslat za jednu minutu.
- `simSpeedup` – Umožňuje zrychlení či zpomalení simulace.
- `locationCapacity` – Udává, kolik objednávek je možné zároveň zpracovávat na jedné lokaci (lze chápat jako počet pracovníků na lokaci).
- `conveyorCapacity` – Udává, kolik kartonů je možné zároveň převážet pomocí jednoho dopravníku.

---

<sup>3</sup>*Simulation Library for C++* – <http://www.fit.vutbr.cz/~peringer/SIMLIB>

- `orderRequestInterval` – Udává interval, po jehož uplynutí přichází do systému nová objednávka.
- `replenishment` – Udává, zda má systém počítat s kvantitami produktů a tedy s jejich doplňováním v případě nutnosti.
- `initialSlotQty` – Tato hodnota říká, jaká je počáteční kvantita produktů v jednotlivých slotech.
- `replenishmentQuantity` – Udává, kolik produktů se bude doplňovat.
- `replenishmentThreshold` – Pokud počet produktů ve slotu klesne pod tuto hodnotu, je pro daný produkt spuštěno doplňování produktů.
- `preprocessing` – Před-zpracování objednávek aby se zrychlila/zjednodušila simulace (tzn. seřazení položek objednávky, aby cesta skladem byla kratší):
  - `None` – Žádné předzpracování.
  - `Normal` – Předzpracování založené na jednoduchém pravidle porovnávající Manhattanové vzdálenosti jednotlivých lokací.
  - `Optimized` – Pro každou objednávku je nalezena optimální cesta skladem pomocí nástroje pathfinder, popsaného v sekci 5.3.

## 5.2.2 Princip fungování a implementace

Na začátku se provede načtení objednávek, modelu skladu a alokace produktů do slotů lokací. Poté se provede před-počítání všech možných cest ve skladu, jejich ohodnocení (délka v metrech) a seřazení od nejvýhodnější po nejméně výhodnou. Tímto způsobem lze získat nejvýhodnější cesty mezi všemi prvky/zařízeními ve skladu. Způsob zpracování je popsán pomocí PT sítě na snímku 5.3.

## 5.2.3 Paralelní simulace

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 znázorněna na snímku 5.4.

- Od rodiče k potomkům se posílá zakódovaná alokace produktů do jednotlivých slotů (celočíslné pole).



Obrázek 5.2: Grafická implementace simulátoru skladu zakomponovaná do aplikace Warehouse Manager. Uprostřed je stále plocha pro tvorbu modelu skladu. Vpravo je kontrolní panel, kde je možné nastavit parametry simulace, sledovat grafy simulace (časy dokončení jednotlivých zákaznických objednávek, nárůst ujeté vzdálenosti na dopravnících a ušlé vzdálenosti pracovníků), vidět počet dokončených doplňovacích objednávek a kontrolovat simulaci (spustit, zastavit). Vlevo jsou data, které simulátor používá, jako je testovací sada objednávek a alokace produktů do slotů lokací. V modelu skladu lze vidět, které sloty lokací jsou zaplněny a je zobrazena také teplotní mapa produktů v lokacích i jednotlivých grafických prvků, kterou lze vidět na snímku 5.8 a která reprezentuje zatížení prvků.

- Od potomků zpět k rodiči se posílá výsledek (doba) simulace, a sice hodnota *fitness* (číslo s plovoucí řádovou čárkou).

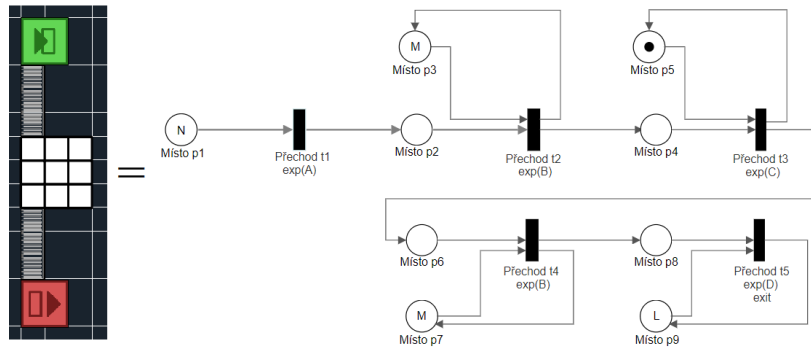
Jedinci (potažmo jejich simulace) jsou mezi procesy rozděleny zcela rovnoměrně a zrychlení optimalizace pomocí paralelizace simulací bylo velmi značné.

### 5.2.4 Řešení problému duplicitních jedinců

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.

### 5.2.5 Doplnování produktů

Doplnování 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)



Obrázek 5.3: Triviální sklad se vstupem, jednou lokací a výstupem propojených dopravníkem. Vpravo je odpovídající PT síť. Na začátku se nachází místo  $p_1$  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 poměr délky dopravníku a rychlosti 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 všechny hodnoty v celém procesu jsou konfigurovatelné.

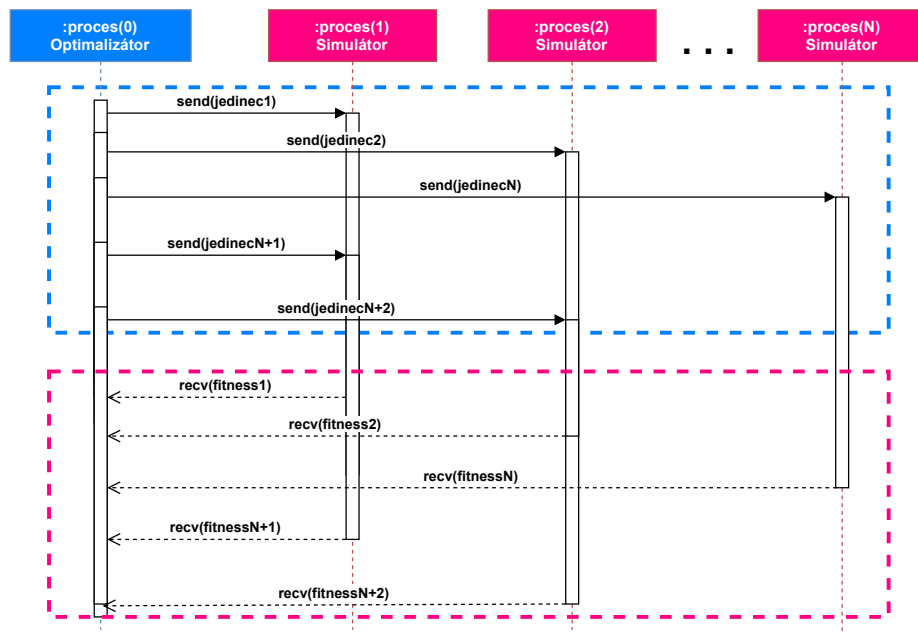
úroveň. Tento mechanismus byl implementován za účelem zvýšení realističnosti simulace skladu.

Objednávka pro doplňování je v podstatě úplně stejná jako zákaznická objednávka a je také reprezentována jako proces v systému, a také se tak chová. Poté, co se objednávka pro doplnění dostane do cílové lokace, jsou veškeré produkty z objednávky doplněny do příslušných slotů lokace a objednávka je ukončena.

Zákaznické objednávky, které nelze napickovat z důvodu nedostatku produktů ve slotu jsou deaktivovány a musí čekat na doplňovací objednávku, což lze v systému chápat jako penalizaci. Poté, co přijde objednávka pro doplnění produktů a je dokončena, jsou všechny takto deaktivované procesy reprezentující zákaznické objednávky (na dané lokaci) opět aktivovány a zkontrolují, zda už je lze napickovat nebo se musí znovu uspat a čekat na „svou“ doplňovací objednávku.

### 5.3 Pathfinder

Nástroj pro optimalizaci cesty byl implementován skrze evoluční algoritmus  $MAX-MIN$  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 vy-



Obrázek 5.4: 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.

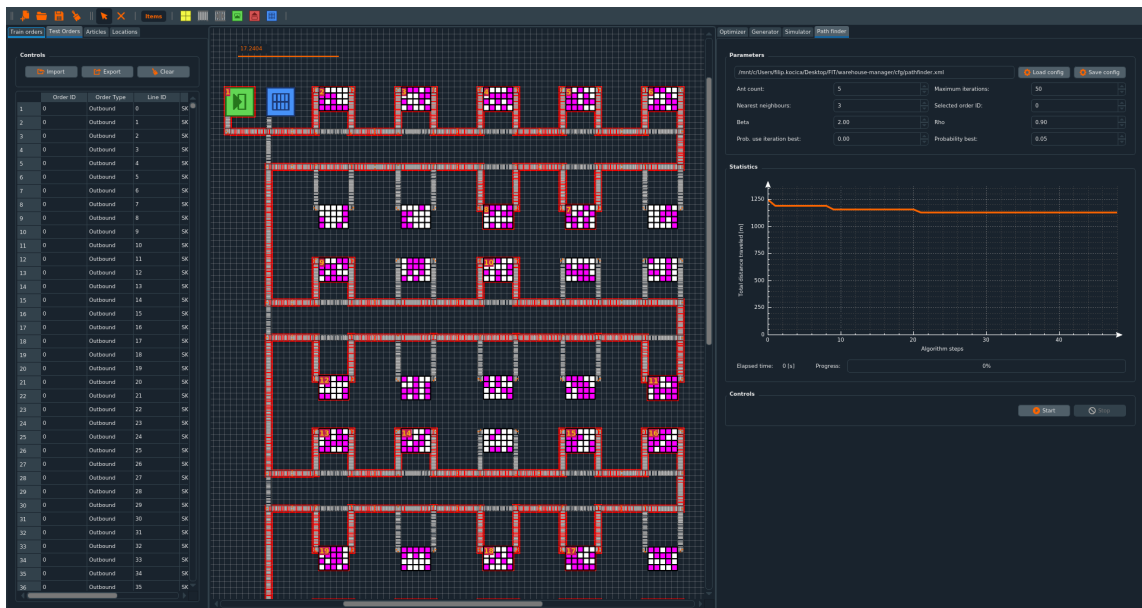
brané objednávky skrze sklad a očíslovat grafické prvky, aby bylo zřejmé, v jakém pořadí je objednávka navštíví. Implementace je inspirována **MAX-MIN-Ant-System**<sup>4</sup>.

### 5.3.1 Konfigurace

Hledač cest je konfigurovatelný buď pomocí konfiguračního souboru ve formátu XML, či v grafickém rozhraní a význam jednotlivých položek je následující:

- **antCount** – Počet umělých mravenců v kolonii.
- **rho** – Udává míru vypařování feromonů.
- **beta** – Hodnota použitá pro výpočet matice heuristik.
- **probBest** – Pravděpodobnost, že výsledně řešení bude obsahovat pouze nejlepší hrany.
- **nearestNeighbours** – Počet nejbližších sousedů uložených pro každý uzel.
- **probUseIterationBest** – Pravděpodobnost s jakou bude pro aktualizaci feromonů použit nejlepší dosažený výsledek v dané iteraci. S pravděpodobností rovnou doplňku této hodnoty do jedné bude pro aktualizaci použit dosavadní nejlepší výsledek.
- **maxIterations** – Maximální počet iterací mravenčího algoritmu.
- **selectedOrderID** – Jednoznačný identifikátor objednávky, pro kterou se má optimální cesta počítat.

<sup>4</sup><https://github.com/RSkinderowicz/MAX-MIN-Ant-System>



Obrázek 5.5: Grafická implementace nástroje pathfinder zakomponovaná do aplikace Warehouse Manager. Uprostřed je stále plocha pro tvorbu modelu skladu. Vpravo je kontrolní panel, kde je možné nastavit parametry nástroje, sledovat graf objektivní funkce a spustit či zastavit hledání optimální cesty. Vlevo jsou opět data používaná nástrojem (objednávky, produkty a sloty lokací). Nástroj je v grafickém rozhraní implementován v separátním vlákně, aby neblokoval hlavní okno. Po každém kroku optimalizačního algoritmu se však provádí zpětné volání (ang. *callback*) do hlavního vlákna pro přidání nové hodnoty do grafu objektivní funkce a překreslení aktuální nejkratší cesty. Cesta je červeně zvýrazněna a prvky skladu jsou doplněny o čísla aby bylo jasné pořadí, v jakém je bude objednávka navštěvovat.

### 5.3.2 Princip fungování a implementace

Na začátku výpočtu se nalezne vstup do skladu (který bude použit vždy jako počátek cesty mravence) a výstup ze skladu (který bude použit vždy jako poslední prvek cesty mravence). Následně se z položek vybrané objednávky zjistí, které lokace musí tato objednávka navštívit (ty budou součástí cesty mezi zmíněným počátkem a koncem). Toto vytvoří graf, který je potřeba projít.

Následně probíhá inicializace matice vzdáleností jednotlivých uzlů (prvků skladu), matice heuristik a nakonec nalezení  $N$  (konfigurovatelné) nejbližších sousedů pro každý uzel. Vzdálenost mezi jednotlivými uzly není Eulerova („vzdušnost čarou“), nýbrž Manhattanová, respektující délku pravoúhle propojených dopravníků propojující tyto dva uzly.

Pro výpočet počátečních limitů hodnot feromonů je nutné sestavit počáteční řešení problému tzv. „hltavým“ způsobem, tzn. první jsou do řešení přidáni všichni nejbližší sousedi daného uzlu, a poté zbytek všech uzlů. Poté lze inicializovat matici udávající množství feromonů na přechodech mezi jednotlivými uzly – kde jsou všechny přechody inicializovány na maximální hodnotu feromonu.

Poté je zahájen výpočet, kdy se v každé iteraci algoritmu sestaví  $N$  řešení (mravenců reprezentujících cestu, konfigurovatelné), kterým je následně spočtena délka cesty. Poté je nalezeno nejlepší řešení v dané iteraci a pokud je třeba, tak i aktualizováno dosavadní

nejlepší nalezené řešení. Poté je množství feromonu sníženo o míru vypařování a hranám, přes které prochází nejlepší řešení je doplněn feromon v závislosti na kvalitě daného řešení.

Po dosažení před-definovaného množství iterací se vypíše nejlepší nalezený výsledek. V případě, že je zapnuta i grafická nástavba, jsou průběžné výsledky pomocí zpětného volání posílány do grafického rozhraní (aby se pro každou iteraci mohlo vykreslit nejlepší nalezené řešení) a souběžně je tvořen graf udávající nejkratší délku cesty v každé iteraci, jak lze vidět na snímku 5.5. Tento nástroj je možné použít pro nalezení optimální cesty každé objednávky v simulátoru.

## 5.4 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. Hlavní myšlenka optimalizátoru v této práci je minimalizace doby nutné pro zpracování všech objednávek skrze vhodné rozložení produktů do jednotlivých slotů v lokacích skladu. Doba zpracování objednávek je aproximována simulátorem popsaném v předešlé kapitole. Doba zde představuje simulační čas, nikoli reálný. Grafickou implementaci lze vidět na snímku 5.6.

Pro nalezení optimální distribuce produktů do slotů lokací byl jako nejvhodnější přístup vybrán GA (genetický algoritmus), a to protože 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 [13, 15, 4, 7, 19] redefinovat pro diskrétní prostor. K tomu pomohla zejména velká podobnost problematiky SLAP a TSP (problému obchodního cestujícího), pro který byly tyto redefinice v odborných člancích popsány).

### 5.4.1 Kódování

Pro použití evolučních algoritmů bylo nutné navrhnout vhodný způsob kódování řešení. Každý ze čtyř použitých evolučních algoritmů (genetické algoritmy, diferenční evoluce, algoritmus umělých včelstev i optimalizace rojem částic) používají jiné názvosloví. Např. v genetických algoritmech se pro jedno zakódované řešení problému používá výraz jedinec či chromozom. V algoritmu umělých včelstev je to pak včela či zdroj potravy a v optimalizaci rojem částic je to částice či jedinec. Vzhledem k tomu, že pojem „jedinec“ se vyskytuje nejčastěji a nejlépe vystihuje podstatu, bude v tomto textu řešení problému označováno jako jedinec.

Výhodou je, že všechny čtyři algoritmy používají stejné kódování jedinců, proto mohla být vytvořena bázeová třída se všemi společnými operacemi (jako inicializace populace, provedení simulace všech jedinců, apod.). Všechny algoritmy byly implementovány ve vlastní třídě vždy odvozené z bázeové třídy, což mimo jiné umožnilo využití polymorfismu. Vzhledem k podstatě problému, který byl řešený v diskrétním prostoru, byl zakódovaný jedinec reprezentován jednoduše jako vektor celých čísel. Tento vektor má vždy délku stejnou jako





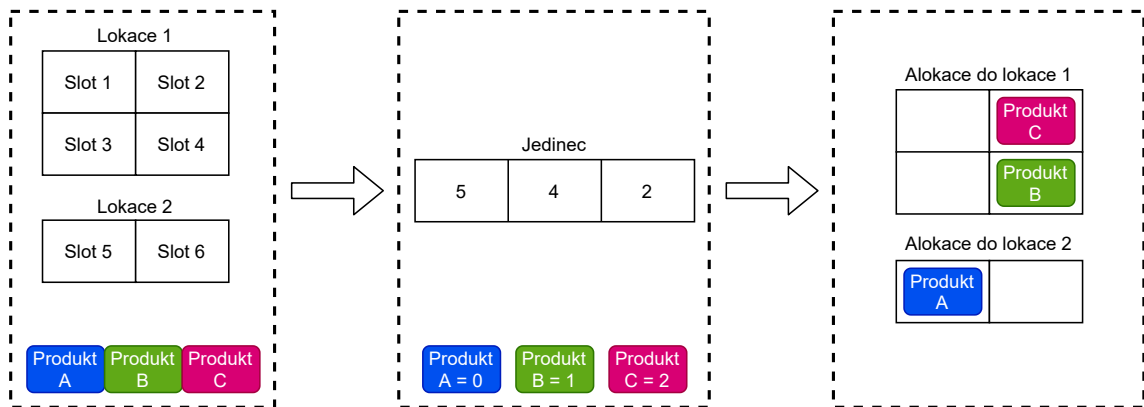
Obrázek 5.6: Grafická implementace optimalizátoru skladu zakomponovaná do aplikace Warehouse Manager. Uprostřed je stále plocha pro tvorbu modelu skladu. Vpravo je kontrolní panel, kde je možné nastavit parametry optimalizace (a vybrat si optimalizační algoritmus a do-nastavit jej v příslušné záložce), vidět graf objektivní funkce a spustit či zastavit optimalizaci. Vlevo jsou opět data používaná optimalizátorem (trénovací objednávky, produkty a sloty lokací). Optimalizátor je v grafickém rozhraní implementován v separátním vlákně, aby neblokoval hlavní okno. Po každém kroku optimalizačního algoritmu se provádí zpětné volání do hlavního vlákna pro přidání nové hodnoty do grafu a překreslení aktuální alokace produktů ve slotech a aktuálního zatížení prvků.

je počet produktů, které se algoritmus snaží optimálně roz distribuovat do slotů lokací. Jednotlivé položky (zvané jako geny, *allele* hodnoty, apod.) poté nabývají celých čísel reprezentující slot lokace, do kterého bude daný produkt uložen. Všem slotům, které se ve skladu nachází algoritmus přiřadí unikátní celočíselný identifikátor, který se poté používá jako hodnota ve vektoru každého jedince). Příklad znázorňující toto kódování lze najít na obrázku 5.7.

Tento přístup má však také svá omezení:

1. Počet produktů může být rozdílný od počtu slotů lokací, avšak počet slotů musí být vyšší než nebo stejný jako je počet produktů (aby byl každý produkt někam umístěn).
2. Pokud by bylo provedeno klasické křížení či mutace jedinců bez omezujících podmínek, s velkou pravděpodobností by v jedinci vznikly duplikáty celočíselných hodnot, což by značilo, že více produktů je umístěno do stejného slotu, což není možné.
3. Bod dva by také implikoval, že některé produkty by nebyly přiřazeny do žádného slotu, což by způsobilo, že objednávky s těmito produkty by nebylo možné dokončit.

Vyjma prvního bodu jsou tyto omezující podmínky v podstatě principiálně stejné, jako tomu je u problému obchodního cestujícího. V podkapitole 3.6 byla popsána teorie redefinice jednotlivých evolučních algoritmů pro diskrétní prostor a pro řešení problematiky obchodního cestujícího. Tato redefinice byla provedena pro čtyři zmíněné evoluční algoritmy.



Obrázek 5.7: Triviální příklad pro navržené kódování. Mějme dvě lokace, první se čtyřmi sloty, druhou se dvěma sloty. Do těchto lokací chceme alokovat tři produkty: A, B, C. Každému slotu je přiřazen unikátní celo-číselný identifikátor. Vezmeme v potaz jednoho jedince populace, který má délku stejnou jako je počet produktů, což je tři. První index reprezentuje první produkt, druhý index druhý produkt atd. Čísla na jednotlivých indexech reprezentují slot, do kterého je produkt alokován. Tzn. produkt A, který obsahuje hodnotu 5 bude umístěn do slotu 5, který se nachází v druhé lokaci.

#### 5.4.2 Grafická teplotní mapa

Jak lze vidět na snímku 5.6, každý ze zaplněných slotů produktem má jinou barvu. To značí, jak velké má produkt ADU – tzn. interpretace množství zakoupení produktu pomocí teplotní mapy (ang. *heatmap*). Vzhledem k tomu, že jednotlivé zařízení ve skladu (dopravníky, lokace, ...) mohou mít různé zatížení (t.j. doba, po kterou je zařízení v provozu z pohledu doby celé simulace), byl implementován mechanismus na zjištění zatížení jednotlivých prvků a jeho grafického znázornění. Grafická vizualizace využívá teplotní mapu na obrázku 5.8, a to tak, že kolem jednotlivých prvků vytvoří barevné ohraničení. A tedy nejvíce vytížené prvky budou mít ohraničení rudě červené, zatímco téměř nevyužité prvky tmavě modré. Tento mechanismus by měl uživatelům pomoci identifikovat „úzká místa“ ve skladu. Uživatelé pak mohou zkusit přijít na řešení pomocí distribuce zátěže mezi více zařízení a opětovné simulace, nebo nechat optimalizátor se s tímto problémem vypořádat za ně. Přidáním přídatných dopravníků se značně snížila zátěž a průchod skladem se zrychlil. Toto zatížení se v případě textového použití optimalizátoru či simulátoru vypisuje textově.

#### 5.4.3 Konfigurace

Nástroj lze opět konfigurovat pomocí XML souboru či v grafickém rozhraní a jednotlivé položky které je uživatel schopen nastavit a jejich významy jsou následující – první blok jsou obecné parametry použité ve více algoritmech, následuje blok s parametry pro genetické algoritmy a zbytek byl pro úsporu místem vynechán:

- `numberDimensions` – Rozměr problému, neboli velikost jedince.
- `problemMin` – Nejnižší hodnota, které může gen nabývat.
- `problemMax` – Nejvyšší hodnota, které může gen nabývat.
- `maxIterations` – Maximální počet cyklů optimalizačního algoritmu.



Obrázek 5.8: Grafická vizualizace použité teplotní mapy. Čím jsou produkty častěji kupované (vyšší ADU), tím je barva slotu do kterého jsou umístěny více vpravo (tzn. nejčastěji kupovaný produkt bude reprezentován červeně). Stejná teplotní mapa je použita také pro reprezentaci zatížení skladových prvků jako jsou dopravníky a lokace<sup>5</sup>.

- `initialWeights` – Počáteční váhy (alokace produktů do slotů).
- `saveWeightsPeriod` – Po kolika cyklech algoritmu se má ukládat nejlepší řešení.
- `maxTrialValue` – Maximální počet cyklů bez zlepšení fitness, než bude jedinec vyhozen z populace.
- `procCount` – Počet procesů, které se mají vytvořit pro účely simulace jedinců.
- `slotHeatReorder` – Produkty v každé z lokací jsou před každým optimalizačním krokem seřazeny podle toho, jak často jsou kupovány.
- `populationSize` – Velikost populace pro genetické algoritmy.
- `selectionSize` – Velikost vybrané části pro křížení pro genetické algoritmy.
- `eliteSize` – Velikost elite (neměnné) části pro genetické algoritmy.
- `probCrossover` – Pravděpodobnost křížení dvou jedinců.
- `probMutationInd` – Pravděpodobnost mutace jedince.
- `probMutationGene` – Pravděpodobnost mutace genu.
- `mutationFunctor` – Funkce, která se má použít pro mutaci jedince (`mutateOrdered`, `mutateInverse`).
- `selectionFunctor` – Funkce, která se má použít pro selekci jedinců (`selectRank`, `selectTrunc`, `selectTournam`, `selectRoulette`).
- `crossoverFunctor` – Funkce, která se má použít pro křížení jedinců (`crossoverOrdered`, `crossoverHeuristic`, `crossoverBinomical`).
- A mnoho dalších parametrů pro ABC, DE, PSO, SLAP a náhodné prohledávání...

<sup>5</sup>Obrázek převzat z <https://stackoverflow.com/a/20793850/8254699>.

Tabulka 5.1: Nejlepší konfigurace genetického algoritmu.

Parametr	Hodnota
Selekce	Turnaj
Mutace	Uspořádaná [7]
Křížení	Uspořádané [7]
Hodnota <i>trial</i>	10
Řazení v lokaci	Vypnuto
Pravděpodobnost křížení	0.6
Pravděpodobnost mutace jedince	0.4
Pravděpodobnost mutace genu	0.2

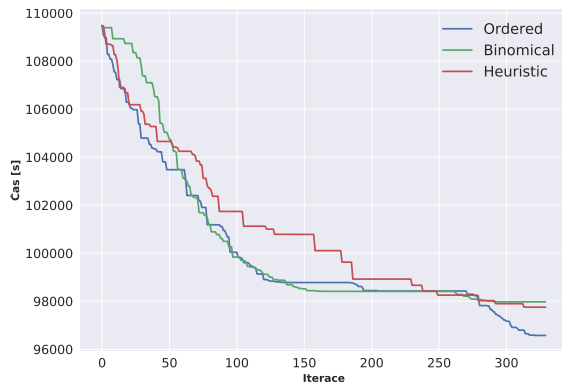
#### 5.4.4 Experimenty

Optimalizace skladu pomocí genetického algoritmu jako taková přinášela velmi dobré výsledky. Avšak stále se nabízely nové způsoby, jak dosáhnout lepších výsledků nebo alespoň podnětného porovnání.

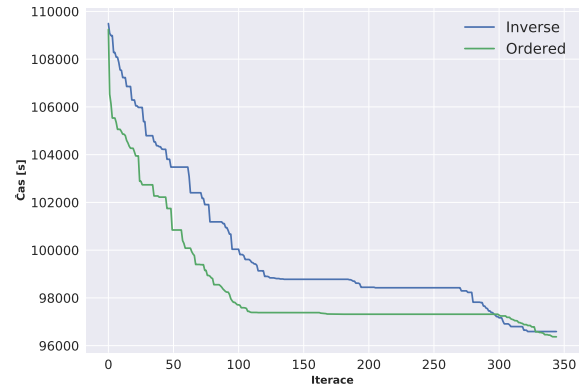
**Optimalizátory pro porovnání** Mimo čtyři optimalizátory rozložení produktů založené na evolučních algoritmech, byly implementovány také dva „optimalizátory“ určené čistě pro účely porovnání. První z nich je náhodné prohledávání. Dalším je optimalizátor založený na klasickém přístupu k rozřazení produktů ve skladu převzatý z práce [3]. Je založený na principu spočtení vzdálenosti každého slotu od vstupního a výstupního bodu skladu, a jejich seřazení. Dále spočtení, jak často jsou kupovány jednotlivé produkty a jejich seřazení. Následně jsou tyto seřazené produkty namapovány na seřazené sloty lokaci (nejčastěji kupovaný produkt do nejvýhodnějšího slotu, atd.), což by teoreticky mělo vést k nejvýhodnější alokaci. Porovnání výsledků optimalizace pomocí všech algoritmů na stejném modelu skladu lze vidět na snímku 6.1, ze kterého vyšel jako nejlepší genetický algoritmus, a zmíněný přístup je označen jako **Batista a spol.**

**Třídění produktů v lokacích** Optimalizace dokázala velmi dobře vyvážit zátěž mezi jednotlivými lokacemi, avšak uskupení produktů ve slotech lokací nebylo úplně ideální. Proto bylo cílem tohoto experimentu řadit produkty ve slotech lokací podle toho, jak často jsou kupovány. To by mělo zrychlit pickování objednávek a ulehčit práci pickerovi. Produkty nejsou přesouvány mezi lokacemi, pouze jsou seřazeny **v rámci lokace**. Porovnání výsledků lze vidět na snímku 5.13. Výsledek při seřazování byl avšak značně horší a tedy se seřazování dále nepoužívalo.

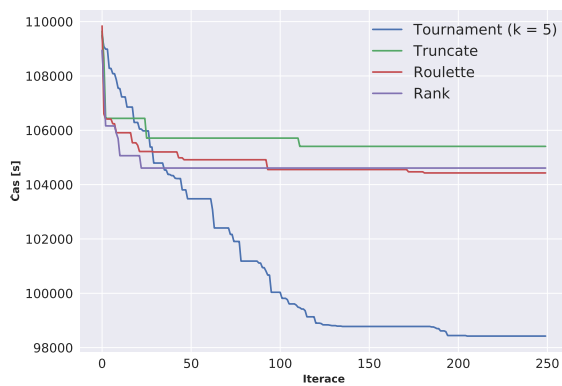
**Kombinování a úprava evolučních algoritmů** Nejúspěšnější experiment vznikl kombinací algoritmu genetických algoritmů a vlastnosti algoritmu umělých včelstev. Algoritmus umělých včelstev pro každé řešení problému udržuje hodnotu *trial*, která udává počet kroků algoritmu, ve kterých se dané řešení nezlepšilo. Pokud tato hodnota dosáhne před-definované hodnoty, je toto řešení nahrazeno novým náhodně vygenerovaným řešením. Problém u genetických algoritmů byl, že se zasekávaly v lokálních minimech, a proto byly doplněny o tuto vlastnost a poté dosahovaly značně lepších výsledků. Porovnání lze vidět na snímku 5.12.



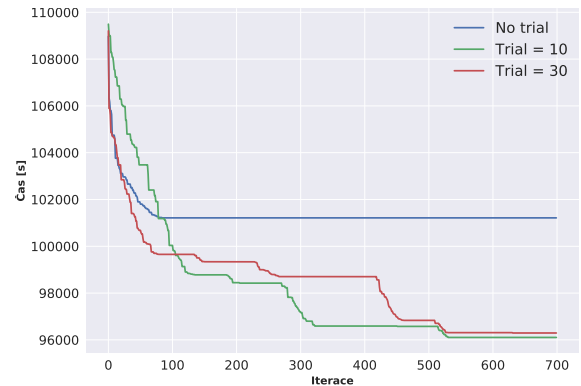
Obrázek 5.9: Experiment porovnávající různé operátory křížení.



Obrázek 5.10: Experiment porovnávající různé operátory mutace.



Obrázek 5.11: Experiment porovnávající různé operátory selekce.

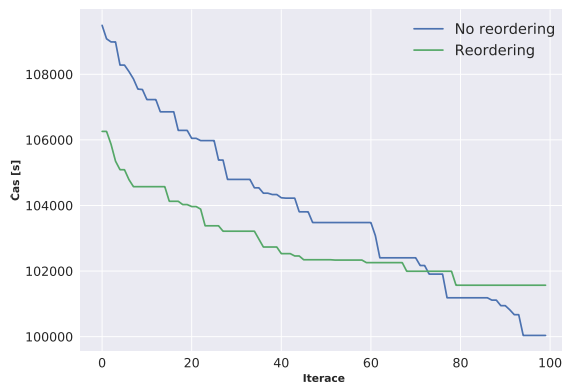


Obrázek 5.12: Experiment porovnávající různé hodnoty *trial*.

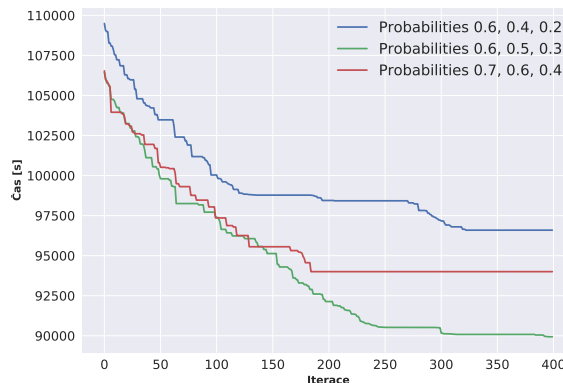
**Změna parametrů a vah** Každý z nástrojů poskytuje bohatou nabídku konfigurací. U simulátoru je možné například zrychlovat/zpomalovat dopravníky, pracovníky či příchody objednávek. U těchto hodnot je však vhodné nechat realistickou konfiguraci pro co největší přiblížení izomorfismu s reálným systémem. Optimalizátor umožňuje mimo obecné parametry jako ukládání vah také nastavení různých parametrů evolučních algoritmů. Bylo experimentováno s různými funkcemi pro křížení, mutaci, selekci a dalšími parametry, což značně ovlivňovalo optimalizaci. Pomocí experimentů bylo zjištěno, že nejlepších výsledků dosahuje právě genetický algoritmus a byla nalezena jeho optimální konfigurace, kterou lze najít v tabulce 5.1 a v XML souboru přiloženému k aplikaci, aby si jej uživatelé mohli jednoduše načíst. Grafické porovnání lze vidět na snímcích 5.9 až 5.14.

### 5.4.5 Průběh optimalizace

Ačkoli operace prováděné v evolučních algoritmech jsou zpravidla velmi jednoduchého rázu, v případě této práce velké množství jedinců implikovalo velké množství potřebných simulací a tedy potřebu vysokého výpočetního výkonu pro dosažení rozumné doby optimalizace komplexnějších modelů skladu. Tento problém byl z části řešen tvorbou konfigurovatelného počtu procesů provádějící simulace paralelně (5.2.3) a „eliminací“ duplicitních jedinců (5.2.4),



Obrázek 5.13: Experiment porovnávající použití uspořádávání a bez použití uspořádávání.



Obrázek 5.14: Experiment porovnávající různé pravděpodobnosti provedení křížení a mutace.

avšak stále byla optimalizace na osobním počítači velmi pomalá, nehledě na potřebu neustálého napájení. Z těchto důvodů bylo využito výpočetních zdrojů MetaCentra [16].

MetaCentrum je virtuální organizace poskytující bezplatné výpočetní a úložné kapacity pro členy akademické obce. Uživatelé si mohou na čelním uzlu připravit data a požádat o naplánování úlohy pomocí PBS (*Portable batch system*). Úlohy lze tvořit buď interaktivní nebo automatizované, ty však vyžadují dávkový soubor. Po přidělení požadovaných prostředků (s většími nároky na zdroje roste i doba než jsou prostředky přiděleny) je úloha spuštěna na jednom z výpočetních uzlů. Požádat o výpočetní zdroje a spuštění úlohy definované dávkovým souborem lze následujícím způsobem:

```
qsub -l select=1:ncpus=64:mem=1gb:scratch_local=1gb:walltime=24:00:00 ga.sh
```

Tento příkaz do fronty vloží požadavek na 64 procesorů, 1GB paměti RAM, 1GB prostoru na disku a dobu alokace těchto prostředků na 24 hodin. Po přidělení prostředků je proveden obsah dávkového souboru `ga.sh` na výpočetním uzlu – to zpravidla zahrnovalo nakopírování zdrojových kódů aplikace *Warehouse Manager* na výpočetní uzel, sestavení programu, použití požadovaného konfiguračního souboru a začátek trénování. V konfiguračním souboru bylo možné nastavit počáteční váhy modelu, a nezačínat tak trénování vždy od znovu. V průběhu výpočtu se průběžně podle konfigurace ukládaly váhy (aby nebyly ztraceny dosažené výsledky v případě pádu aplikace) a po dokončení úlohy nakopírování těchto dosažených výsledků zpět na úložiště.

## 5.5 Warehouse Manager

Grafická aplikace, ve výsledku nazvaná Warehouse Manager disponuje mimo původní účel (t.j. tvorba 2D modelu skladu), také veškerými funkcionalitami implementovanými v rámci této práce. To znamená veškerou kontrolu nad simulátorem, generátorem, pathfinderem i optimalizátorem. To umožňuje možnost plného využití této práce i technicky méně zdatným jedincům, jako např. logistickým manažerům, zcela bez nutnosti využít příkazovou řádku.

### 5.5.1 Layout aplikace

Rozložení grafické aplikace bylo vytvořeno za pomoci aplikace **Qt Designer**: v horní liště lze najít tlačítka pro ovládání, jako např. vytvoření nového modelu skladu, načtení již existujícího modelu ze souboru, atd. Dále se v této liště nachází jednotlivé prvky/zařízení skladu, které může uživatel využít pro vytvoření modelu skladu. Na levé straně aplikace jsou záložky, které slouží pro import, export a náhled dat využívaných jednotlivými nástroji, jako jsou (zleva): objednávky pro trénování a 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 exportovat do XML.
- **Statistiky** – Zde jsou obsaženy grafy, do kterých jsou postupně doplňovány hodnoty produkované daným nástrojem.
- **Řízení** – Poskytuje tlačítka pro kontrolu jednotlivých nástrojů, jako je spuštění či zastavení. Ve chvíli, kdy je spuštěn jeden z uvedených nástrojů, jsou veškerá kontrolní tlačítka, stejně tak úprava modelu skladu, zakázána. A to aby uživatel nijak nenarušil běh daného nástroje. Naopak se povolí tlačítko pro zastavení běhu nástroje. Zastavení nebo dokončení práce nástrojem opět povolí úpravu modelu skladu a veškerá tlačítka.

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:

- 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.
- Přiblížení a oddálení scény/modelu skladu<sup>6</sup>.

### 5.5.2 Tvorba modelu skladu

Uživatel, který chce aplikaci plně využít ve svůj prospěch, musí být schopen vytvořit model skladu dle jeho potřeb. Zejména pro tyto účely byla grafická aplikace navržena, a později doplněna o veškerou ostatní funkcionalitu. Pro implementaci plochy pro tvorbu modelu skladu byla využita grafická scéna (**QGraphicsScene**<sup>7</sup>) a grafický pohled (**QGraphicsView**<sup>8</sup>). Do grafické scény jsou postupně umísťovány prvky skladu odvozené od grafických prvků

<sup>6</sup>Inspirováno <https://stackoverflow.com/a/19114517/8254699>.

<sup>7</sup><https://doc.qt.io/qt-5/qgraphicscene.html>

<sup>8</sup><https://doc.qt.io/qt-5/qgraphicsview.html>

(`QGraphicsItem`<sup>9</sup>), doplňující je o specifickou funkcionalitu. Při implementaci byly využity volně dostupné doplňky Qt: `QDarkStyleSheet`<sup>10</sup>, `QCustomPlot`<sup>11</sup> a `QtEditableItems`<sup>12</sup>.

## Grafická scéna

Grafický pohled umožňuje zobrazení grafické scény ve widgetu. Grafická scéna slouží jako kontejner grafických prvků a v kombinaci s grafickým pohledem slouží k vizualizaci takovýchto prvků na 2D povrchu. Pro účely přepsání jistých vlastností grafické scény (jako např. zobrazení kontextového menu) byla tato třída odvozena. Stejně tak grafické prvky, které byly do scény umísťovány, byly odvozeny a velmi značně přepsány pro daný *use-case*. Takovéto odvození umožnilo například vytvoření manipulátorů pro změnu velikosti prvků a také jejich rotaci, viz 5.15. Uživatel je na začátku práce dotázán na velikost skladu v metrech. Tyto rozměry jsou interně přepočteny a je vytvořeno ohraničení skladu bílými čarami, mřížkou pro jednodušší manipulaci s prvky a také měřítkem (vlevo nahoře). Délka měřítka je vždy jedna pětina šířky grafické scény, a číslo nad ní udává, kolik metrů ve skutečnosti tato délka (část scény) reprezentuje.

Grafický pohled a scéna byly doplněny mimo jiné také o možnost přibližování a oddalování, bez kterých by byla aplikace jen velmi těžce použitelná. Tohoto lze dosáhnout pomocí přidržení klávesy `CTRL` a pohybem rolovacího tlačítka myši. Při této akci se také přepočítává měřítko vůči reálnému světu. Pro usnadnění tvorby komplexních modelů skladu byla implementována možnost hromadného označování prvků a následně možnost kopírování a přemísťování označených prvků, což může velmi usnadnit tvorbu modelu.

## Grafické prvky

Jak již bylo zmíněno, grafické prvky jsou odvozeny z `QGraphicsItem`. Poté jsou z této odvozené třídy odvozeny další třídy pro každý typ zařízení, a sice:

- `UiWarehouseItemLocation_t` – Reprezentuje lokace.
- `UiWarehouseItemConveyor_t` – Reprezentuje dopravníky.
- `UiWarehouseItemGate_t` – Reprezentuje vstupní a výstupní brány skladu.

Veškeré grafické prvky jsou reprezentovány jako ukazatele na dynamicky alokovanou paměť, tzv. „hromadu“ (ang. *heap*), a to z důvodu aby grafické prvky existovaly i po opuštění rozsahu části programu, kde jsou vytvořeny. O uvolnění paměti se poté starají mechanismy frameworku Qt díky specifikaci rodičovských prvků, až po hlavní okno. Tzn. při destrukci rodiče jsou uvolněni veškerí následníci, poté jejich následníci, a tak dále. Jednotlivé třídy jsou rozšířeny o funkcionalitu specifickou pro dané zařízení skladu. Např. lokace je rozšířena o mřížku slotů, tedy kontejner instancí třídy `UiWarehouseSlot_t` atd.

Propojování grafických prvků lze provádět na základě portů – `UiWarehousePort_t` (oranžové šipky na každém z prvků, viz 5.15). Po připojení daného portu k jinému se tyto spojené porty skryjí a ukáží se opět při odpojení prvku. Každé ze zařízení má jiný počet portů, dle jeho způsobu použití.

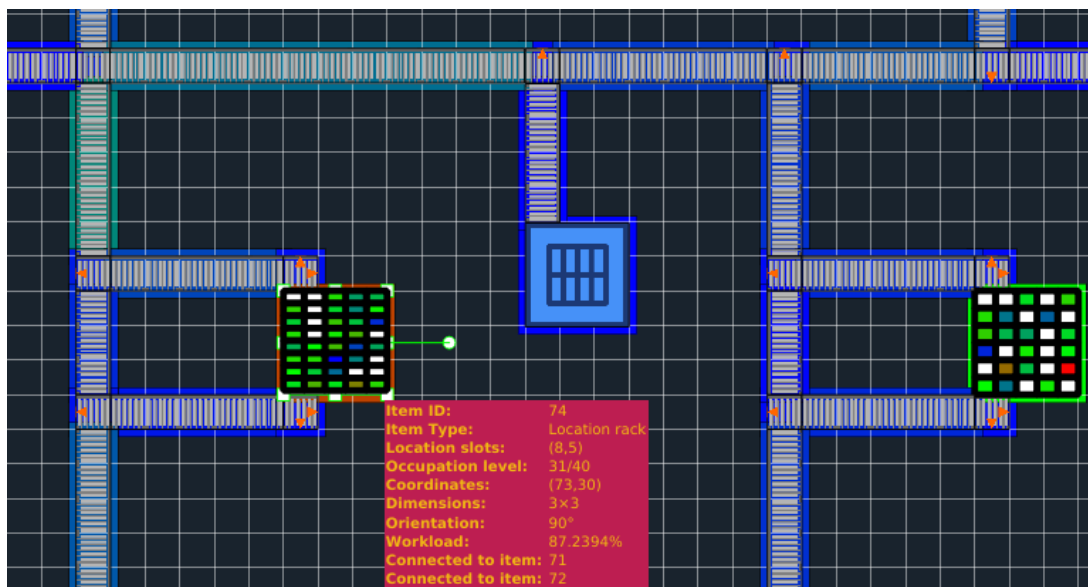
<sup>9</sup><https://doc.qt.io/qt-5/qgraphicsitem.html>

<sup>10</sup><https://github.com/ColinDuquesnoy/QDarkStyleSheet>

<sup>11</sup><https://www.qcustomplot.com>

<sup>12</sup><https://github.com/fadyosman/QtEditableItems>





Obrázek 5.15: Zobrazení informací o zařízení skladu pomocí najetí kurzorem (přesně se jedná o lokaci). Dále lze na snímku vidět porty sloužící k propojování prvků (oranžové šipky) a nakonec také manipulátory, pomocí kterých lze vybraný prvek rotovat a měnit jeho velikost.

Informace o grafickém prvku lze zobrazit pomocí najetí myší na daný prvek, viz 5.15. V zobrazené tabulce lze najít veškeré informace o prvku, ať už jde o pozici, velikost a natočení, nebo specifické informace jako zatížení prvku nebo úroveň obsazenosti lokace.

### Serializace modelu skladu

Model skladu může být poměrně komplexní, a jeho opětovné vytváření při každém spuštění by nedávalo smysl. Proto je možné model serializovat (uložit) do souboru ve formátu XML (pomocí knihovny `libxml2`), a také jej ze souboru deserializovat (načíst) do grafického rozhraní. Velmi jednoduchý příklad lze vidět na následujícím výpisu.

```

1 <WarehouseLayout dim_x="20" dim_y="5" ratio="50"/>
2
3 <WarehouseItem id="0" type="1" x="50" y="50" w="750" h="50" o="0"/>
4   <WarehousePort id="0"/>
5   <WarehousePort id="1" conn_id="0"/>
6 </WarehouseItem>
7
8 <WarehouseItem id="1" type="0" x="801" y="50" w="150" h="150" o="0"/>
9   <LocationRackDimensions slots_x="10" slots_y="10"/>
10  <WarehousePort id="0" conn_id="0"/>
11  <WarehousePort id="1"/>
12 </WarehouseItem>
13
14 <WarehouseConnection id="0"/>
15   <WarehousePortTo item_id="0" port_id="1"/>
16   <WarehousePortFrom item_id="1" port_id="0"/>
17 </WarehouseConnection>

```

Výpis 5.1: Serializovaný model skladu ve formátu XML. První element obsahuje informace o layoutu, jako je výška, šířka a kolik jeden metr představuje bodů ve scéně. Další element představuje dopravník i s informacemi, kde se ve scéně nachází, jak je velký a jak je natočený. Následuje lokace, která mimo pozici apod. obsahuje také informaci o slotech. Následuje jediný propoj vytvořený mezi těmito dvěma prvky.

## 5.6 Moduly programu

V této sekci jsou velmi krátce popsány jednotlivé moduly programu, které jsou v různé míře využity všemi implementovanými nástroji. Operátor rozsahu platnosti zde identifikuje jednotlivé jmenné prostory, ve kterých jsou moduly uloženy:

- `whm::ConfigParser_t` – Tento modul slouží pro serializaci a de-serializaci konfiguračních souborů v XML, a umožňuje získávání konfiguračních hodnot pomocí šablonové metody.
- `whm::Logger_t` – Komplexní modul pro zaznamenávání běhu aplikace na výstup či do souboru.
- `whm::utils::*` – Utility (šablonové funkce) plošně využívané v celé aplikaci.
- `whm::WarehouseDataGenerator_t` – Modul který obstarává generování zákaznických objednávek.
- `whm::WarehouseSimulatorSIMLIB_t` – Modul který obstarává simulaci skladu pomocí knihovny SIMLIB/C++.
- `whm::WarehousePathFinder_t` – Modul, který rekurzivně vyhledá veškeré cesty ve skladu, uloží do vhodné datové struktury a umožňuje nalezení nejkratší cesty mezi lokacemi, apod.
- `whm::WarehousePathFinderACO_t` – Implementace mravenčího algoritmu pro nalezení nejkratší cesty objednávky skrze celý sklad.

- `whm::WarehouseOptimizer{Base|GA|DE|ABC|PSO|SLAP|RAND}_t` – Implementace čtyř evolučních algoritmů v diskrétním prostoru, metodiky SLAP, náhodného prohledávání a bázové třídy, ze které jsou všechny optimalizátory odvozeny. Optimalizátory využívají definici řešení problému definovaného v modulu `whm::Solution_t`.
- `whm::WarehouseOrder_t` – Modul implementující zákaznickou objednávku.
- `whm::WarehouseOrderLine_t` – Modul implementující část (položku) zákaznické objednávky.
- `whm::WarehouseLocationRack_t` – Modul implementující lokaci, která se skládá ze slotů a jsou v ní uloženy produkty.
- `whm::WarehouseLocationSlot_t` – Modul implementující sloty, ze kterých sestává lokace.
- `whm::WarehousePort_t` – Modul implementující port pomocí kterého lze propojovat jednotlivé prvky.
- `whm::WarehouseConnection_t` – Modul implementující propojení prvků ve skladu.
- `whm::WarehouseItem_t` – Modul implementující prvek/zařízení skladu.
- `whm::WarehouseLayout_t` – Modul poskytující *singleton*, který obsahuje veškeré důležité informace o celém skladu.
- `whm::gui::*` – Grafická nastavba textového klienta (souborů výše), která sestává z cca. 15 modulů, mimo jiné také z vláken, které složí k provádění náročných operací jako je např. optimalizace – aby neblokovaly hlavní smyčku (grafické okno). Dále pak z odvozených grafických prvků, scény, apod. Pokud je definován symbol `WHM_GUI`, jsou některé moduly v seznamu nahoře doplněny o zpětné volání do grafického rozhraní.

## Kapitola 6

# Vyhodnocení účinnosti vytvořených nástrojů

Cílem této práce bylo zejména optimalizovat sklad tak, aby se maximalizovala propustnost skladu, tedy počet zpracovaných objednávek za jednotku času. To lze chápat také jako zpracování daných objednávek v co nejkratším možném čase. Vyhodnocení je rozděleno na tři části: optimalizaci rozložení produktů, cesty a nakonec jejich kombinaci.

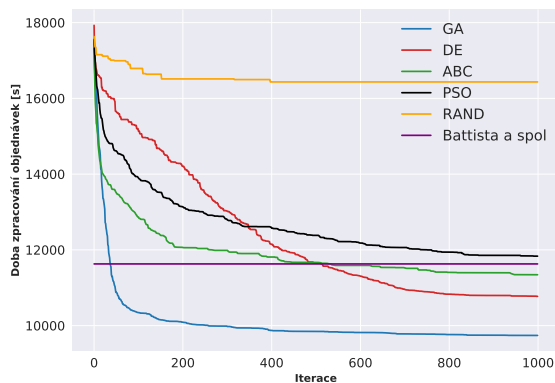
### 6.1 Optimalizace rozložení produktů

Jak již bylo zmíněno, při optimalizaci tohoto problému se minimalizuje doba potřebná ke zpracování sady objednávek. Na grafu na obrázku 6.1 lze vidět průběh optimalizace pomocí evolučních algoritmů na problému rozřazení 150 produktů do 200 slotů, což lze vyjádřit jako kombinatorický problém:

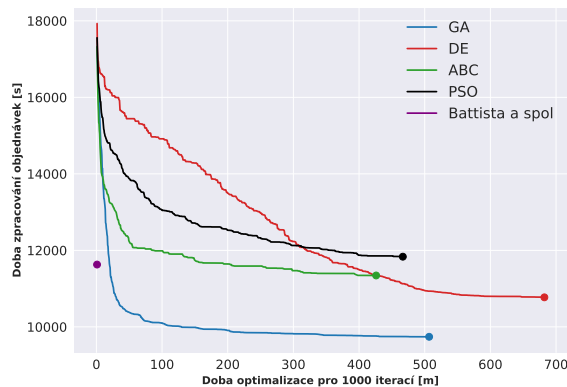
$$V_{150}(200) = \frac{200!}{50!} = 2.593067e + 310.$$

Dále je na grafu vidět také průběh náhodného prohledávání prostoru (RAND) a klasické metodiky (Battista a spol.) použité v práci [3] – tato metodika je jednokrokový výpočet mapující nejčastěji kupované produkty do nejvýhodnějších slotů, a proto je konstantní. Nejlépe si vedl genetický algoritmus, kterému se povedlo snížit dobu potřebnou ke zpracování 1000 objednávek téměř na polovinu (57%). Stejnou optimalizaci avšak v časové doméně (*trade-off* úspěšnosti a času) lze vidět na obrázku 6.2. Vyhodnocení optimalizovaných modelů na testovací sadě vždy cca. odpovídalo úrovni optimalizace na trénovací sadě objednávek. Při kombinatorickém nárůstu možných řešení a při zachování nastavení optimalizátoru a délky optimalizace se kvalita optimalizace snižuje, viz obrázek 6.3 a obrázek 6.4.

Při analýze průběhů trénování lze vidět, že náhodné prohledávání je z pohledu optimalizace téměř bezvýznamné. Genetický algoritmus velmi rychle konverguje na začátku, ale zhruba od iterace 400 už se vůbec nezlepšuje. Další iterace by tomuto algoritmu tedy už nejspíše nepomohly. Na druhou stranu algoritmus umělých včelstev, optimalizace rojem částic a diferenční evoluce konvergují mnohem pomaleji a při dalších iteracích by se výsledek nejspíše dále lehce zlepšoval. Algoritmus optimalizace rojem částic je z uvedených evolučních algoritmů pro tuto problematiku nejméně vhodný a jako jediný dosahuje horších výsledků, než klasická metoda Battista a spol [3]. Nejlépe tedy dle očekávání dopadl genetický algoritmus, a sice na všech instancích problému, kterému se podařilo dobu potřebnou pro zpracování objednávek snížit téměř na polovinu.



Obrázek 6.1: Graf průběhu optimalizace pomocí čtyř evolučních algoritmů, metodiky Battista a spol. [3] a náhodného prohledávání. Ve všech případech byl použit stejný model skladu (který lze vidět na snímku 5.2) a stejné trénovací objednávky. Na vodorovné ose jsou iterace algoritmu a na svislé ose je doba potřebná pro zpracování sady trénovacích objednávek v sekundách. Jedná se o aritmetický průměr pěti běhů na metodu.



Obrázek 6.2: Průběh optimalizace z grafu na obrázku 6.1 v časové doméně, tzn. jak dlouho trvalo 1000 iterací jednotlivých metod. Pokud bychom nebrali v potaz genetické algoritmy, nejvhodnější by byla diferenční evoluce, pokud bychom však měli na optimalizaci méně než 400 minut, vhodnější by byl algoritmus umělých včelstev a do 300 minut by vycházel lépe dokonce i algoritmus optimalizace rojem částic.

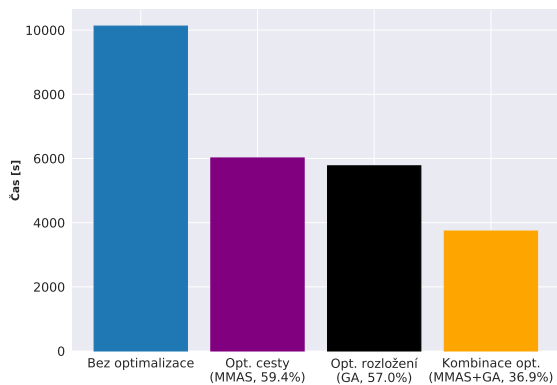
Vzhledem k tomu, že z porovnání algoritmů pro řešení problematiky SLAP vyšel jako nejlepší genetický algoritmus, pro další experimenty s optimalizací rozložení produktů byl využíván už pouze tento algoritmus. Na snímcích 5.9 až 5.14 lze vidět experimenty s různými parametry genetického algoritmu. Z těchto experimentů bylo možné najít nejlepší konfiguraci optimalizátoru, která byla nadále používána a kterou lze vidět v tabulce 5.1.

## 6.2 Optimalizace cesty

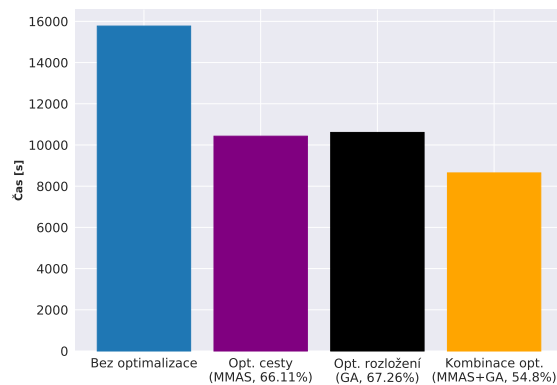
V rámci této práce byl jako doplněk implementován také nástroj pathfinder, který pomocí mravenčího algoritmu dokáže nalézt optimální cestu objednávky skrze sklad. Na obrázku 6.5 lze vidět průběh hledání optimální cesty na skladu o velikosti 200 lokací, což už je z hlediska skladových politik více než obrovský sklad a tedy nemá význam řešit tento problém pro větší modely skladu. Algoritmus v závislosti na konfiguraci dokáže nalézt optimální řešení (cestu s nejkratší vzdáleností) pro zmíněný sklad do 400 iterací algoritmu. U skladů typických velikostí je optimální cesta nalezena maximálně do 100 iterací algoritmu.

## 6.3 Kombinace optimalizací

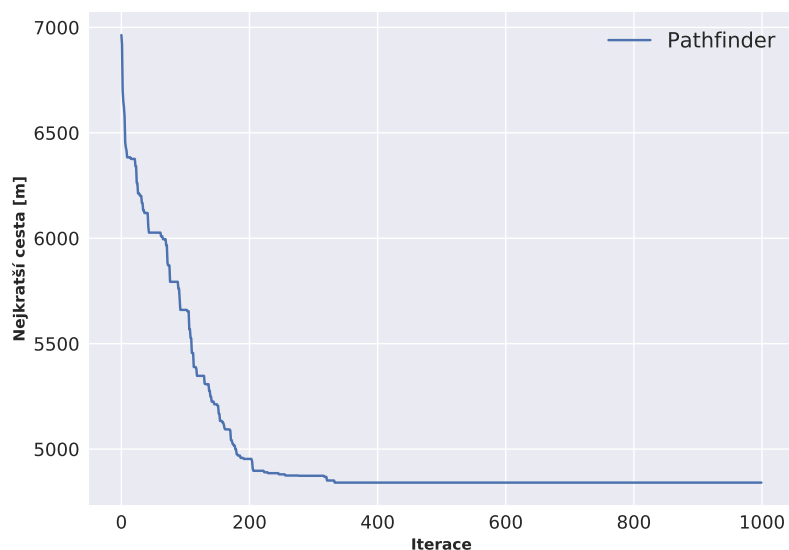
Nástroj pathfinder lze použít v rámci simulátoru pro hledání optimálních cest pro objednávky. Vzhledem k tomu, že optimalizátor rozložení používá simulátor pro aproximaci kvality řešení, lze použít všechny tři nástroje zároveň a optimalizovat jednak rozložení produktů a zároveň délku cesty. Na grafu 6.3 lze vidět porovnání (na problému 200 produktů, 400 slotů): neoptimalizovaný sklad, nejlepší dosažené výsledky samostatných optimalizací a nakonec kombinace těchto optimalizací. Jak lze vidět, kombinací těchto optimalizátorů



Obrázek 6.3: Sloupcový graf porovnávající optimalizaci cesty pomocí  $\mathcal{MMAS}$ , optimalizaci rozložení produktů pomocí genetického algoritmu a nakonec jejich kombinaci (sklad 200 produktů, 400 slotů).



Obrázek 6.4: Sloupcový graf porovnávající optimalizaci cesty pomocí  $\mathcal{MMAS}$ , optimalizaci rozložení produktů pomocí genetického algoritmu a nakonec jejich kombinaci (sklad 750 produktů, 1000 slotů).



Obrázek 6.5: Graf průběhu hledání optimální cesty pomocí algoritmu  $\mathcal{MMAS}$  na velkém skladu. Jedná se o aritmetický průměr pěti běhů.

lze dosáhnout ještě lepších výsledků, avšak za velice vysokou cenu doby trénování – optimalizace cesty trvá jednotky minut, optimalizace rozložení jednotky hodin (viz graf na obrázku 6.2) a jejich kombinace pak desítky hodin. Kombinace optimalizací trvajících desítky hodin už není v souladu s fungováním moderních skladů, které musí být schopny relativně rychle reagovat na změny požadavků na trhu.

# Kapitola 7

## Závěr

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 objednávek 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 kombinatorickém nárůstu možných řešení a při zachování nastavení optimalizátoru a délky optimalizace se kvalita optimalizace rozložení snižuje – nejvýše však o **10%**.

Přínosem této práce je úplný grafický nástroj, jenž dosahuje velmi dobrých výsledků a poskytuje mnoho užitečných funkcí v oblasti skladového hospodářství. 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 – celkovou dobu zpracování sady objednávek.

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).

# Literatura

- [1] Genetic algorithms for modelling and optimisation. *Journal of Computational and Applied Mathematics*. 2005, sv. 184, č. 1, s. 205–222. DOI: <https://doi.org/10.1016/j.cam.2004.07.034>.
- [2] A.E. EIBEN, J. E. S. *Introduction to Evolutionary Computing*. Berlin: Springer-Verlag Berlin Heidelberg, 2015. ISBN 978-3-662-44874-8.
- [3] BATTISTA, C., FUMI, A., GIORDANO, F. a SCHIRALDI, M. Storage Location Assignment Problem: implementation in a warehouse design optimization tool. *Proceedings of the Conference Breaking Down the Barriers between Research and Industry*. Leden 2011.
- [4] BORNA, K., KHEZRI, R. a YIU, C. A combination of genetic algorithm and particle swarm optimization method for solving traveling salesman problem. *Cogent Mathematics*. Prosinec 2015, sv. 2. DOI: 10.1080/23311835.2015.1048581.
- [5] BOTTANI, E., CECCONI, M., VIGNALI, G. a MONTANARI, R. Optimisation of storage allocation in order picking operations through a genetic algorithm. *International Journal of Logistics Research and Applications*. Taylor Francis. 2012, sv. 15, č. 2, s. 127–146. DOI: 10.1080/13675567.2012.694860.
- [6] BRABAZON ANTHONY, M. S. *Natural Computing Algorithms*. Berlin: Springer-Verlag Berlin Heidelberg, 2015. ISBN 978-3-662-43631-8.
- [7] CARYL, M. *Travelling Salesman Problem* [online]. Naposledy navštíveno 3. 1. 2021. Dostupné z: <http://www.permutationcity.co.uk/projects/mutants/tsp.html>.
- [8] CHEN, L., LANGEVIN, A. a RIOPEL, D. The storage location assignment and interleaving problem in an automated storage/retrieval system with shared storage. *International Journal of Production Research - INT J PROD RES*. Prosinec 2007, sv. 48. DOI: 10.1080/00207540802506218.
- [9] COLLA, V. a NASTASI, G. Modelling and Simulation of an Automated Warehouse for the Comparison of Storage Strategies. In: *Modelling Simulation and Optimization*. InTech, Feb 2010. DOI: 10.5772/7658.
- [10] HOFFMAN, K. L. a PADBERG, M. Traveling Salesman Problem (TSP) Traveling salesman problem. In: GASS, S. I. a HARRIS, C. M., ed. *Encyclopedia of Operations Research and Management Science*. New York, NY: Springer US, 2001, s. 849–853. DOI: 10.1007/1-4020-0611-X\_1068. ISBN 978-1-4020-0611-1.



- [11] JUAN JOSÉ ROJAS REYES, J. R. M.-T. The storage location assignment problem: A literature review. *International Journal of Industrial Engineering Computations*. 2019, sv. 10, č. 2, s. 199–224. DOI: 10.5267/j.ijiec.2018.8.001.
- [12] KARABOGA, D. a BASTURK, B. A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm. *Journal of Global Optimization*. Listopad 2007, sv. 39, s. 459–471. DOI: 10.1007/s10898-007-9149-x.
- [13] KHAN, I. a MAITI, M. K. A swap sequence based Artificial Bee Colony algorithm for Traveling Salesman Problem. *Swarm and Evolutionary Computation*. 2019, sv. 44, s. 428 – 438. DOI: <https://doi.org/10.1016/j.swevo.2018.05.006>. ISSN 2210-6502.
- [14] LINGARAJ, H. A Study on Genetic Algorithm and its Applications. *International Journal of Computer Sciences and Engineering*. Říjen 2016, sv. 4, s. 139–143.
- [15] LIU, T. a MAEDA, M. 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)*. 2014, s. 81–86. DOI: 10.1109/SCIS-ISIS.2014.7044726.
- [16] METACENTRUM. *MetaCentrum VO – virtuální organizace pro celou akademickou obec* [<https://metavo.metacentrum.cz/cs>]. [Online; navštíveno 07.12.2020].
- [17] NASTASI, C. V. C. S. e. a. Implementation and comparison of algorithms for multi-objective optimization based on genetic algorithms applied to the management of an automated warehouse. *Journal of Intelligent Manufacturing*. 2018, sv. 29, s. 1545 – 1557. DOI: <https://doi.org/10.1007/s10845-016-1198-x>.
- [18] PAN, J. C.-H., SHIH, P.-H., WU, M.-H. a LIN, J.-H. A storage assignment heuristic method based on genetic algorithm for a pick-and-pass warehousing system. *Computers Industrial Engineering*. 2015, sv. 81, s. 1 – 13. DOI: <https://doi.org/10.1016/j.cie.2014.12.010>. ISSN 0360-8352.
- [19] STOLTZ, E. *Evolution of a salesman: A complete genetic algorithm tutorial for Python* [online]. Naposledy navštíveno 11. 5. 2020. Dostupné z: <http://towardsdatascience.com/6fe5d2b3ca35>.
- [20] THOMAS STÜTZLE, H. H. H. *MAX–MIN* Ant System. *Future Generation Computer Systems*. 2000, sv. 16, č. 8, s. 889–914. DOI: [https://doi.org/10.1016/S0167-739X\(00\)00043-1](https://doi.org/10.1016/S0167-739X(00)00043-1). ISSN 0167-739X.

## Příloha A

# Obsah příloženého paměťového média

- `src` – Zdrojové kódy vytvořené aplikace `WarehouseManager`.
- `utils` – Skripty používané v průběhu práce, zejména ke trénování na výpočetním clusteru a k vyhodnocení.
- `doc` – Plakát použitý k prezentaci této práce a video prezentující dosažené výsledky.
- `README` – Více obsáhlá příručka v anglickém jazyce (ve formátu `md` – *markdown*).
- `report` – Adresář obsahující zdrojové kódy tohoto dokumentu ve formátu `LATEX`, a také jejich přeloženou verzi ve formátu `PDF`.

## Příloha B

# Manuál

Sestavení a použití aplikace WarehouseManager na linuxovém OS:


1. Instalace grafického frameworku Qt5 a knihovny SIMLIB/C++ dle jejich dokumentace.
2. Spuštění kompilace pomocí příkazu `make` v kořenovém adresáři projektu.
3. Nastavení konfigurace v grafickém rozhraní nebo v souborech v adresáři `cfg/`.
4. Po úspěšné kompilaci lze provést spuštění nástrojů následovně:

```
./whm_gui
./whm_gen -o orders.xml -a articles.csv
./whm_sim -o orders_test.xml -i locations.csv \
-l layout.xml
./whm_paf -o orders_test.xml -i locations.csv \
-l layout.xml [-s]
./whm_opt -o orders_train.xml -i locations.csv \
-l layout.xml -a articles.csv -O 1-6
```

5. Data pro testování lze najít v adresáři `data/` a jeho podadresářích.
6. Výsledky lze při použití příkazové řádky vidět na standardním výstupu a v případě použití grafické aplikace přímo v ní.
7. Pro více informací lze nahlédnout do `README` nebo vyvolat pomoc přepínačem `-h`.

# Příloha C

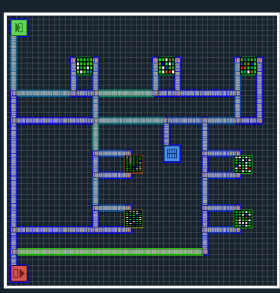
## Plakát



Nástroj pro pokročilou simulaci a optimalizaci skladových operací  
Autor: Bc. Filip Kočica | Vedoucí: Ing. Oldřich Kodym

1

Lísta  
1. Akce  
2. Měd  
3. Prvky



Konfigurace  
Statistický grafy  
Kontrola

Záložky s daty (objednávky, produkty, sloty)

Editor pro tvorbu 2D modelu skladu (zatížení prvků a aktuální alokace produktů)

Záložky se čtyřmi nástroji popsanými níže (každý s vlastní konfigurací a statistikami)

### # Generátor

- Generování syntetických zákaznických objednávek (datových sad)
- Použití uživatelem definovaných pravděpodobnostních modelů
- Vytvoření objednávek pro trénování a testování dalších nástrojů

### # Simulátor

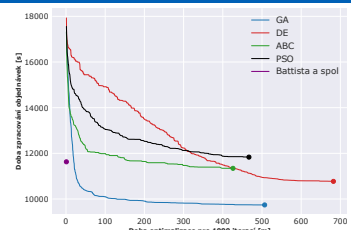
- Realistická simulace zpracování objednávek ve vytvořeném skladu
- Použití jako aproximace kvality řešení v optimalizátoru
- Zvýraznění zatížených prvků, rozsáhlé statistiky (viz screenshot)

### # Optimalizátor

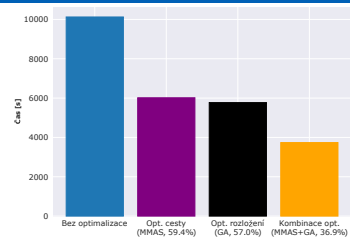
- Optimalizace rozložení produktů pro zvýšení propustnosti skladu
- Použití a srovnání čtyř evolučních algoritmů (GA, DE, ABC, PSO)
- Téměř dvojnásobné zvýšení propustnosti skladu (viz grafy)

### # Pathfinder

- Nalezení optimální cesty objednávky skrze sklad
- Použití dalšího evolučního algoritmu **MIN-MAX Ant System**
- Nalezení optimální cesty skladem do 300 iterací i pro velké sklady



Graf optimalizace rozložení produktů pomocí čtyř evolučních algoritmů a jednorázového výpočtu Battista a spol. v časové doméně. Nejlépe si vedl genetický algoritmus a téměř dvojnásobně snížil dobu potřebnou ke zpracování sady 1000 trénovacích objednávek na ~57% původní doby. Jedná se vždy o průměr pěti běhů pro každou metodu.



Nástroj pathfinder dokáže nalézt optimální cestu skladem v malém počtu iterací a zrychlení zpracování sady objednávek je téměř dvojnásobné (~59.4%). Stejně tak optimalizátor rozložení produktů dokáže téměř dvojnásobně zrychlit zpracování objednávek (~57%). Jejich kombinací pak lze dosáhnout ještě lepších výsledků (~36.9%).

- Dobré výsledky optimalizace, rozsáhlé experimentální vyhodnocení implementovaných metod.
- Nespočet parametrů pro experimentování, včetně pěti implementovaných evolučních algoritmů.
- Nezávislost simulace a optimalizace na modelu skladu (uživatel si jej vytvoří v editoru dle potřeby).
- Rada užitečných funkcí pro identifikaci úzkých míst skladu, detekci zátěže jednotlivých prvků, apod.
- Nový způsob optimalizace kombinací state of the art technik a nové optimalizační kritérium.

