VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

ROZVRHOVÁNÍ ÚKOLŮ V LOGISTICKÝCH SKLADECH

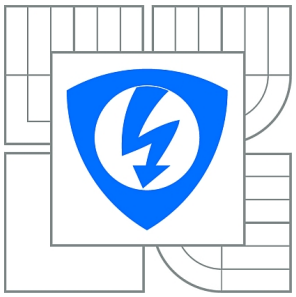
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. LUKÁŠ POVODA
AUTHOR

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

ROZVRHOVÁNÍ ÚKOLŮ V LOGISTICKÝCH SKLADECH

JOB SCHEDULING IN LOGISTIC WAREHOUSES

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

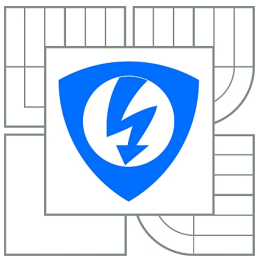
AUTOR PRÁCE — Bc. LUKÁŠ POVODA

AUTHOR

VEDOUCÍ PRÁCE — Ing. JAN KARÁSEK

SUPERVISOR

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Lukáš Povoda	***ID:***	125607
Ročník:	2	***Akademický rok:***	2013/2014

NÁZEV TÉMATU:

Rozvrhování úkolů v logistických skladech

POKYNY PRO VYPRACOVÁNÍ:

V teoretické části práce student zpracuje rešerši týkající se rozvrhovacích problému se zaměřením na meta-heuristické algoritmy s ohledem na praktické použití ve skladových prostorách. V praktické části práce student na základě historie reálného provozu skladových prostor extrahuje desítky testovacích případů z historických dat představující provoz skladu, vytvoří generátor syntetických případů simulující reálný provoz a tyto případy vyhodnotí s různým nastavením algoritmu. Dále student otestuje parametry fitness funkce z pohledu celkového času zpracování úloh, z pohledu počtu potenciálních kolizí a kombinace těchto faktorů. Student identifikuje části kódu, vhodné pro paralelizaci a tuto výhodu implementuje. Na základě známých pravidel pro optimalizaci rozvrhovacích problémů nastudovaných v teoretické části práce student provede návrh struktury tří vlastních pravidel, která budou kombinovat výhody logicky strukturovaných pravidel ověřených praxí a výhody náhodného prohledávání. Nakonec student provede simulaci reálných i syntetických případů pro všechna pravidla a srovnání s původním evolučním algoritmem. Implementace bude provedena v programovacím jazyce JAVA a JavaFX.

DOPORUČENÁ LITERATURA:

[1] Liberti, L.; Maculan, N.; Global Optimization: From Theory to Implementation, Springer, Feb 1 2006.
[2] Pinedo, M. L.; Scheduling: Theory, Algorithms, and Systems, Springer, 4th edition, January 6 2012.
[3] Zhang, G.; Gao, L.; Shi, Y.; “An Effective Genetic Algorithm for the Flexible Job-Shop Scheduling Problem,” Expert Systems with Applications, vol. 38, no. 4, pp. 3563–3573, April 2011.

Termín zadání:	10.2.2014	***Termín odevzdání:***	28.5.2014

Vedoucí práce: Ing. Jan Karásek
Konzultanti diplomové práce:

doc. Ing. Jiří Mišurec, CSc.
Předseda oborové rady

ABSTRAKT

Táto práca sa zaoberá problémom rozvrhovania a riadenia práce v logistických skladoch. Riadenie a rozvrhovanie práce je v súčastnosti často riešený problém, na ktorý neexistuje jednoduché a jednoznačné riešenie kvôli komplexnosti problému. Tento problém je nutné riešiť z dôvodu nedostatočnej efektivity práce pri vyššom zaťažení skladu, ako je tomu napr. v období vianočných sviatkov. V tejto práci sú opísané metódy využívané k riešeniu tohto problému, zameriava sa hlavne na využitie prehľadávacích algoritmov, evolučných algoritmov, konkrétne gramatikou riadeného genetického programovania. V práci je rozobraný problém riadenia a rozvrhovania práce na jednoduchom teoretickom príklade. Implementovaný algoritmus pre riešenie tohto problému bol podrobený testom inšpirovaných dátami z reálneho skladu, ako aj synteticky vytvoreným testom s vyšším počtom úloh a vyšším počtom pracovníkov. Syntetické testy boli generované náhodne. Všetky testy boli preto spustené viackrát a výsledky spriemerované. V závere práce sú uvedené výsledky úspešnosti algoritmu, ako aj optimálne nastavenia parametrov pre rôzne veľkosti problémov a požiadavky na riešenie. Použitý algoritmus bol rozšírený o výpočet vhodnosti daného jedinca s prihliadnutím na počet kolízií pri vykonávaní jednotlivých úloh, o aplikovanie prioritných pravidiel v priebehu genetického algoritmu, a niektoré časti algoritmu boli paralelizované.

KĽÚČOVÉ SLOVÁ

Riadenie práce, rozvrhovanie práce, logistika, algoritmus genetického programovania, evolučný algoritmus, gramatikou riadené genetické programovanie, heuristika

ABSTRACT

The main aim of this thesis is flow shop and job shop scheduling problem in logistics warehouses. Managing and scheduling works is currently often problem. There is no simple solution due to complexity of this problem. This problem must be resolved because of a lack efficiency of work with a higher load such as during the christmas holidays. This paper describes the methods used to solve this problem focusing mainly on the use of search algorithms, evolutionary algorithms, specifically grammar guided genetic programming. This paper describes the problem of job shop scheduling on a simple theoretical example. The implemented algorithm for solving this problem was subjected to tests inspired on data from real warehouse, as well as synthetically created tests with more jobs and a greater number of workers. Synthetic tests were generated randomly. All tests were therefore run several times and the results were averaged. In conclusion of this work are presented the results of the algorithm and the optimum parameter settings for different sizes of problems and requirements for the solution. Genetic algorithm has been extended to calculate fitness of individuals with regard to number of collisions, extended to use priority rules during run of evolution, and some parts of algorithm was parallelized.

KEYWORDS

Job shop, flow shop, logistics, genetic algorithm, evolutionary algorithm, grammar guided genetic programming, heuristics

POVODA, Lukáš *Rozvrhovanie úloh v logistických skladoch*: diplomová práca. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikácií, 2013. 64 s. Vedúci práce bol Ing. Jan Karásek

PREHLÁSENIE

Prehlasujem, že som svoju diplomovoú prácu na tému „Rozvrhovanie úloh v logistických skladoch“ vypracoval samostatne pod vedením vedúceho diplomovej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej diplomovej práce ďalej prehlasujem, že v súvislosti s vytvorením tejto diplomovej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/nebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona č. 121/2000 Sb., o právu autorském, o právoch súvisejúcich s právom autorským a o zmeně niektorých zákonov (autorský zákon), vo znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhé, hlavy VI. diel 4 Trestného zákoníka č. 40/2009 Sb.

Brno ....

(podpis autora)

POĎAKOVANIE

Rád by som poďakoval vedúcemu diplomovej práce pánovi Ing. Janovi Karáskovi za uvedenie do problematiky evolučných algoritmov, ako aj problematiky riadenia a rozvrhovania práce, rovnako za odborné vedenie, pravidelné konzultácie, trpezlivosť a podnetné návrhy k práci.

Brno ....

(podpis autora)

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic

http://www.six.feec.vutbr.cz

POĎAKOVANIE

Výzkum popsaný v této diplomové práci byl realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

Brno . . . . . .

. . . . . .

(podpis autora)

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OP Výzkum a vývoj
pro inovace

OBSAH

ZOZNAM OBRÁZKOV

ZOZNAM TABULIEK

ÚVOD

Táto práca sa venuje oblasti rozvrhovania a riadenia práce v logistických skladoch, ako aj optimalizácií prác založenej na využití evolučných algoritmov, konkrétne gramatikou riadeného genetického programovania.

Riadenie práce je v súčastnosti jedným z najobtiažnejších problémov. Riadenie a rozvrhovanie práce v logistických skladoch je v praxi riešené vedúcim prevádzky, teda skúsenou osobou, ktorá má detailné znalosti v tejto oblasti. Kvôli obrovskému počtu dát, ktoré musí pri návrhu zohľadniť môže byť táto činnosť veľmi časovo náročná. Problémy sa prejavujú najmä pri väčšom vyťažení skladu, ktoré nastáva napr. v období vianočných sviatkov.

Práca pojednáva o technikách, ktoré boli aplikované na riešenie problému riadenia práce, v mnohých prípadoch ide len o teoretické výskumy možností. Jedná sa o matematické metódy, využitie expertných systémov, neurónových sietí a prehľadávacích algoritmov. Práca informuje aj o prípadných nedostatkoch jednotlivých techník.

Naviazaním na už existujúcu štruktúru algoritmu genetického programovania bol vytvorený systém pre riadenie a rozvrhovanie práce zohľadňujúci počet kolízií nastávajúcich pri plnení jednotlivých úloh. Vďaka tomuto možno predpokladať menej problémov pri budúcej aplikácií v reálnom sklade, ako aj zníženie skutočného času určeného na prepravu tovaru z jedného miesta na druhé. V pôvodnom algoritme boli obsiahnuté tiež všetky genetické operátory, ako aj plne funkčné grafické rozhranie aplikácie.

Keďže bolo nutné zmerať úspešnosť implementovaného algoritmu, bola vytvorená sadu testov inšpirovaných reálnymi dátami. Implementácia testov, ako aj samotného algoritmu bola prevedená v programovacom jazyku Java. Účinnosť algoritmu bola meraná na základe maximálneho času potrebného na prepravu tovaru na určené miesta. Pre jednoduchú demonštráciu účinnosti algoritmu boli výsledné časy porovnané s časmi v prípadoch kedy prácu rozvrhoval človek. Algoritmus by mal dosahovať lepšie výsledky hlavne pri zložitejších prípadoch, kedy človek nie je schopný rozvrhnúť práce optimálne. Pre meranie účinnosti algoritmu pri väčších problémoch s vysokým počtom pracovníkov a úloh bol navrhnutý generátor syntetických testovacích prípadov. Vďaka tomuto algoritmu je možné určiť optimálne nastavenie genetických operátorov pre rôzne veľkosti problémov.

V rámci práce boli implementované prostriedky na zrýchlenie behu algoritmu, ako aj zrýchlenie iterácie algoritmu genetického programovania k optimálnym výsledkom. Zrýchlenie behu algoritmu, teda zníženie výpočtového času bolo dosiahnuté vhodnou paralelizáciou častí algoritmu. Zabezpečenie iterácie v priebehu evolúcie ako aj zrýchlenie prehľadávania priestoru riešení je zabezpečené použitím troch im-

plementovaných prioritných pravidiel. Pre otestovanie vplyvu takýchto krokov a zmien algoritmu je opäť nutné vytvoriť testy zamerané na tieto parametri.

Kapitola 1 obsahuje základnú definíciu problému, ako aj metódy určené pre riešenie elementárnych problémov spojených s rozvrhovaním práce. Aktuálny stav vedy a techniky v riešení problému riadenia práce je bližšie rozobraný v kap. 1.3. Ďalšia kapitola zhŕňa teoretické poznatky z oboru evolučných algoritmov, venuje sa bližšie genetickému programovaniu a aplikácií riešenia na konkrétny problém riadenia logistického skladu. Kapitola 3 opisuje implementované algoritmy, sadu reálnych testov a generátor syntetických testov. V kapitolách 3.1.4 a 3.2.2 sú obsiahnuté výsledky jednotlivých testov, porovnania a vyhodnotenie výstupných dát.

1 TEORETICKÉ ZÁKLADY PRÁCE

Rozvrhovanie a riadenie práce je v súčastnosti jedným z najobtiažnejších a najkomplexnejších problémov. Už po desaťročia je predmetom mnohých výskumov, avšak zatiaľ sa nedospelo k žiadnemu efektívnemu riešeniu. Hlavnou úlohou je nájsť postupnosť v ktorej sú jednotlivé úlohy pridelené medzi pracovníkov tak, aby boli splnené všetky úlohy v čo najkratšom čase a s čo najmenším omeškaním, pričom sa snažíme dosiahnuť rovnomerné využitie dostupných zdrojov a teda i maximálnu úroveň produkcie. [1] [2]

Problém riadenia a rozvrhovania práce patrí k problémom označovaných ako NP-kompletné, teda časť nedeterministických polynomiálnych problémov, ktoré sú v istom zmysle tie najobtiažnejšie z triedy NP. Tieto problémy sú obvykle riešené aproximáciou alebo pomocou heuristických metód. [3]

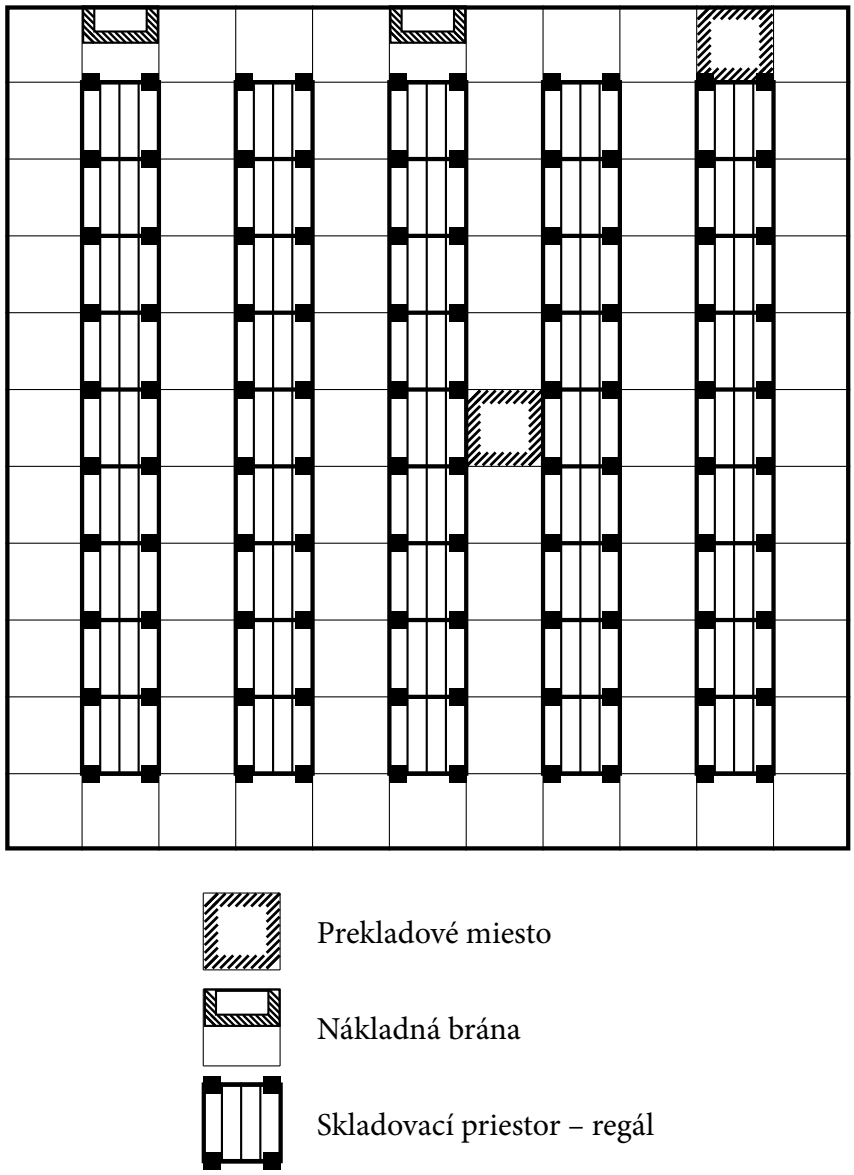
1.1 Skladové priestory

Problém rozvrhovania práce v sklade môžme jednoducho popísať na prípade, kedy máme uskladniť určitý počet paliet tovaru. K dispozícií máme daný počet zamestnancov, ktorým sú pridelené vozíky. Aby sme si problém uľahčili, rátame že každý zamestnanec má pridelený práve jeden vozík a nemá k dispozícií možnosť výmeny za iný. Každý typ vozíku má inú rýchlosť a môže dosiahnuť do rôznych poschodí skladovacieho priestoru. Opäť si problém zjednodušíme a budeme počítať iba s dvojrozmerným priestorom, teda že všetok tovar sa uloží na najnižšie poschodie regálu.

Skladový priestor sme teda schopní popísať matematicky ako maticu, pričom jednotlivé polia rozdelíme na dve základné skupiny – polia po ktorých sme schopný tovar prevážať, a polia do ktorých sa tovar uskladňuje. Toto rozdelenie sa rozšíri o ďalšie polia ktoré nám pomôžu rozšíriť možnosti riadenia práce, napr. polia pre preklad tovaru. Takéto pole môžeme definovať jednoznačne na určitej pozícií, napr. niekde na okrajoch skladu. Aby sme boli však presní, dočasné prekladové pole môže vzniknúť aj medzi regálami, kedy je nutné predať tovar inému pracovníkovi. Jednoduchý model skladu je znázornený na obr. 1.1, definované je aj umiestnenie nákladných brán a prekladových miest, legenda je na obrázku.

Jednotliví pracovníci sa môžu pohybovať po poliach medzi regálami, vkladať tovar do regálu môžu len z pravej alebo ľavej strany. Pracovníci majú k dispozícií rôzne druhy vozíkov, my si ich však rozdelíme na 3 triedy, ktoré budú jednoznačne definovať ich možnosti:

1. Ručný vozík – malý vozík, určený prevažne na prevoz paliet z jedného miesta na druhé a uskladňovanie do najnižšej pozície regálu, teda v úrovni podlahy.



Obr. 1.1: Jednoduchý model skladu.

2. Nízky vozík – môžeme predpokladať že bude mať najvyššiu rýchlosť spomedzi všetkých vozíkov, môže ukladať tovar do vyšších poschodí regálu, má však obmedzenú maximálnu výšku, do ktorej dosiahne.
3. Vysokozdvižný vozík – má nižšiu rýchlosť, no dosiahne aj do najvyššieho poschodia úložného priestoru.

1.1.1 Statický problém

Ak sa na riadenie práce pozrieme ako na statický problém, tak máme definovaný nemenný zoznam úloh, ktoré treba vykonať aby sme sa dostali do konečného

stavu – v našom prípade uskladnenie všetkých paliet s tovarom. Pokiaľ dokončenú úlohu zo zoznamu odoberieme, v konečnom stave ostane zoznam prázdny. Takáto situácia však v reálnom sklade nemôže nastať, pretože sa zoznam úloh stále mení (ak napr. niektorý z pracovníkov potrebuje prestávku, ak dorazí neočakávaná zásielka, ak zásielka mešká, nedorazila apod.), ako aj počet dostupných pracovníkov (napr. pri zmenovej práci), musíme teda riadenie práce riešiť ako dynamický problém.

1.1.2 Dynamický problém

Ak sa na riadenie práce v sklade pozrieme ako na dynamický problém, komplexnosť možného riešenia opäť vzrastie. Pokiaľ sa však rozdelí tento problém na viacero statických problémov, dostávame sa do riešiteľných medzí – vždy pracujeme so statickým zoznamom úloh, pričom ak nastane situácia kedy požadujeme zmenu tohto zoznamu, opäť rozdelíme jednotlivé pracovné úlohy medzi pracovníkov. Úlohy, ktoré sa už začali vykonávať nesmieme však meniť.

1.2 Priority úloh

K lepším riešeniam riadenie práce sa môžeme dopracovať za pomoci definovania priority jednotlivých úloh. V praxi možno využiť 6 najčastejších pravidiel pre určovanie priority úloh [3] [4]:

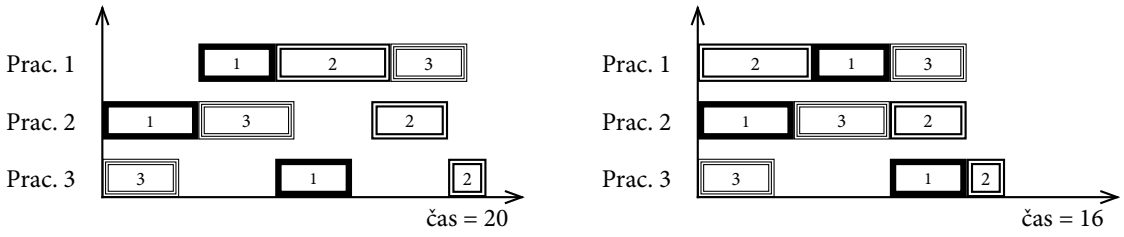
1. FCFS (First Come, First Served) – prvé sú vykonávané úlohy, ktoré dorazili skôr do systému. V praxi najčastejšie narazíme na toto pravidlo v službách, napr. v reštaurácií.
2. EDD (Earliest Due Date) – úlohy sú zoradené podľa požadovaného času dokončenia a vykonávané v tomto poradí. Toto pravidlo však nezaručuje, že všetky úlohy budú dokončené načas.
3. SPT (Shortest Processing Time) – úlohy s najkratším trvaním sú vykonávané ako prvé. Obtiažne dlhotrvajúce úlohy môžu však v systéme dlho čakať kým dôjde k ich spracovaniu.
4. CR (Critical Ratio) – najvyššiu prioritu majú úlohy s najvyššou kritickou hodnotou. Túto vypočítame ako podiel požadovaného času dokončenia a ostávajúceho času.
5. MINSOP (Minimum Slack Time Per Operation) – najvyššiu prioritu dostáva úloha s minimálnymi časovými rezervami jeho jednotlivých operácií. Časové rezervy sú definované ako rozdiel medzi najneskorším časom dokončenia operácie, ktorý nespôsobí meškanie úlohy, a najskorším možným časom dokončenia operácie.

6. COVERT (Cost Over Time) – omeškanie úloh je ocenené a prednostne sú spracovávané tie, ktoré majú najvyššiu cenu za oneskorenie.

V článku [5] boli predstavené ďalšie 3 pravidlá pre určovanie priority a zachádzanie s jednotlivými úlohami pri rozdeľovaní a riadení práce, ktoré využijeme najmä vtedy, ak máme niektoré úlohy závislé na iných.

1.2.1 **Partial Reordering** (PR) – čiastočné preusporiadanie

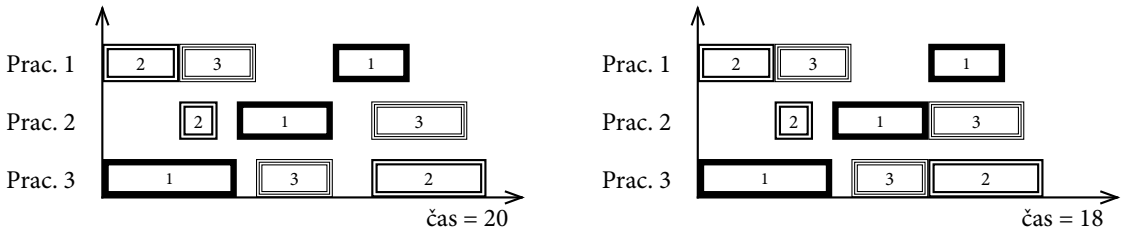
Ak posunieme úlohu z jej aktuálnej pozície na prvú pozíciu, môže nastať situácia kedy ostatné úlohy musia byť spracované neskôr, ako bolo pôvodne plánované. Toto však môže spôsobiť, že budeme schopní presunúť iné úlohy u ďalších pracovníkov na skoršie spracovanie, viď obr. 1.2.



Obr. 1.2: Príklad použitia pravidla PR.

1.2.2 **Gap Reduction** (GR) – redukcia medzier

V niektorých prípadoch je možné odstrániť medzery medzi jednotlivými úlohami veľmi jednoducho – pomocou posunutia začiatku nasledujúcej úlohy. Samozrejme sa musíme uistiť, že nenastane žiaden konflikt a že všetky podmienky pre vykonanie úlohy sú splnené.

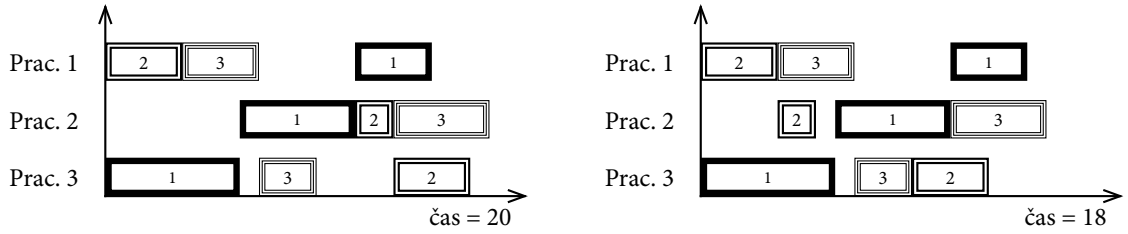


Obr. 1.3: Príklad použitia pravidla GR.

Toto pravidlo teda jednoznačne identifikuje medzery medzi jednotlivými úlohami daného pracovníka a zistí, ktoré úlohy môžu byť posunuté bez toho aby spôsobili predĺženie potrebného času na vykonanie všetkých úloh. Gantov diagram príkladu použitia tohto pravidla je na obr. 1.3.

1.2.3 **Restricted Swappings** (RS) – obmedzenie zmien

Toto pravidlo dovoľuje výmenu poradia dvoch po sebe nasledujúcich úloh, no iba vtedy ak to dovoľujú všetky podmienky úlohy. Pravidlo sa najskôr použije na najdlhšie trvajúcu úlohu, následne náhodne na všetky ostatné. Príklad použitia tohto pravidla v kombinácií s GR je na obr. 1.4



Obr. 1.4: Príklad použitia pravidla RS v kombinácií s GR.

1.3 Aktuálny stav vedy a techniky

V súčastnosti existuje mnoho štúdií a výskumov ako vyriešiť problém rozvrhovania a riadenia práce najmä vo výrobných odvetviach priemyslu, často s viacerými paralelnými výrobami rozdielnych produktov, kde každý produkt potrebuje veľa rozličných krokov a zariadení pre jeho úspešné dokončenie. Aj preto vznikajú rôzne smery, ktorými je možné sa vydať. V nasledujúcej časti sa pokúsim jednotlivé smery optimalizácie riadenia práce trochu priblížiť a spomenúť ich základné vlastnosti.

1.3.1 Matematické metódy

V [11] sa autori pokúšali vyriešiť tento problém pomocou dekompozície. Vytvorili dve vrstvy plánovania práce. Vrchná vrstva definovala minimálny možný počiatočný čas a maximálny možný konečný čas pre každú pracovnú úlohu. Spodná vrstva potom rozširovala tieto limitné časy podľa poradia jednotlivých úloh. Následne definovali funkciu, ktorá zahŕňala meškania jednotlivých úloh, výrobnú kapacitu a náklady.

Narazili však na problém pri pracovných úlohách, ktoré sú niečim obmedzené. Prítomnosť viacerých podmienok spôsobovalo často konflikty, ktoré bolo veľmi ťažké popísať prostredníctvom matematických vzťahov. [2]

Ďalším pokusom využitia matematických metód bolo popísať problém pomocou rozhodovacieho stromu. Každý uzol stromu predstavoval jedno rozhodnutie, ktoré viedlo k čiastočnému riešeniu. Pre každý uzol vznikajú ďalšie uzly. Listy tohto stromu sú výsledné riešenia problému. Problémom bolo často obmedzenie maximálnej veľkosti dátových typov, najmä u celých čísel. Tento problém bol vyriešený použitím Lagrangeovej relaxácie. [12] Toto riešenie je však veľmi výpočtovo náročné pre komplexnejšie problémy s vyšším počtom pracovných úloh. [13] [14]

1.3.2 Expertné systémy

Využitie umelej inteligencie pri riadení práce prináša určité výhody. Expertné systémy prinášajú možnosť generovania omnoho komplexnejšej heuristiky ako je tá, ktorú som spomínal v kap. 1.2. Výber vhodnej heuristiky môže byť založený na obrovskom množstve dát, ako je napr. súčasný zoznam úloh, očakávané úlohy, súčasný stav zdrojov apod. V neposlednom rade je nutné spomenúť, že tieto systémy zachytávajú komplexné vzťahy v jednej dátovej štruktúre, s ktorou však nie je jednoduché manipulovať. Plnenie dátami je časovo náročné, ako aj kontrola, údržba alebo dodatočná zmena či rozšírenie. [2]

Najznámejšími expertnými systémami sú ISIS, MPECS, OPTIMUM–AIV, OPIS a SONIA. Systém ISIS využíva 3 úrovne rozhodovania. Na prvej úrovni určí poradie úloh. Na druhej úrovni prebieha analýza kapacity jednotlivých zdrojov. Na tretej úrovni následne prebieha detailné rozdelenie úloh. [15]

Expertné systémy sú často obmedzené veľkosťou vlastnej znalostnej báze. Preto ich často nie je možné využiť pre riešenie obtiažnych problémov riadenia práce. Riešenie sa však našlo v použití viacerých menších expertných alebo znalostných systémov, ktoré spolupracujú pri riešení problému. Takéto systémy využívajú paralelné spracovanie na obrovskom počte procesorov. Jeden takýto proces v tomto systéme sa nazýva agent, ide o unikátny proces, ktorý spolupracuje s inými procesmi. Často sa delia na dva typy: na agentov, ktorí sú zodpovedný za plánovanie, a agentov, ktorí majú na starosti zdroje, či už ide o jeden zdroj alebo o určitú skupinu zdrojov. Táto druhá skupina agentov je často fyzicky pripojená na konkrétny zdroj a inicializuje prerozdelenie prác pokiaľ je to nutné. Nesmieme však zabudnúť zabezpečiť stabilitu systému. [2] [16]

Systémy na báze agentov vznikli na podnet väčších spoločností, najmä takých, ktoré produkujú obrovské množstvo produktov a zasiela ich v menších počtoch

kusov. Využitie následne našli najmä v oblasti riadenia práce, logistiky, dopravy, distribúcie energií, medicínskych testoch a projektovom riadení. [16]

1.3.3 Neurónové siete

K ďalším pokusom ako vyriešiť problém riadenia práce pribudla aj možnosť využitia neurónových sietí s algoritmom spätného šírenia chyby. [17] Neurónová sieť bola naučená pre riešenie rozdeľovania práce pre 3, 4, 5, 8, 10 a 20 rôznych pracovných strojov. Táto sieť bola otestovaná na numerických problémoch, pričom dosahovala výsledky lepšie o 25% až 50% ako použitie jednotlivých pravidiel z kap. 1.2.

Toto však nebola posledná aplikácia neurónovej siete na náš problém. Žiadna sa však nedostala do produkčného prostredia. Do neurónovej siete musí vždy vstupovať veľké množstvo premenných a tak sa stáva aplikácia výpočtovo neefektívna a v mnohých prípadoch je priam nemožné dopracovať sa k výslednému riešeniu. [2]

1.3.4 Prehľadávacie metódy

Prehľadávacie metódy dosahujú vynikajúce výsledky najmä v kombinácií s inou heuristikou. Tieto metódy sú založené na evaluácií výsledných riešení až do doby, kedy už nedochádza k žiadnemu zlepšeniu. Klasické prehľadávacie algoritmy majú tendenciu zastaviť sa na lokálnom optime. Tento problém riešia známe prehľadávacie techniky ako metóda TABU search, simulované žíhanie a genetické algoritmy. Vo všetkých sú použité mechanizmy, ktoré majú za úlohu predísť uviaznutiu prehľadávania v lokálnom optime. Takéto algoritmy sú často označované ako metaheuristika. [18]

Metóda TABU search bola založená na povolení prehľadávania riešení, ktoré z počiatku nevykazujú žiadne lepšie výsledky. Zacykleniu sa predchádza vďaka ukladaniu prehľadaných riešení do zoznamu. Tieto potenciálne riešenia sú označené ako TABU a teda sa naďalej už s nimi nebude pracovať. [19]

Simulované žíhanie vychádza z fyzikálneho procesu ochladzovania a rekryštalizácie kovov. Táto metóda je založená na pravdepodobnostnom výbere horších riešení. je teda schopná riešiť nelineárne problémy s chaotickými a zašumenými dátami. Tento algoritmus je univerzálny, nie je obmedzovaný žiadnymi vlastnosťami modelu. [20]

Spomínané metódy už boli úspešne implementované. Metóda TABU search bola použitá pre správu optickej telekomunikačnej siete, ako aj pre správu zaťaženia heterogénnych multiprocesorových výpočtových jednotiek. Metóda simulovaného žíhania bola zas úspešne aplikovaná na správu výrobných jednotiek a riadenie s tým spojené. Genetické algoritmy neboli doposiaľ na tento problém aplikované v produkčnom prostredí. Teoretické výskumy však preukázali, že genetické algoritmy sú

závislé na počiatočnej populácií. Pokiaľ sa jedná o úplne náhodne vygenerovanú populáciu, dosahuje sa omnoho nižšia efektivita ako pri simulovanom žíhaní, ale stále ide o efektívnejšie riešenie ako použitie samotnej heuristiky. Pokiaľ však počiatočnú populáciu vytvoríme za použitia heuristiky, genetické algoritmy môžu dosahovať omnoho lepšie výsledky ako simulované žíhanie. [2] Genetické algoritmy ako smer evolučných algoritmov si popíšeme bližšie v kap. 2.

2 NÁVRH ALGORITMU

Evolučné algoritmy sú pomerne rozsiahlou triedou pravdepodobnostných populačných prehľadávacích algoritmov, vhodných pre riešenie optimalizačných úloh. Využívajú sa najmä v prípadoch kedy nemusíme nájsť optimálne riešenie daného problému ale stačí nám aj niektoré suboptimálne riešenie, ktoré je svojou kvalitou blízke tomu optimálnemu. [6]

V minulosti bolo mnoho pokusov využiť niektorú z teórií evolúcie (či už ide o evolúciu podľa Ch. Darvina alebo teóriu génovej dedičnosti podľa G. Mendela) ako základ pre tvorbu algoritmov. Evolučné algoritmy ako také majú jasne danú štruktúru, no jednotlivé prístupy riešenia sa môžu mierne líšiť. Časom boli definované viaceré smery, ktorými sa prístup riešenia problému uberal. Mnoho z nich vznikalo nezávisle na sebe, niektoré vznikli pri prispôsobovaní na konkrétnu riešenú úlohu [6]:

1. Evolučná stratégia (1965)
2. Evolučné programovanie (1966)
3. Genetický algoritmus (1975)
4. Genetické programovanie (1985)
5. Šľachtiteľský algoritmus (označovaný ako BGA, 1993)
6. Diferenciálna evolúcia (1997)
7. Eugenická evolúcia (1998)
8. Samoorganizujúci sa migračný algoritmus (označovaný ako SOMA, 1999)
9. Harmonické prehľadávanie (2001)

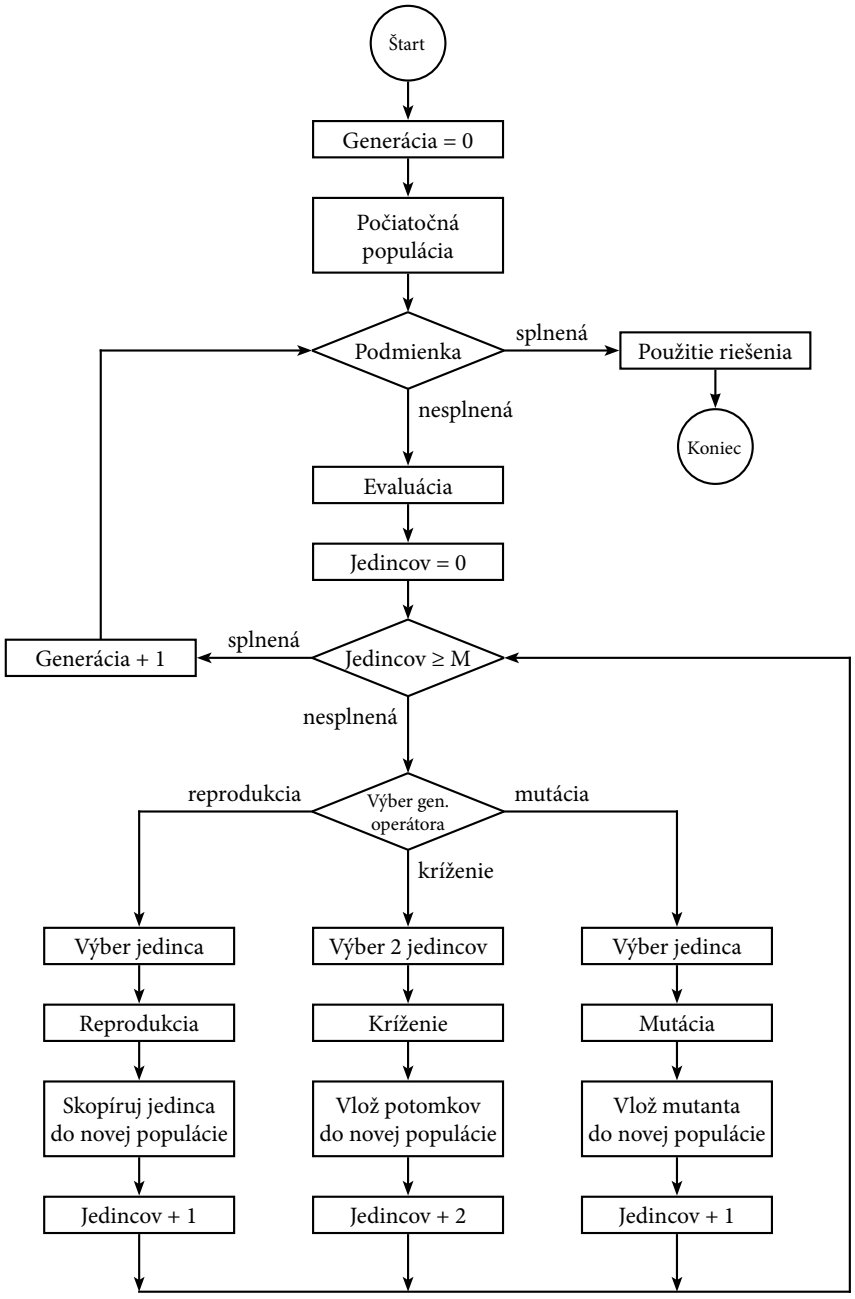
Evolučné algoritmy, konkrétne smer genetických algoritmov, sú v súčasnosti využívané v mnohých odvetviach – napr. v bioinformatike, v ekonomike, v ekológií pre popis optimálneho životného prostredia určitého druhu, v rozvrhovaní a plánovaní, pri kontrole kvality, vo finančníctve atď. [7] [8] [9]

Keďže problém riadenia práce v sklade nie je možné matematicky jednoznačne určiť, pri riešení tohto problému využijeme evolučné algoritmy, konkrétne genetické programovanie. Základná štruktúra takéhoto algoritmu[1] je znázornená na obr. 2.1. Bližšie informácie o aplikácií genetického algoritmu na tento problém sú popísané v kap. 2.2.

2.1 Evolučné výpočtové techniky

Terminológia evolučných algoritmov sa inšpiruje v biológií – jedno potenciálne riešenie nášho problému je často označované ako *jedinec*, jeho vnútorné premenné sú

[1] Zdroj: http://www.geneticprogramming.com/Tutorial/



Obr. 2.1: Štruktúra algoritmu genetického programovania.

označované ako *gény*, ktorých hodnoty sú ďalej nazývané ako *alely*. Na výpočet vhodnosti zvoleného riešenia sa používa tzv. *fitness funkcia*, ktorá vracia reálnu hodnotu kvality riešenia úlohy. Skupina jedincov v jednom cykle algoritmu nazývame *populácia* a poradové číslo cyklu určuje *generáciu*.

Na začiatku činnosti algoritmu musíme vytvoriť počiatočnú populáciu. Vygenerujeme teda určitý počet jedincov s náhodnými génmi tak, aby pravdepodobnosť

vygenerovania každého potencionálneho riešenia bola rovnaká. Následne určíme ich vhodnosť. V jednotlivých cykloch algoritmu vyberáme z populácie vždy najvhodnejších jedincov s ktorými prevádzame genetické operácie – kríženie jedincov a náhodné mutácie ich génov.

Na konci každého cyklu sa z pôvodnej populácie a nových vzniknutých jedincov vytvorí nová populácia. Pôvodná populácia môže však byť úplne nahradená populáciou nových jedincov. Cyklus je prerušený po splnení stanovených podmienok, napr. pri dosiahnutí maximálneho počtu iterácií alebo ak v jednotlivých generáciách nenastávajú dostatočne veľké zmeny v kvalite riešenia. [7]

2.2 Kroky inicializácie

Pred samotnou tvorbou algoritmu genetického programovania je nutné prejsť 5 krokov inicializácie. Týmto si jasne definujeme základné črty a vlastnosti algoritmu, a teda i nášho problému. Viac informácií k inicializácií môžete nájsť v [21] a [22]. V nasledujúcich sekciách sa zameriam na náš konkrétny problém.

2.2.1 Definícia terminálov

Terminál predstavuje určitý vstup do algoritmu. V tomto prípade možno terminálami označiť jeden nedeliteľný úkon v sklade, zdroje (teda vozíky pre prevoz tovaru a pracovníkov) a samotný tovar. Implementácia algoritmu bola prevedená v programovacom jazyku Java, preto sa pokúsim priblížiť konkrétne i jednotlivé triedy terminálov.

Jeden atomický úkon v sklade si nazveme ako `Job` a definujeme ho ako abstraktnú triedu. Následne si môžeme definovať tieto základne triedy úkonov pre prácu s tovarom:

- `JobLoad` – naloženie tovaru na vozík,
- `JobMove` – presun z jedného miesta na druhé (zahŕňa i prevoz tovaru),
- `JobUnload` – vyloženie tovaru do regálu alebo na prekladové miesto.

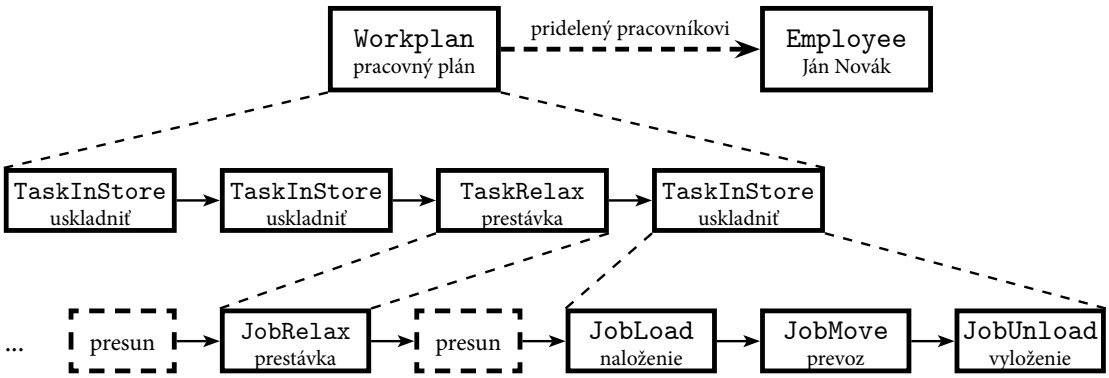
V sklade však vznikajú i úkony, ktoré sa nevzťahujú priamo k manipulácií s tovarom. Medzi ne patrí napríklad `JobRelax`, ktorý predstavuje prestávku jedného konkrétneho pracovníka skladu. Pre zníženie počtu kolízií môžeme ďalej definovať napr. `JobWait` pre čakanie na aktuálnej pozícií apod. Každému úkonu musíme priradiť čas trvania, vždy sa bude jednať o konštantnú hodnotu, okrem úkonu `JobMove`, kedy musíme čas trvania vypočítať na základe vzdialenosti a prípadne i rýchlosti použitého vozíka.

K ďalším terminálom sa prepracujeme, pokiaľ si rozoberieme naše zdroje v sklade. Medzi zdroje[2] spadá pracovník skladu `Employee` a vybavenie v sklade `Equipment`. Vybavením skladu sa rozumejú vozíky pre prevoz tovaru (definované v kap. 1.1): `ForkLiftHand`, `ForkLiftLow` a `ForkLiftHigh`. Týmto triedam môžeme ďalej definovať rôzne parametre, u vozíkov napr. veľkosť a maximálnu rýchlosť, u pracovníkov napr. výkonnosť, kompetencie apod. Tieto parametre môžeme uplatniť pri počítaní času trvania jednej úlohy a pri počítaní kolízií.

2.2.2 Definícia funkcií

Ďalším krokom je definovanie vzťahov, respektíve gramatiky. Gramatika nám udáva syntaktické obmedzenia a jasne definuje zoznam povolených výrazov v jazyku pochopiteľnom pre stroje. Použitím gramatiky pri genetickom programovaní je možné o poznanie zmenšiť veľkosť prehľadávacieho priestoru. [23] [24] [25]

Pri riadení práce v skladových priestorov možno popísať prácu jedného pracovníka jeho pracovným plánom `Workplan`, v ktorom má pracovník presne určené aké úlohy má vykonať a v akom poradí. Jedna úloha všeobecného charakteru (`Task`) bude pozostávať z jedného alebo viacerých úkonov (`Job`), pričom pre jednotlivé úlohy si presne definujeme v akom poradí musia nasledovať. Preto si jednoznačne definujeme jednotlivé úlohy, ktoré je možné v sklade riešiť. Príklad takejto štruktúry je znázornený na obr. 2.2.



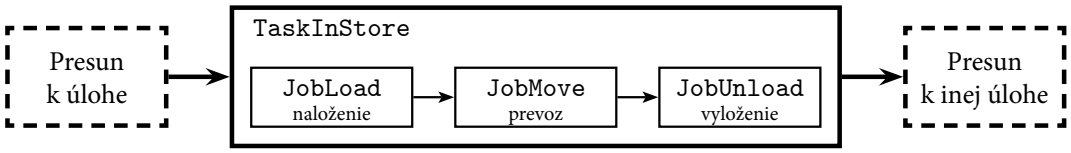
Obr. 2.2: Príklad štruktúry pracovného plánu.

[2]Zdroje si môžeme opäť označiť ako abstraktnú triedu `Resource`, to však v danej fáze algoritmu nie je dôležité, keďže sa nejedná o nedeliteľný terminál.

Uskladnenie tovaru

Úlohu uskladnenia tovaru si označíme ako `TaskInStore`. Vo väčšine prípadov nás tovar ktorý chceme uskladniť čaká na nákladnej bráne. Pre zjednodušenie zatiaľ pracujeme s jednotkami paliet, teda každý tovar – každý objekt `Commodity` predstavuje jednu paletu tovaru. Úlohu uskladnenia však môžeme v niektorých prípadoch rozdeliť na dve iné úlohy, kedy uskladnenie prebehne len na dočasné prekladové miesto. Toto nastáva v prípadoch, kedy je nutná kooperácia viacerých druhov vozíkov a teda aj viacerých pracovníkov. V tomto prípade teda prevezieme tovar z brány na prekladové miesto, kde následne iný pracovník naloží tovar, prevezie ho na požadované miesto a uskladní ho.

V oboch prípadoch sa úloha skladá z rovnakých úkonov – naloženie tovaru na vozík, prevoz tovaru na určené miesto a vyloženie tovaru. Aby sme si prácu zbytočne nekomplikovali, v jednotlivých úlohách nebudeme definovať prvotný nutný presun k tovaru. Tento bude prebiehať automaticky medzi dvomi úlohami. Výsledný celý proces uskladnenia je uvedený na obr. 2.3.



Obr. 2.3: Úloha uskladnenia tovaru.

Prestávka

Ďalšou úlohou ktorú musíme zapracovať do modelu je úloha prestávky pre konkrétneho pracovníka. Táto úloha musí byť naviazaná na konkrétneho pracovníka a teda s týmto vzťahom musíme v algoritme počítať. Samotná úloha `RelaxTask` bude pozostávať iba z jedného úkonu, a síce z `JobRelax`. Tento úkon bude v sebe obsahovať dĺžku požadovanej prestávky, aby sme vedeli s akým časom v algoritme počítať. Proces prestávky je znázornený na obr. 2.4.

Obr. 2.4: Úloha prestávky pracovníka.

2.2.3 Určenie vhodnosti

Ďalším krokom prípravy je definovanie funkcie merania vhodnosti jednotlivých jedincov populácie. Vo väčšine prípadov je vhodnosť definovaná chybou výsledného jedinca genetického algoritmu. Čím je bližšie táto hodnota k nule, tým kvalitnejšie riešenie máme. Nie je možné jednoznačne určiť skutočné optimálne riešenie problému rozvrhovania práce. Hlavnou úlohou však je dosiahnuť čo najnižší čas potrebný na úspešne vykonanie všetkých úloh. Výslednú vhodnosť je možné získať ako maximum konečných časov spomedzi všetkých pracovných plánov, teda $\max(C_1 \dots C_n)$. Toto maximum predstavuje prevrátenú hodnotu vhodnosti riešenia, teda najlepším riešením bude jedinec s čo najnižšou hodnotou maxima konečného času.

$$\frac{1}{fitness} = w_1 \cdot \max(C_1 \dots C_n)$$

Aby sa predišlo zbytočným komplikáciám pri nasadzovaní v praxi, počíta sa i s kolíziami vozíkov. Vo výslednej fitness funkcií bude teda súčet maxima konečného času a počtu kolízií, ktoré nastali behom všetkých úloh v celom sklade. Jednotlivým zložkám sa pridelia váhy, vhodnosť bude teda:

$$\frac{1}{fitness} = w_1 \cdot \max(C_1 \dots C_n) + w_2 \cdot collisions$$

Takto vytvorená fitness funkcia by mala postačovať. K lepším výsledkom a rovnomernejšiemu rozdeleniu úloh sa však prepracujeme za použitia ďalšieho parametru, priemernej dĺžky trvania jednej úlohy, teda $(\sum_{i=1}^{n} t_{task_i})/n$. Výsledná fitness funkcia je teda nasledovná:

$$\frac{1}{fitness} = w_1 \cdot \max(C_1 \dots C_n) + w_2 \cdot collisions + w_3 \cdot \frac{\sum_{i=1}^{n} t_{task_i}}{n}$$

2.2.4 Parametre kontroly behu

Vďaka gramatike definovanej v kap. 2.2.2 je zabezpečená istá heuristika pre začiatok genetického algoritmu. Ako som popisoval v kap. 1.3.4, na začiatku algoritmu musíme vygenerovať prvotnú populáciu za použitia heuristiky. Toto máme splnené a môžeme sa teda pustiť do definície samotných genetických operátorov. Operátory budú pracovať s dátovou štruktúrou, ktorú som načrtol v kap.2.2.2. Ako vidíme na obr. 2.2, dátová štruktúra je rozdelená do niekoľkých vrstiev – vrstva pracovných plánov, vrstva úloh a vrstva úkonov. Pri tvorbe genetických operátorov nás budú zaujímať len dve vrchné vrstvy. Posledná vrstva definuje už spomínanú gramatiku a teda zásah do nej by efektivite algoritmu len uškodil.

V priebeh genetického algoritmu ovplyvňujú genetické operátory. Pri riadení a rozvrhovaní práce je možné definovať na základe terminálov a celkovej štruktúry problému tieto základné operátory:

Výmena poradia úloh

Prvým a najjednoduchším operátorom ktorý si definujeme a rozoberieme je výmena poradia úloh v jednom pracovnom pláne. Pokiaľ dôjde k výmene dvoch náhodných úloh v pracovnom pláne a zároveň v tom nebráni žiadne obmedzenie, môžeme sa prepracovať k lepším výsledkom. Dobrým príkladom je závislosť začiatočnej pozície vykonávania úlohy od aktuálnej polohy pracovníka. Pokiaľ bude zvolená prvá úloha taká, ktorej tovar sa nachádza bližšie k pracovníkovi a následne vykonávame inú úlohu, môžeme dostať lepší výsledok vzhľadom na čas nutný pre presun pracovníka k tovaru, prípadne sa môžeme vyhnúť kolízií s iným pracovníkom, ktorá by opäť zhoršila výslednú hodnotu fitness funkcie.

Výmena pracovných plánov

Podobné situácie môžu nastať aj pri jednoduchej výmene pracovných plánov medzi dvomi pracovníkmi. V tomto prípade však musíme skontrolovať, či je možné skutočne vymeniť celé pracovné plány medzi pracovníkmi – niektoré úlohy môžu mať obmedzenia, ktoré tejto výmene môžu zabrániť. Dobrým príkladom je výmena plánov pokiaľ sa v niektorom nachádza úloha prestávky. Vtedy nesmieme priradiť túto prestávku pracovníkovi, ktorý si o ňu nepožiadal. Výmena plánov môže nastať, no prestávku musíme z pracovného plánu presunúť do druhého, ideálne na rovnakú pozíciu ako bola v pôvodnom pracovnom pláne pracovníka.

Na podobné obmedzenia môžeme naraziť aj pri iných úlohách. V týchto prípadoch budeme teda úlohy vždy nechávať staticky priradené jednému pracovníkovi, teda tomu, ktorý má dostatočnú kvalifikáciu alebo je úloha určená priamo jemu.

Výmena úloh medzi pracovnými plánmi

Aby sme zabezpečili určitú voľnosť v rozhadzovaní úloh, musíme zabezpečiť aj výmenu úloh medzi pracovnými plánmi. Tento operátor zväčší prehľadávací priestor a môžeme sa tak dostať k omnoho lepším výsledkom. Ide o výmenu dvoch rôznych úloh z dvoch rôznych pracovných plánov, teda aj priradenie úlohy inému pracovníkovi. Opäť sa teda stretávame s problémom, kedy je nutné skontrolovať, či pracovník môže vykonávať danú úlohu. Pokiaľ nie, táto mutácia neprebehne a noví vzniknutí jedinci do populácie nebudú zahrnutí.

Rozdelenie úlohy

Beh algoritmu genetického programovania môžeme ďalej rozšíriť o mutáciu rozdeľujúcu úlohy na dve iné úlohy – podúlohy, pričom každá bude vykonávaná iným

pracovníkom. Táto mutácia pomáha najmä v prípadoch, kedy potrebujeme uskladniť tovar do vyšších poschodí. Vtedy totiž potrebujeme využiť vysokozdvižný vozík, ktorý je schopný do týchto poschodí tovar vyložiť. Rýchlosť takéhoto vozíku je však príliš nízka na to, aby sme s ním prešli celý sklad a tovar previezli a uskladnili. Vtedy sa hodí rozdelenie úloh, teda malým a rýchlim vozíkom sa dovezie tovar bližšie a následne sa uskladní pomocou vysokozdvižného vozíku.

Rozdelením vzniknú dve kratšie trvajúce úlohy, pričom časť úlohy zostane pridelená v rovnakom pracovnom pláne a druhá časť úlohy sa vloží do pracovného plánu iného pracovníka. V jednom pracovnom pláne nastanú posuny a úlohy sa začnú vykonávať skôr, keďže pôvodná úloha bola zamenená za kratšie trvajúcu. V druhom pláne pribudne úloha naviac. Opäť sa otvára mnoho ciest akými sa ďalší vývoj nasledujúcich populácií genetického algoritmu môže uberať. Úlohám však nepriradíme žiadne obmedzenie, žiadnu podmienku ktorá by vynútila poradie spracovania úloh. Rozhodnutie, či čakanie pracovníka na určenej pozícii je alebo nie je výhodné ponecháme na samotnom algoritme.

Zmena cesty

Posledným genetickým operátorom, ktorý si definujeme je zmena cesty. Tento operátor zabezpečí zníženie počtu kolízií. Pokiaľ sa pracovník s vozíkom musí presunúť z jednej strany skladu na opačnú, presun inou uličkou môže zabrániť kolízií s iným pracovníkom.

2.2.5 Podmienka ukončenia

Posledným krokom inicializácie je definovanie podmienky, ktorá ukončí algoritmus. Ako som spomínal v kap. 2.2.3, nepoznáme skutočné optimálne riešenie, preto je podmienka ukončenia definovaná ako maximálny počet generácií. Po dosiahnutí maxima vyberieme najlepšieho jedinca a použijeme toto riešenie.

3 IMPLEMENTÁCIA ALGORITMU

Implementácia algoritmu bola prevedená v programovacom jazyku Java, ktorý sa vyznačuje svojím objektovým prístupom a prenositeľnosťou zdrojových kódov na veľké množstvo platforiem. K výsledkom sa budem snažiť priložiť aj informácie o náročnosti behu algoritmu, ako aj jeho pamäťové nároky.

3.1 Sada reálnych testovacích príkladov

Pre testovanie algoritmu bolo navrhnutých 60 testových príkladov, aby bola možnosť porovnania úspešnosti algoritmu voči riešeniu navrhnutého človekom. Testy boli založené na extrahovaných dátach z reálnej prevádzky skladu nemenovanej logistickej spoločnosti z predvianočného obdobia roku 2011. Náročnosť jednotlivých testov bola rozdelená do piatich úrovní:

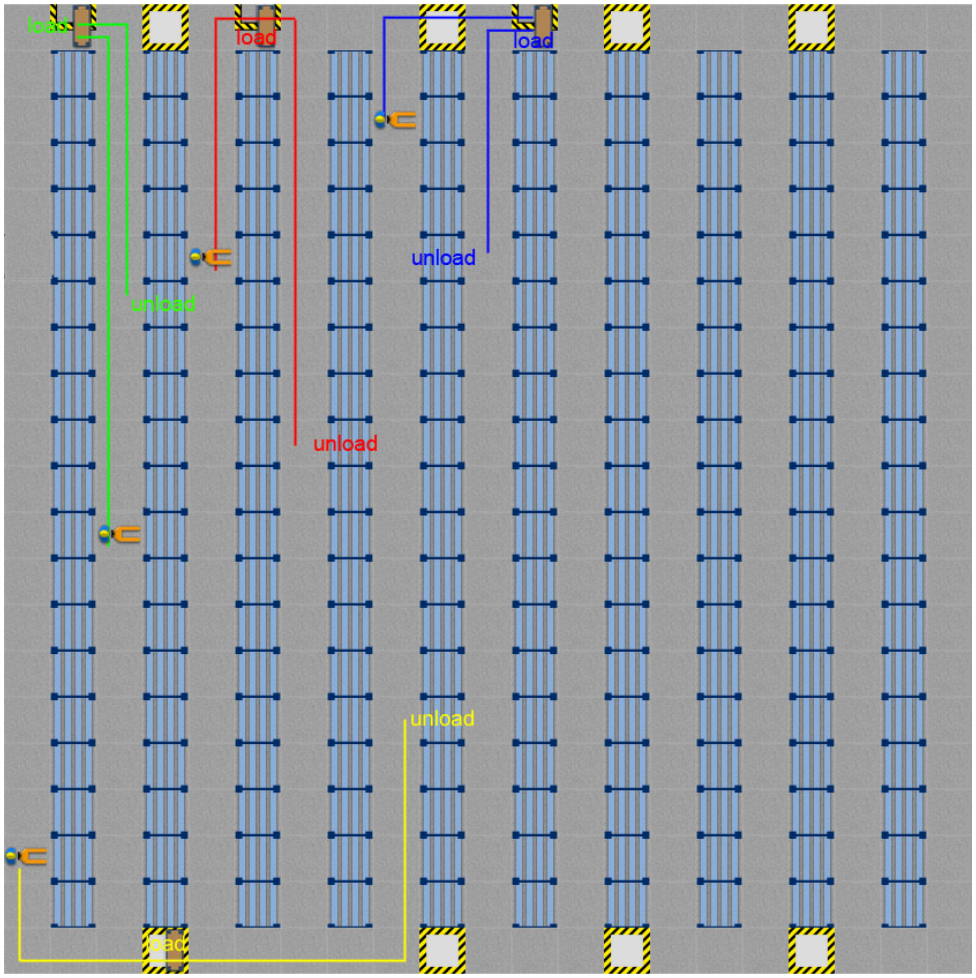
1. Testy 1 až 10 sú jednoduché prípady, kedy sa v sklade nachádzajú len ručné vozíky, pričom každému pracovníkovi bola priradená jedna jednoduchá úloha uskladnenia tovaru.
2. Testy 11 až 20 sú podobné prípady ako z predošlej skupiny, boli však rozšírené o ďalšie typy vozíkov. Každý pracovník má opäť definovanú jednu jednoduchú úlohu.
3. Testy 21 až 30 sú stredne zložité prípady, kedy pracovníci majú vykonať väčší počet úloh. Úlohy sú jednoduché, ide o nové scenáre alebo o rozšírenia scenárov z predošlých testov.
4. Testy 31 až 40 sú stredne zložité prípady, najmä rozšírené testy z predošlej skupiny. V týchto testoch bolo rozvrhnutie úloh navrhnuté tak, aby nedochádzalo ku kolíziám.
5. Testy 41 až 60 sú zložité prípady s vyšším počtom úloh, kedy sa často vyžaduje spolupráca pracovníkov. Opäť sú úlohy rozvrhnuté tak, aby dochádzalo k čo najnižšiemu počtu kolízií.

3.1.1 Popis testu

Na obr. 3.1 sa nachádza názorná ukážka testu č. 3, ide o jednoduchý prípad kedy máme dostupných štyroch pracovníkov, ktorí majú uskladniť štyri palety na presne definované miesta v regáloch. Tento test má za úlohu zistiť, či algoritmus dokáže nájsť vhodné rozdelenie pracovných úloh tak, aby každý pracovník uskladňoval paletu, ktorá je umiestnená najbližšie k jeho aktuálnej pozícií. Všetky vozíky sú v tomto príklade typu `ForkLiftHand`, teda ručné vozíky s nízkou rýchlosťou.

Tab. 3.1: Popis parametrov testového prípadu č. 3.

Počet pracovníkov:	4
Počet vozíkov:	4x ručný vozík
Počet úloh:	4
Problém:	Uskladnenie 4 paliet ručnými vozíkmi. Poloha pracovníkov a paliet bola volená tak, aby šlo tento problém vyriešiť jednoducho a jednoznačne.



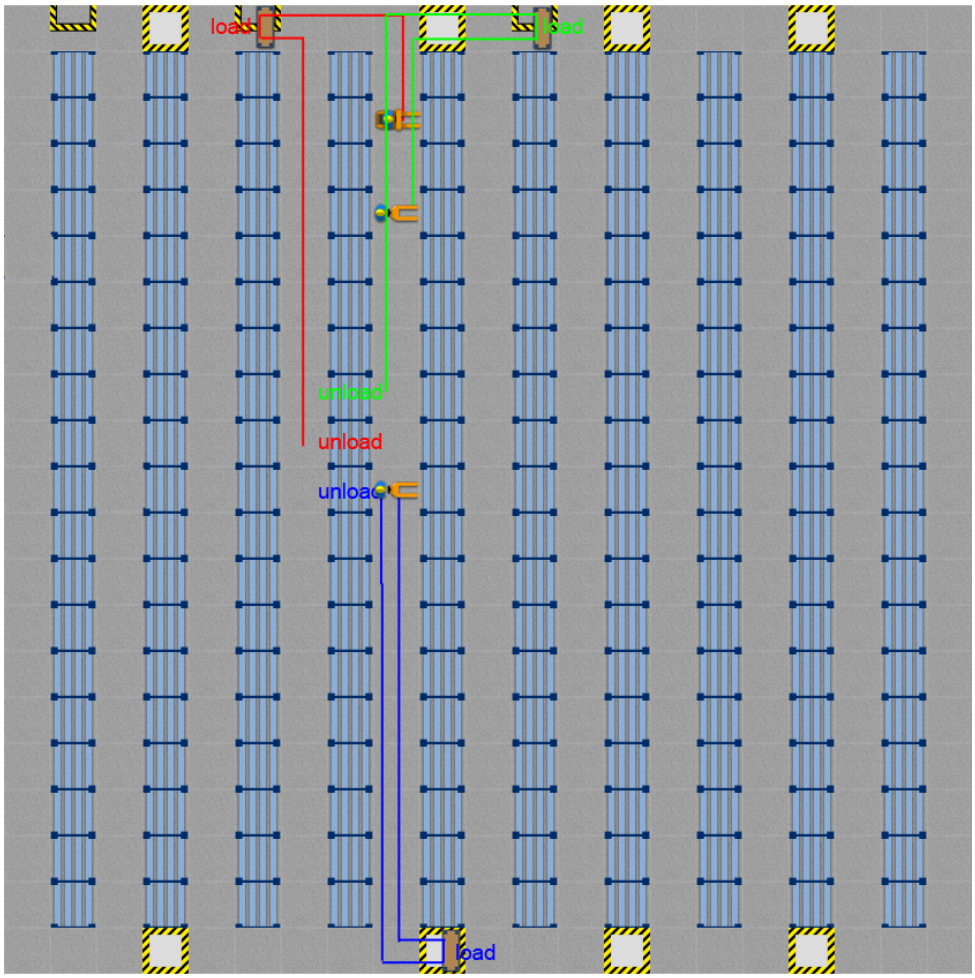
Obr. 3.1: Testový prípad č. 3.

Cesta každého pracovníka bola pri návrhu kvôli názornosti označená vždy inou farbou. U pracovníka so žltou farbou ide o uskladnenie palety z prekladového miesta do regálu, ostatné palety čakajú na bráne. To však vôbec nemení logiku zadania pracovných úloh a je teda možné pokladať tento test za jednoduchý, skladajúci sa zo štyroch jednoduchých úloh.

Na obr. 3.2 je znázornený testový príklad z druhej kategórie zložitosti. V tomto prípade máme k dispozícií rôzne typy vozíkov. Rýchly vozík by mal vybehnúť z uličky

Tab. 3.2: Popis parametrov testového prípadu č. 16.

Počet pracovníkov:	3
Počet vozíkov:	2x ručný vozík, 1x vysokozdvižný vozík
Počet úloh:	3
Problém:	Volenie vhodného smeru na základe pôvodnej polohy pracovníkov.



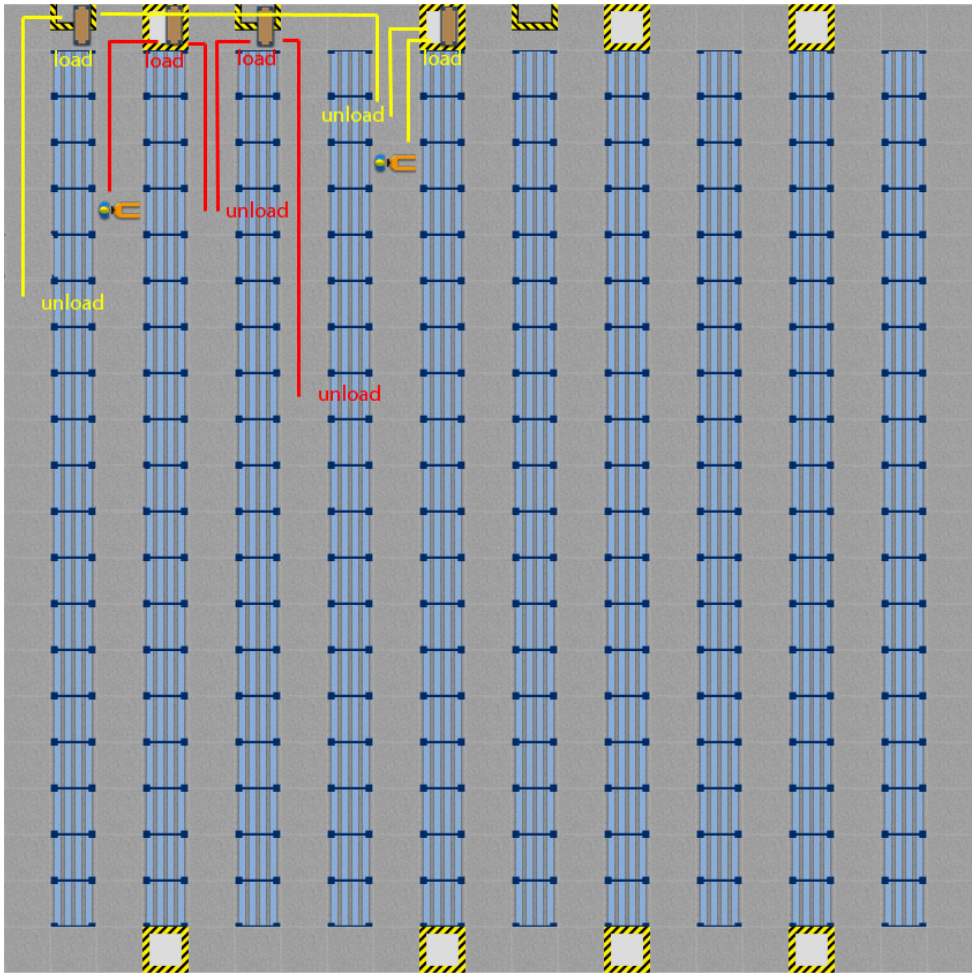
Obr. 3.2: Testový prípad č. 16.

a zvoliť si jednu paletu k uskladneniu. Musí si však zvoliť smer, v ktorom nebude obmedzovať ostatných pracovníkov a dôjde k najrýchlejšiemu uskladneniu. Najdlhšou úlohou bude v tomto teste uskladnenie palety z prekladového miesta, teda pracovník s označením modrej cesty, no v žiadnom prípade by sa nemal vybrať k paletám ktoré sa nachádzajú na nákladných bránach.

Test z ďalšej kategórie je znázornený na obr. 3.3. V tomto teste sú v sklade dostupní dvaja pracovníci, pričom každý z nich má k dispozícií len ručný vozík, teda by s ním nemali absolvovať veľké vzdialenosti. Rozvrhnutie ciest bolo pod-

Tab. 3.3: Popis parametrov testového prípadu č. 22.

Počet pracovníkov:	2
Počet vozíkov:	2x ručný vozík
Počet úloh:	4
Problém:	Rovnomerné rozdelenie úloh medzi pracovníkov.



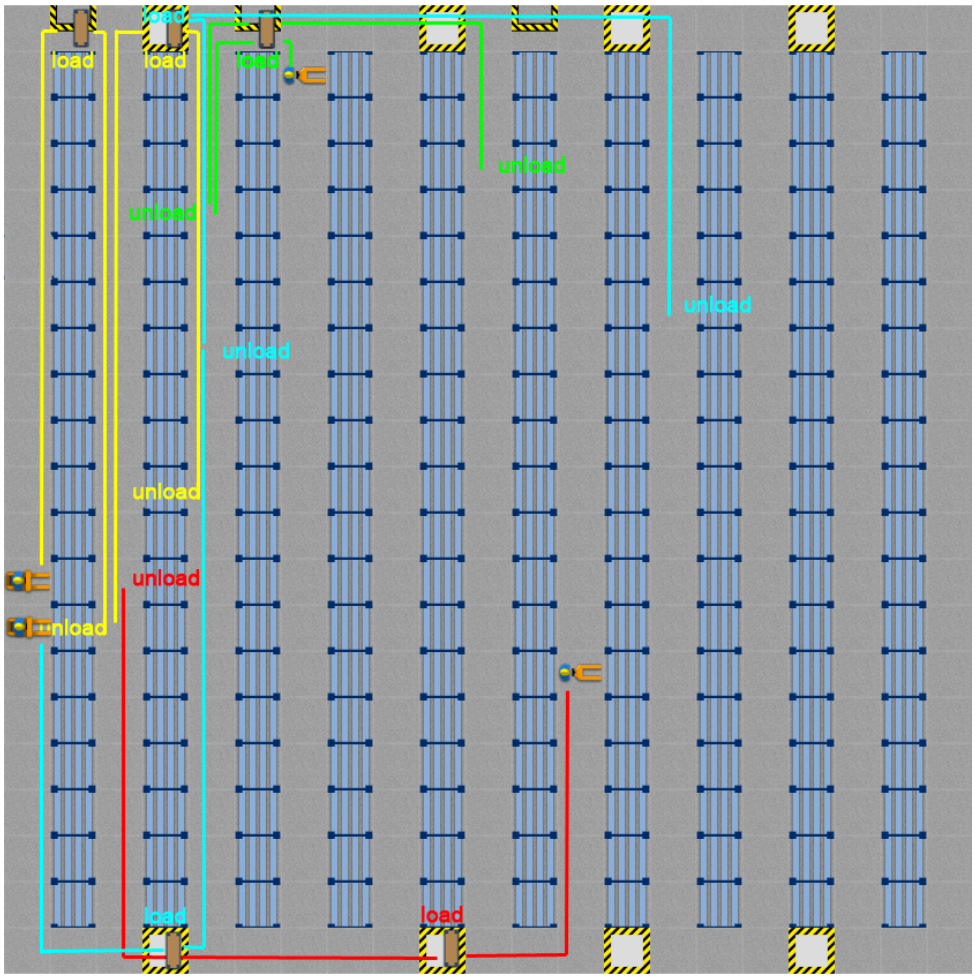
Obr. 3.3: Testový prípad č. 22.

mienené zhruba rovnakým časom pre uskladnenie 2 paliet jedným z pracovníkov. Teda aby pracovníci boli zhruba rovnako zaťažení. Pokiaľ jeden musí zachádzať viac krát do uličky medzi regálami, druhý pracovník spracuje paletu na vzdialenejšej nákladnej bráne.

Jedným z testov zameraných na minimalizáciu kolízií je napr. test č. 32, je znázornený na obr. 3.4. S vyšším počtom úloh nastáva väčšia pravdepodobnosť kolízií. Rozvrhnutie práce v tomto prípade bolo navrhnuté tak, aby rýchle vozíky vyriešili prevoz palety na väčšie vzdialenosti a stihli to pred tým, ako po danej trase

Tab. 3.4: Popis parametrov testového prípadu č. 32.

Počet pracovníkov:	4
Počet vozíkov:	2x ručný vozík, 2x nízky rýchly vozík
Počet úloh:	7
Problém:	Rovnomerné rozdelenie úloh medzi pracovníkov, minimalizácia kolízií.



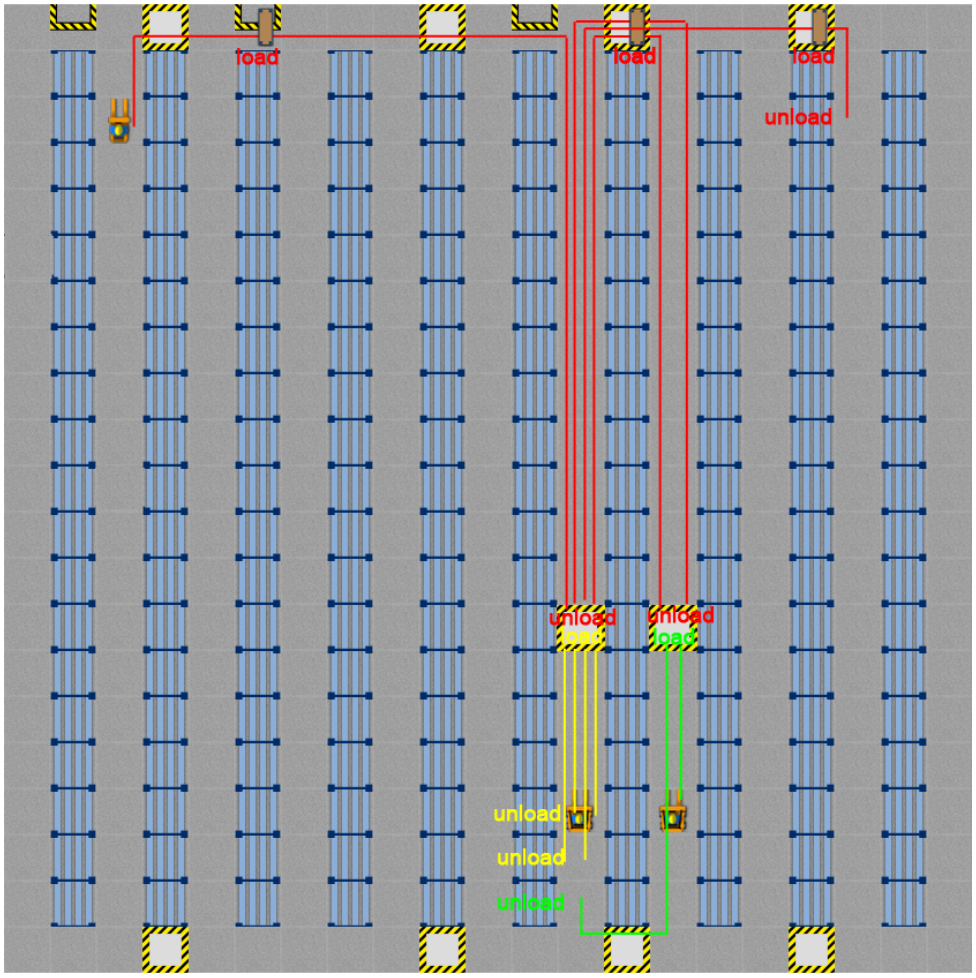
Obr. 3.4: Testový prípad č. 32.

pôjde pracovník s ručným vozíkom s nižšou rýchlosťou. Všetky úlohy by mali byť teda splnené s minimálnym možným počtom kolízií, pričom voľba pracovníka, ktorý úlohu spracuje bola takisto závislá na prípadnom vzniku kolízií.

Zložitejší príklad testu je uvedený na obr. 3.5. Tu sa vyžaduje spolupráca jedného rýchleho vozíku, ktorý zabezpečí prevoz jednotlivých paliet na prekladové miesta, bližšie k pracovníkom s vysokozdvižnými vozíkmi, ktoré nie sú schopné dosiahnuť tak vysokú rýchlosť. Presun takéhoto vozíku pozdĺž celého skladu by bol časovo náročný, je teda jednoduchšie použiť jeden rýchly vozík pre prevoz troch paliet bližšie k nim.

Tab. 3.5: Popis parametrov testového prípadu č. 56.

Počet pracovníkov:	4
Počet vozíkov:	1x nízky rýchly vozík, 2x vysokozdvižný vozík
Počet úloh:	4
Problém:	Spolupráca rôznych typov vozíkov, čakanie vozíkov.



Obr. 3.5: Testový prípad č. 56.

Počet úloh a počet pracovníkov je v tomto konkrétnom prípade rovnaký, no keďže sa vyžaduje spolupráca pracovníkov, ide o prípad z poslednej spomínanej kategórie – zložité prípady. Tento prípad bol zvolený kvôli názornosti, v tejto kategórií sa nachádzajú aj prípady, ktoré nie je jednoduché graficky znázorniť, v niektorých prípadoch až nemožné bez použitia viacerých vrstiev alebo animácie.

3.1.2 Algoritmus testu

Pri tvorbe všetkých testov bola dodržaná logická štruktúra, ktorá bola zavedená pri plnení hlavného programu testovacími dátami. Na začiatku je nutné definovať databázové modely pre objektový prístup k dátam v databáze. Takisto je nutné vymazať obsah všetkých databázových tabuliek, aby nedošlo k použitiu dát z predošlých testov:

```
DaoBase daoEmployee = new DaoBase(Employee.class);
DaoBase daoEquipment = new DaoBase(Equipment.class);
DaoBase daoCommodity = new DaoBase(Commodity.class);
DaoBase daoTaskInStore = new DaoBase(TaskInStore.class);

daoEmployee.deleteTables();
```

Takto vyčistená databáza sa naplní testovacími dátami konkrétneho prípadu. Najskôr je však nutné vytvoriť objekty, ktoré sú potrebné pre samotný beh algoritmu a animáciu. Tieto budeme potrebovať v ďalšej fáze testu, v simulácií.

```
StorehouseMap map = StorehouseMap.generateMap(21, 21);
FastPathFinder pathFinder = new FastPathFinder(map);
StorehouseResources resources = new StorehouseResources();
TasksBuffer buffer = new TasksBuffer();
```

Ďalším krokom je definícia pracovníkov v sklade. Každý pracovník musí mať nastavenú štartovaciu pozíciu, pre lepšiu orientáciu aj meno. Ďalšie parametre sú veľkosť a rýchlosť, prevzaté od zdroja.

```
Employee emp1 = new Employee(0, 2, 11, "Jana␣Vaskova", 1, 2);
resources.addEmployee(emp1);
```

Každý pracovník musí mať priradený vozík, z dôvodu už spomínaného zjednodušenia. Dáta je nutné uložiť do databázy.

```
Equipment eq1 = new ForkLiftHand(0, 2, 11, "Rucni␣vozik␣1", 4);
daoEquipment.insert(eq1);
resources.addVehicle(eq1);

emp1.setEquipment(eq1);
daoEmployee.insert(emp1);
daoEquipment.update(eq1);
```

Posledným krokom je vytvorenie paliet tovaru a pridelenie úloh k týmto paletám. Všetkým úlohám musí byť predaný konfiguračný objekt aplikácie, ktorý obsahuje potrebné objekty – mapu skladu, zdroje, zásobník úloh a objekt, ktorý pomáha hľadať cestu skladom. Úlohy obsahujú informácie o aktuálnej pozícií tovaru, o konečnej pozícií kam sa má tovar uskladniť a informácie o penalizácií, pokiaľ by táto úloha meškala.

```
Commodity c1 = new Commodity(0, 4, "Karoserie");
daoCommodity.insert(c1);

config = new AppConfig(map, resources, buffer, pathFinder);

Task t1 = new TaskInStore(config, "1", 1, 0, 3, 6, c1, new DeadlinePenalty(3.0), null);
daoTaskInStore.insert(t1);
buffer.addTask(t1);
```

Takto pripravené dáta sa použijú pre vytvorenie návrhu riešenia človekom. Každý výsledok testu pozostáva z dvoch častí – výsledok riadenia človekom a výsledok riadenia algoritmom genetického programovania. Aby sme boli schopný zaznamenať všetky procesy, ktoré je nutné vykonať pre uloženie tovaru, bola navrhnutá trieda `ForkLiftSim` ktorá zjednodušuje prácu s vkladaním dát, napr.:

```
ForkLiftSim sim1 = elg.createSim(eq1, c1.getId(), emp1.getId());
sim1.move(2, 0, t1.getDescription());
sim1.load(1, 0, t1.getDescription());
sim1.move(2, 0, t1.getDescription());
sim1.move(2, 6, t1.getDescription());
sim1.unload(3, 6, t1.getDescription());
```

Všetky akcie sa zapisujú pomocou `EventLogGenerator` do výstupného súboru XML, ktorý je určený pre prípadné neskoršie využitie.

3.1.3 Testovanie v dávkach

Aby sa dali testy spustiť automaticky, jeden za druhým, boli navrhnuté dávkové testy. Trieda `BatchFile` sa stará o dávkové spustenie testov a o zápis výsledkov do tabuľky. V tejto triede sa definuje rozsah testov, ktoré požadujeme spustiť. Prebehne tu automatická inicializácia dát a výpočet času, ako by daný problém riešil človek. Pripravené XML sú následne použité v genetickom algoritme, ktorému sú postupne povolené rôzne genetické operátory. Z každého spusteného testu je získaný čas potrebný na splnenie všetkých úloh.

3.1.4 Výsledky

Sadou testov sme získali dáta pre porovnanie rozvrhovania práce človekov a genetickým algoritmom. V tabuľkách sú uvedené rôzne kombinácie genetických operátorov a výsledné časy potrebné pre úspešné vykonanie všetkých úloh. Jednotlivé stĺpce sú označené nasledovne:

- **ČL** – celkový čas práce pri rozvrhovaní prác človekom (vedúcim prevádzky),
- **P** – využitie `PathMutation`, teda operátoru zmenu cesty,
- **TO** – využitie `TaskOrderMutation`, teda výmeny poradia úloh,
- **ST** – využitie `SwapTaskMutation`, teda výmena medzi pracovnými plánmi,

- **SW** – využitie `SwapWorkplanMutation`, teda výmena pracovných plánov,
- **SP** – využitie `SplitTaskMutation`, teda rozdelenie úloh,

pričom stĺpce označené ako **xGO** predstavujú kombinácie genetických operátorov, kde **x** definuje ich počet. Číselné hodnoty v jednotlivých stĺpcoch predstavujú čas potrebný na vykonanie všetkých úloh, udaný v jednotkách ktoré sme používali v algoritme a simuláciách. Pre lepšiu viditeľnosť výsledkov bol pridaný stĺpec **ÚSP**, ktorý definuje úspech genetického algoritmu, čo je vlastne percentuálne vyjadrenie zlepšenia času pri riešení problému počítačom.

Beh algoritmu bol zastavený pri 20. generácií, počet jedincov v jednotlivých generáciách bol obmedzený na 20. Vo fitness funkcií mala najvyššiu váhu hodnota obsahujúca maximálny čas potrebný na vykonanie všetkých úloh, váha kolízií bola najmenšia. Nastavenie bolo zvolené tak, aby sa práca rozdelila rovnomerne medzi pracovníkov a aby pracovníci nestáli na mieste z dôvodu vyhnutiu sa prípadným kolíziám.

Tab. 3.6: Tabuľka porovnania výsledkov testov 1–20.

Test	ČL	P	TO	ST	SW	SP	ÚSP	2GO	ÚSP	3GO	ÚSP	5GO	ÚSP
1	7,0	7,0	7,0	7,0	7,0	7,0	0%	7,0	0%	7,0	0%	7,0	0%
2	8,2	8,2	8,2	8,2	8,2	8,2	0%	8,2	0%	8,2	0%	8,2	0%
3	7,0	12,2	12,2	12,2	12,2	12,2	-42%	12,2	-42%	12,2	-42%	12,2	-42%
4	8,2	8,2	8,2	8,2	8,2	8,2	0%	8,2	0%	8,2	0%	8,2	0%
5	6,8	6,8	6,8	6,8	6,8	6,8	0%	6,8	0%	6,8	0%	6,8	0%
6	7,5	11,5	11,5	11,5	11,5	11,5	-35%	11,5	-35%	11,5	-35%	11,5	-35%
7	7,7	7,7	7,7	7,7	7,7	7,7	0%	7,7	0%	7,7	0%	7,7	0%
8	5,7	5,3	5,3	5,3	5,3	5,3	6%	5,3	6%	5,3	6%	5,3	6%
9	7,0	6,8	6,8	6,8	6,8	6,8	2%	6,8	2%	6,8	2%	6,8	2%
10	8,2	8,0	8,0	8,0	8,0	8,0	2%	8,0	2%	8,0	2%	8,0	2%
11	6,3	6,3	6,3	6,3	6,3	6,3	0%	6,3	0%	6,3	0%	6,3	0%
12	7,3	7,7	7,7	7,7	7,7	7,7	-4%	7,7	-4%	7,7	-4%	7,7	-4%
13	7,0	10,9	10,9	10,9	10,9	10,9	-36%	10,9	-36%	10,9	-36%	10,9	-36%
14	7,3	7,3	7,3	7,3	7,3	7,3	0%	7,3	0%	7,3	0%	7,3	0%
15	6,3	6,4	6,4	6,4	6,4	6,4	-1%	6,4	-1%	6,4	-1%	6,4	-1%
16	7,5	10,6	10,6	10,6	10,6	10,6	-29%	10,6	-29%	10,6	-29%	10,6	-29%
17	7,7	7,1	7,1	7,1	7,1	7,1	8%	7,1	8%	7,1	8%	7,1	8%
18	5,5	5,3	5,3	5,3	5,3	5,3	3%	5,3	3%	5,3	3%	5,3	3%
19	7,0	6,7	6,7	6,7	6,7	6,7	5%	6,7	5%	6,7	5%	6,7	5%
20	8,2	7,0	7,0	7,0	7,0	7,0	17%	7,0	17%	7,0	17%	7,0	17%

V tab. 3.6 môžeme vidieť, že algoritmus dosahuje v testoch 1 až 20 zhruba rovnakú úspešnosť ako človek, v priemere ide o 95% úspešnosť, pričom pri použití viacerých genetických operátorov nedochádzalo k žiadnym zmenám. Vo väčšine prípadov spôsobilo zdržanie hlavne to, že pracovníci ukladajú tovar do regálov vždy z pravej strany.

Výsledky testovacích prípadov 21–30 sú uvedené v tab. 3.7. V nich môžeme pozorovať o niečo väčšiu úspešnosť riešení algoritmu, zhruba 107%. Najlepšiu úspešnosť mal algoritmus v prípadoch, kedy sme mu povolili použiť 3 genetické operátory.

Tab. 3.7: Tabuľka porovnania výsledkov testov 21–30.

Test	ČL	P	TO	ST	SW	SP	ÚSP	2GO	ÚSP	3GO	ÚSP	5GO	ÚSP
21	10,5	10,4	10,4	9,8	10,0	10,4	7%	9,8	7%	9,8	7%	9,8	7%
22	11,7	11,7	11,7	11,7	11,7	11,7	0%	11,7	0%	11,7	0%	11,7	0%
23	13,8	14,8	14,8	14,8	14,8	14,8	-7%	14,0	-2%	14,0	-2%	14,0	-2%
24	17,0	11,5	11,5	11,5	11,5	11,5	48%	11,5	48%	11,5	48%	11,5	48%
25	13,4	12,8	12,5	11,9	12,8	12,6	13%	11,9	13%	11,9	13%	11,9	13%
26	12,3	12,0	12,0	12,0	12,0	12,0	2%	11,2	10%	11,2	10%	11,2	10%
27	20,4	20,8	22,3	20,4	19,5	21,0	4%	19,3	6%	19,3	6%	19,5	4%
28	21,6	27,4	26,1	26,8	27,0	27,9	-17%	25,8	-16%	25,9	-16%	27,0	-20%
29	20,4	25,0	25,0	23,3	26,0	24,4	-13%	21,3	-4%	21,3	-4%	23,1	-12%
30	17,0	16,6	16,6	14,9	16,6	16,6	14%	15,1	12%	14,9	14%	14,9	14%

Musím podotknúť, že všetkým genetickým operátorom bola nastavená 60% pravdepodobnosť použitia, okrem operátora rozdelujúceho úlohy. Tomuto bola nastavená len 10% pravdepodobnosť, keďže predpokladáme, že k deleniu úloh nedochádza až tak často. Z dát môžeme vyčítať, že najväčšiu úspešnosť mali pri riešení operátory `SwapTaskMutation` a `SwapWorkplanMutation`. Pri použití väčšieho počtu operátorov úspešnosť opäť klesá, môžeme však predpokladať, že pri nastavení väčšieho maximálneho počtu generácií by sme sa dopracovali k rovnakým výsledkom.

Tab. 3.8: Tabuľka porovnania výsledkov testov 31–40.

Test	ČL	P	TO	ST	SW	SP	ÚSP	2GO	ÚSP	3GO	ÚSP	5GO	ÚSP
31	14,3	14,8	15,0	16,8	14,8	14,8	-4%	14,8	-4%	14,8	-4%	14,8	-4%
32	14,3	17,0	18,5	16,6	16,1	18,5	-12%	16,1	-12%	16,1	-12%	16,1	-12%
33	14,6	16,6	16,6	17,0	16,6	16,6	-12%	15,9	-8%	15,9	-8%	16,1	-9%
34	16,0	14,0	14,8	14,7	14,0	14,0	14%	14,0	14%	14,0	14%	14,0	14%
35	10,8	10,8	10,8	10,8	10,8	10,8	0%	10,8	0%	10,8	0%	10,8	0%
36	13,6	16,2	15,0	15,3	15,0	15,3	-9%	13,7	0%	13,6	0%	13,6	0%
37	20,4	27,3	27,5	23,0	26,3	24,4	-11%	22,0	-7%	21,3	-4%	23,0	-11%
38	15,3	18,9	21,0	18,9	18,9	18,9	-19%	18,9	-19%	18,9	-19%	18,9	-19%
39	18,3	23,0	22,5	22,1	24,1	25,0	-18%	21,6	-16%	21,8	-16%	22,4	-18%
40	14,3	13,8	13,8	13,5	13,9	15,2	6%	13,2	8%	13,2	8%	13,2	8%

V testoch 31–40 sme dosiahli opäť úspešnosť porovnateľnú s rozvrhovaním prác človekom, zhruba 95%. V tab. 3.8 si môžeme všimnúť, že úspešnosť nie je rovnomerne rozložená. Veľkú úlohu pri týchto testoch hrala náhoda.

Pri testoch z poslednej kategórie (zložité prípady s väčším množstvom úloh) bola dosiahnutá úspešnosť zhruba 173%. V niektorých prípadoch bola dokonca dosiahnutá tak vysoká úspešnosť, že pri riadení prác pomocou genetického algoritmu sme zvládli úspešne splniť všetky úlohy za takmer tretinu pôvodného času. Z tab. 3.9 vidíme, že najväčšú úspešnosť dosahujeme pri použití dvoch až troch genetických operátorov, výborným príkladom sú testy 51–55. Z týchto výsledkov môžeme vyvodiť záver, že k lepším výsledkom pri zložitejších prípadoch je možné sa prepracovať pri použití všetkých genetických operátorov spomínaných v kap. 2.2.4, pokiaľ im však nastavíme správnu pravdepodobnosť použitia.

Tab. 3.9: Tabuľka porovnania výsledkov testov 41–60.

Test	ČL	P	TO	ST	SW	SP	ÚSP	2GO	ÚSP	3GO	ÚSP	5GO	ÚSP
41	18,3	14,3	12,4	12,4	14,3	14,3	47%	12,4	47%	12,4	47%	12,4	47%
42	11,5	7,9	7,9	7,9	7,9	7,9	46%	7,9	46%	7,9	46%	7,9	46%
43	15,5	8,0	8,0	8,0	8,0	8,0	94%	8,0	94%	8,0	94%	8,0	94%
44	11,0	8,0	8,0	8,0	8,0	8,0	38%	8,0	38%	8,0	38%	8,0	38%
45	16,4	7,8	7,8	7,8	7,8	7,8	111%	7,8	111%	7,8	111%	7,8	111%
46	10,8	7,1	7,1	7,1	7,1	7,1	51%	7,1	51%	7,1	51%	7,1	51%
47	26,9	16,9	16,4	16,4	16,9	16,9	64%	16,4	64%	16,4	64%	16,4	64%
48	22,3	12,0	12,0	12,0	12,0	12,0	85%	12,0	85%	12,0	85%	12,0	85%
49	16,5	8,2	8,2	8,2	8,2	8,2	102%	8,2	102%	8,2	102%	8,2	102%
50	22,8	13,0	12,0	12,0	13,0	13,0	90%	12,0	90%	12,0	90%	12,0	90%
51	31,9	21,0	20,4	21,0	21,1	21,5	56%	18,9	69%	18,9	69%	20,4	56%
52	27,9	26,0	26,0	26,0	26,0	26,0	7%	25,5	9%	25,6	9%	26,0	7%
53	33,5	28,9	28,3	25,0	28,4	28,4	34%	24,9	35%	24,9	35%	26,4	27%
54	20,0	15,8	15,8	15,8	15,8	15,8	27%	15,8	27%	15,8	27%	15,8	27%
55	29,3	25,8	24,6	23,9	24,0	24,0	23%	23,9	23%	23,9	23%	24,0	22%
56	27,5	14,8	14,8	14,8	14,8	14,8	86%	14,8	86%	14,8	86%	14,8	86%
57	30,1	12,8	12,8	12,8	12,8	12,8	135%	12,8	135%	12,8	135%	12,8	135%
58	37,1	17,6	17,6	17,6	17,6	17,6	111%	17,6	111%	17,6	111%	17,6	111%
59	26,4	13,6	13,6	13,6	13,6	13,6	94%	13,6	94%	13,6	94%	13,6	94%
60	32,2	12,6	12,6	12,6	12,6	13,1	155%	12,6	155%	12,6	155%	12,6	155%

Pamäťová náročnosť genetického algoritmu vykazuje u všetkých testoch podobné hodnoty. Aj pri meraní na najnáročnejších testoch (teda na testoch 55–60) sa špičky hodnôt alokovanej pamäte pohybujú v rozmedzí 500–700MB.

3.2 Generátor syntetických testov

Algoritmus genetického programovania je nutné otestovať aj na väčších príkladoch. Z tohto dôvodu bol navrhnutý generátor testov, ktorý dokáže na základe určitých vstupných parametrov vytvoriť náhodnú ukážku problému riadenia práce v sklade. Generátor je založený na podobnom princípe ako sada vytvorených testov, využíva rovnakú štruktúru kódu popísanú v kap. 3.1.2.

3.2.1 Algoritmus testu

Generátor pozostáva z troch častí, pričom v prvej nás budú zaujímať parametre:

- `employeeCount` – počet dostupných zamestnancov v sklade, ktorým budú nasledovne náhodne priradené vozíky,
- `probabilityForkHand` – pravdepodobnosť, že zamestnancovi bude priradený ručný vozík,
- `probabilityForkLow` – pravdepodobnosť, že sa zamestnancovi pridelí malý vozík – najrýchlejšia varianta,

- `probabilityForkHigh` – pravdepodobnosť pridelenia vysokozdvižného vozíku úrčeného pre najvyššie poschodia.

V prvej časti sa vygeneruje presne daný počet pracovníkov. Pracovníkovi musí byť daná štartovacia pozícia (súradnice x a y), pričom súradnicu x volíme podľa uličky. Tá sa vyberá pomocou vzorca

$$x = 2 \cdot \left(i \bmod \frac{w+1}{2} \right),$$

kde i je počet pracovníkov znížený o počet už vygenerovaných pracovníkov a w je celková šírka skladu, v našom prípade 21. Súradnicu y je volená náhodne v rozsahu uličky. Všetky pozície sa ukladajú do zoznamu `close`, aby nenastala situácia, kedy by na jednu pozíciu boli priradení dvaja pracovníci.

Pri generovaní pracovníka mu je priradené aj meno, ktoré je generované náhodne. Následne sa pracovníkovi priradí vozík. Typ vozíka je vybraný vzhľadom na vstupné parametre, pričom názov „pravdepodobnosť“ nie je výstižný. Jedná sa skôr o pomer zastúpenia daného typu k ostatným. Ak teda zadáme pre ručný vozík hodnotu `probabilityForkHand` na hodnotu `0.5`, a ostatné vozíky nastavíme tak, aby celkový súčet dával hodnotu `1.0`, potom je vysoká pravdepodobnosť, že polovica celkového počtu vozíkov bude typu `ForkLiftHand`. Zadávanie jednotlivých hodnôt v takomto tvare nie je nutné dodržať, algoritmus si dokáže tieto pomery dopočítať z akýchkoľvek zadaných čísel.

V druhej časti algoritmu generovania úloh sa generujú palety s tovarom. Opäť sa počíta s tým, že jeden objekt `Commodity` predstavuje jednu paletu. Keďže tento objekt nemá definovanú počiatočnú pozíciu, generujú sa v tomto kroku aj úlohy uskladnenia týchto paliet, v ktorých sú uložené ich počiatočné a konečné pozície. Tu sú zaujímavé tieto vstupné parametre:

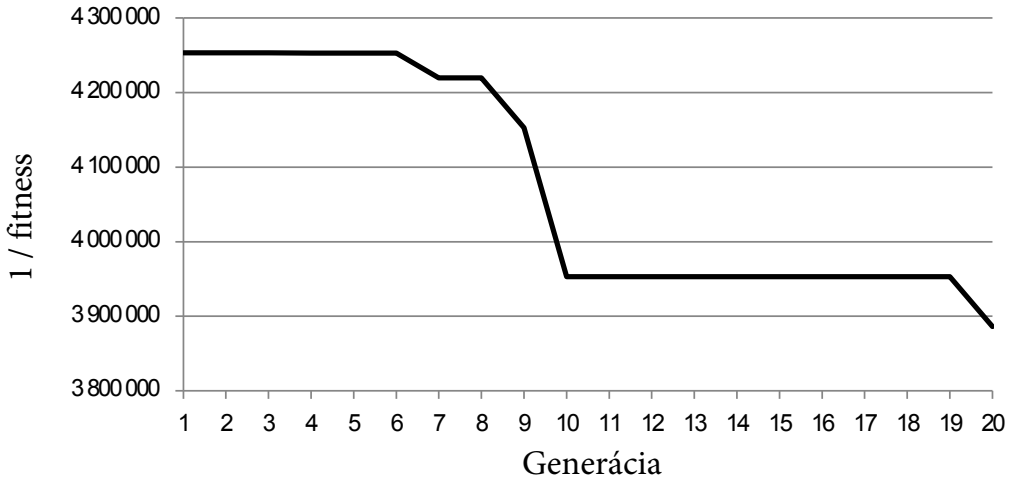
- celkový počet paliet tovaru (`commodityCount`),
- pravdepodobnosť (`probabilityInStore`), že paleta čaká na nákladnej bráne,
- pravdepodobnosť (`probabilityIntermediate`), že nečaká na bráne, ale už je umiestnená na prekladovom mieste.

Jednotlivé palety majú kvôli rozlíšeniu definované len číslo. Počiatočné pozície sa generujú rovnako v predošlom kroku u generovania pracovníkov, pokiaľ teda ide o paletu ktorá je už pripravená na prekladovom mieste. Pokiaľ sa jedná o paletu, ktorá má čakať na nákladnej bráne, vyberá sa náhodne pozícia jednej z nákladných brán. Koncové pozície jednotlivých úloh sú náhodne vybrané pozície regálov, pričom súradnica x je vždy nepárne číslo z rozsahu šírky skladu a súradnica y je celé číslo z rozsahu $\langle 1, h-2 \rangle$, kde h je výška skladu[1].

[1] Výškou skladu rozumieme veľkosť v súradnici y, v prípade dvojrozmerného pohľadu.

3.2.2 Výsledky

Názorná ukážka vývoja hodnôt fitness funkcie vzhľadom na aktuálnu generáciu sa nachádza na obr. 3.6. Hodnoty pre tento graf boli získané z jedného náhodne vygenerovaného prípadu s náhodnou kombináciou pravdepodobností[2] jednotlivých genetických operátorov. Vertikálna os tohto grafu predstavuje prevrátenú hodnotu fitness funkcie, teda hodnoty ktoré vracia trieda `MultiObjectiveFitness`. Z priebehu je zrejmé, že v niektorých generáciách nedochádza k žiadnemu zlepšeniu vlastností jedincov. V desiatej generácií však nastáva skok a najlepší jedinec vykazuje omnoho lepšie výsledky. Ide o čiste náhodný prípad, grafické zobrazenie iného prípadu s použitím iných kombinácií pravdepodobností operátorov by mohlo klesať omnoho rýchlejšie.



Obr. 3.6: Vývoj hodnôt fitness funkcie.

V tab. 3.10 sa nachádza porovnanie výsledkov vo vygenerovaných prípadoch s desiatimi úlohami, ktoré majú byť spracované piatimi pracovníkmi. Výsledné časy a počty kolízií sú vypočítané priemerom z 5 náhodne vygenerovaných testov. Zvýraznené hodnoty sú najnižšie dosiahnuté, teda konkrétna kombinácia pravdepodobností použitia daného operátora zabezpečí čo najnižší čas spracovania, respektíve čo najnižší počet kolízií v sklade. V záhlaví tabuľky je použité rovnaké značenie ako v kap.3.1.4.

Záverom je teda, že pre jednoduchšie problémy (zhruba 10 úloh) bude najvhodnejšia kombinácia pravdepodobností 60/60/60/60/20, pokiaľ chceme dosiahnuť čo najnižší čas spracovania úloh. Aby však nastalo zníženie počtu kolízií, musí byť použitá kombinácia pravdepodobností 60/60/60/20/20.

[2]Vždy však ide o hodnoty 20% alebo 60%, viac hodnôt by výrazne zväčšilo dĺžku simulácie.

Tab. 3.10: Porovnanie vplyvu pravdepodobnosti použitia operátorov.

P	TO	ST	SW	SP	10 úloh		50 úloh	
					Čas	Kolízie	Čas	Kolízie
20%	20%	20%	20%	20%	**19,01**	16,80	37,08	331,80
60%	20%	20%	20%	20%	21,23	13,80	37,04	320,00
20%	60%	20%	20%	20%	20,18	14,20	38,32	312,80
60%	60%	20%	20%	20%	19,71	13,20	38,53	320,40
20%	20%	60%	20%	20%	**18,91**	13,80	36,95	332,40
60%	20%	60%	20%	20%	20,23	14,80	37,88	327,00
20%	60%	60%	20%	20%	20,47	15,80	36,78	304,40
60%	60%	60%	20%	20%	20,33	**9,40**	36,01	333,00
20%	20%	20%	60%	20%	20,43	11,80	**34,38**	323,60
60%	20%	20%	60%	20%	19,74	12,00	36,53	322,80
20%	60%	20%	60%	20%	20,31	15,60	36,68	321,80
60%	60%	20%	60%	20%	21,51	15,80	36,23	332,60
20%	20%	60%	60%	20%	20,55	14,00	35,48	369,40
60%	20%	60%	60%	20%	20,17	13,00	**34,35**	329,60
20%	60%	60%	60%	20%	19,24	17,20	35,97	361,20
60%	60%	60%	60%	20%	**18,70**	13,00	**34,96**	343,20
20%	20%	20%	20%	60%	21,98	14,40	37,15	367,00
60%	20%	20%	20%	60%	21,45	16,40	36,14	335,80
20%	60%	20%	20%	60%	20,95	18,20	37,33	335,00
60%	60%	20%	20%	60%	19,48	14,20	36,41	351,00
20%	20%	60%	20%	60%	21,38	13,20	38,53	309,60
60%	20%	60%	20%	60%	19,39	10,80	36,53	338,20
20%	60%	60%	20%	60%	22,83	11,40	36,24	323,20
60%	60%	60%	20%	60%	20,90	14,20	37,07	361,80
20%	20%	20%	60%	60%	21,06	12,40	38,66	**292,60**
60%	20%	20%	60%	60%	20,40	12,80	37,33	347,60
20%	60%	20%	60%	60%	19,38	14,00	36,87	316,60
60%	60%	20%	60%	60%	21,10	13,20	37,70	337,20
20%	20%	60%	60%	60%	21,13	14,80	37,88	350,40
60%	20%	60%	60%	60%	21,07	13,80	38,08	313,20
20%	60%	60%	60%	60%	19,45	14,60	36,93	349,80
60%	60%	60%	60%	60%	19,59	12,20	38,28	321,60

V tab. 3.10 sú uvedené aj výsledky pre väčšie prípady, kedy má 20 pracovníkov splniť 50 úloh. Požiadavky na nastavenie pravdepodobností genetických operátorov sa mierne odlišujú. Z výsledkov je zrejmé, že vhodnejšia kombinácia pre minimalizáciu času spracovania je 60/20/60/60/20, pre minimalizáciu kolízií je to potom 20/20/20/60/60.

Hodnoty pamäťovej náročnosti spomínané v kap. 3.1.4 boli potvrdené, beh testov všetkých kombinácií pravdepodobností pri 50 úlohách bol však omnoho výpočtovo náročnejší. Doba trvania testovania bola zhruba 15 hodín.[3]

[3]Test bežal na notebooku zakúpenom v roku 2011, procesor Intel Core i7 2620M Sandy Bridge s frekvenciou 2,7 GHz, operačný systém Windows 7.

3.3 Paralelizácia algoritmu

Možnosti testovania algoritmu sú obmedzené najmä jeho výpočtovou náročnosťou. Pokiaľ je potreba testovať algoritmus aj s väčšími populáciami, zvyšuje sa časová náročnosť behu algoritmu. Zvýšením veľkosti populácie o jeden rád súčasne zvýšime časovú náročnosť algoritmu takisto zhruba o jeden rád. Túto závislosť je možné vidieť v tab. 3.11. Zmenšenie strmosti tejto závislosti možno dosiahnuť paralelizáciou algoritmu, respektíve niektorých jeho kritických častí.

Tab. 3.11: Výpočtová náročnosť algoritmu v testoch 41–50.

Veľkosť	**Test a doba behu v sekundách**									
populácie	41	42	43	44	45	46	47	48	49	50
20	18,9	16,0	16,3	17,1	17,0	16,6	18,7	19,8	16,8	21,4
200	180,4	160,1	182,5	184,3	180,1	192,5	206,5	212,6	185,9	219,7
2000	2164,7	1874,0	2605,6	2493,8	2353,2	2620,3	2862,3	2603,2	2279,3	2763,0

3.3.1 Paralelizácia inicializácie

Inicializácia je založená na jednoduchom vytváraní náhodných chromozómov v cykle až pokiaľ populácia nenadobudne požadovanú veľkosť. Úlohou paralelizácie v tomto prípade bude rozdeliť toto vytváranie počiatočnej populácie do viacerých vlákien, pričom v každom vlákne sa vytvorí len časť populácie. Počet chromozómov uložený v `currentSize` je predaný ako parameter konštruktoru daného vlákna, spolu s objektom nastavenia aplikácie, ktorý je potrebný pre vytvorenie chromozómu. Objekty vlákien sa ukladajú do poľa, pričom beh každého vytvoreného vlákna je okamžite spustený prostredníctvom `thread.start()`.

```
ArrayList<InitializePopulationThread> threads = new ArrayList<InitializePopulationThread>();
int restAfterModulo = populationSize % Config.NUM_OF_THREADS;
int popSizeInThread = (populationSize - restAfterModulo) / Config.NUM_OF_THREADS;
int min = Math.min(Config.NUM_OF_THREADS, populationSize);
for (int i = 0; i < min; i++) {
        int currentSize = (popSizeInThread + (restAfterModulo-- > 0 ? 1 : 0));
        InitializePopulationThread thread =
                new InitializePopulationThread(currentSize, conf);

        threads.add(thread);
        thread.start();
}
```

Po dobehnutí všetkých vlákien sa chromozómy z každého vlákna zozbierajú a už existujúce objekty chromozómov sa vložia do doposiaľ prázdnej populácie.

```
for(int i = 0; i < min; i++) {
        try {
                threads.get(i).join(); // wait until thread end
        } catch (InterruptedException e) {
                e.printStackTrace();
        }
        addAll(threads.get(i).getChromosomes());
}
```

Trieda `InitializePopulationThread` je teda pomerne jednoduchá, jej úlohou je postarať sa o naplnenie zoznamu chromozómov a možnosť vrátiť tento zoznam. O vytvorenie náhodného chromozómu sa postará konštruktor chromozómu, ktorému stačí poskytnúť už spomínané nastavenie aplikácie.

```
chromosomes = new ArrayList<Chromosome>();
for (int i = 0; i < size; i++) {
        chromosomes.add(new Chromosome(conf));
}
```

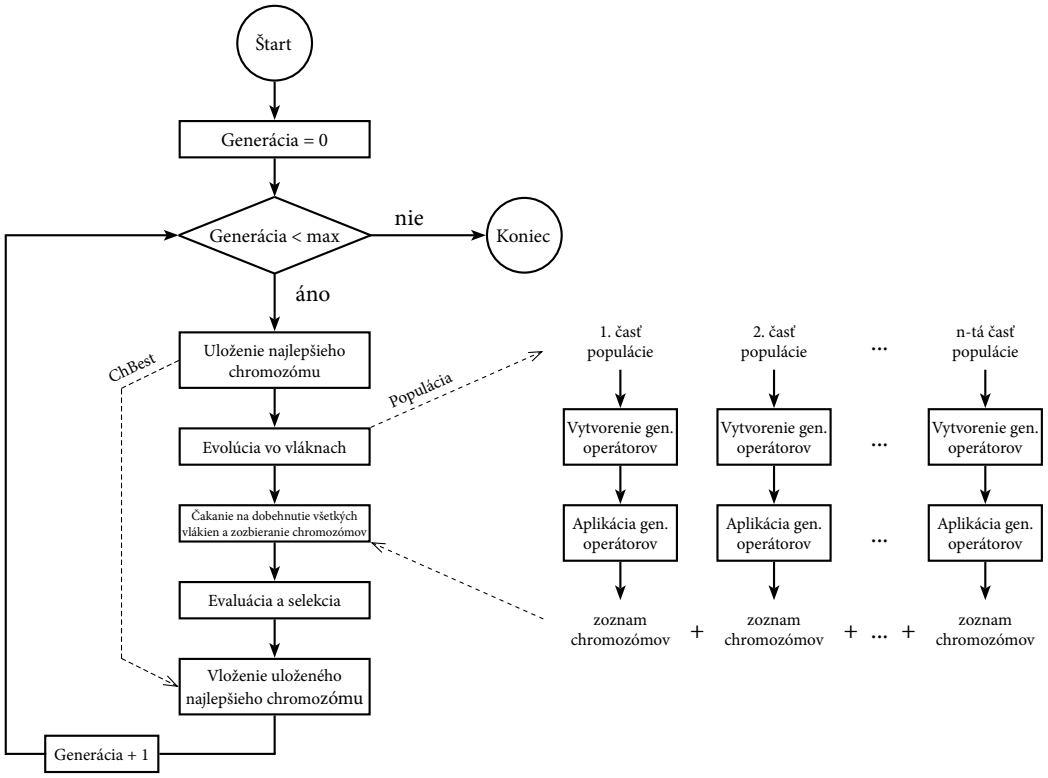
3.3.2 Paralelizácia evolúcie

Proces evolúcie je založený na postupnom spracovaní všetkých chromozómov, pričom jednotlivé genetické operátory sú použité na daný chromozóm na základe zadanej pravdepodobnosti. Tu sa dá opäť uplatniť rovnaká myšlienka ako v kap. 3.3.1 – rozdeliť beh algoritmu do viacerých vlákien, pričom v každom vlákne sa spracuje len časť populácie. Priebeh obsluhy evolúcie je naznačený na diagrame na obr. 3.7, ako aj samotné vlákna evolúcie.

Pre zachovanie najlepšieho možného výsledku dochádza na začiatku ku klonovaniu najlepšieho chromozómu a na konci evolučného cyklu k jeho vloženiu do novej populácie. Zrýchlenie behu jedného evolučného cyklu možno očakávať hlavne v prípadoch, kedy budú použité operátory s väčšou pravdepodobnosťou aplikácie. Pri nevyužití žiadneho operátora v danom vlákne k zrýchleniu evolúcie nedôjde. Skôr naopak – k spomaleniu, keďže je nutné celú populáciu deliť na časti a následne chromozómy vkladať späť.

3.3.3 Výpočtová náročnosť po paralelizácií

Algoritmus bol podrobený rovnakým testom ako v kap. 3.3, ide o testy 41 až 60 založené na reálnych dátach, pričom sledujeme dobu potrebnú na výpočet 20 generácií, so zvyšujúcim sa počtom chromozómov v jednom evolučnom cykle. Tieto konkrétne testy boli vybrané kvôli ich náročnosti, ktorou sa približujú k náročnosti v reálnom logistickom sklade v určitom krátkom časovom intervale. Porovnanie náročnosti pred a po paralelizácií sa nachádza v tab.3.12. To že vyšší počet chromozómov je žiadúci



Obr. 3.7: Priebeh obsluhy vlákien evolúcie.

vidno na obr. 3.8. Pri vyššom počte chromozómov dochádza k zlepšeniam kvality riešenia častejšie, teda dochádza k rýchlejšej iterácií k optimálnym riešeniam.

Pomocou paralelizácie inicializácie (kap. 3.3.1) a evolúcie (kap. 3.3.2) sa dosiahlo v testoch priemerné zlepšenie o 10%, pričom v niektorých prípadoch aj o viac ako 20% oproti pôvodnému času, ktorý je potrebný na výpočet 20 evolučných cyklov.

So zvyšujúcim sa počtom chromozómov v jednom cykle sa síce dostávame k lepším výsledkom omnoho rýchlejšie, algoritmus dokáže prehľadávať omnoho väčší priestor v jednej generácií, ale zvyšuje sa pamäťová náročnosť, ktorej špičky dosahovali pre 2000 chromozómov v niektorých prípadoch až cez 4GB. V praxi teda bude nutné využiť výpočtovú jednotku s väčšou operačnou pamäťou.

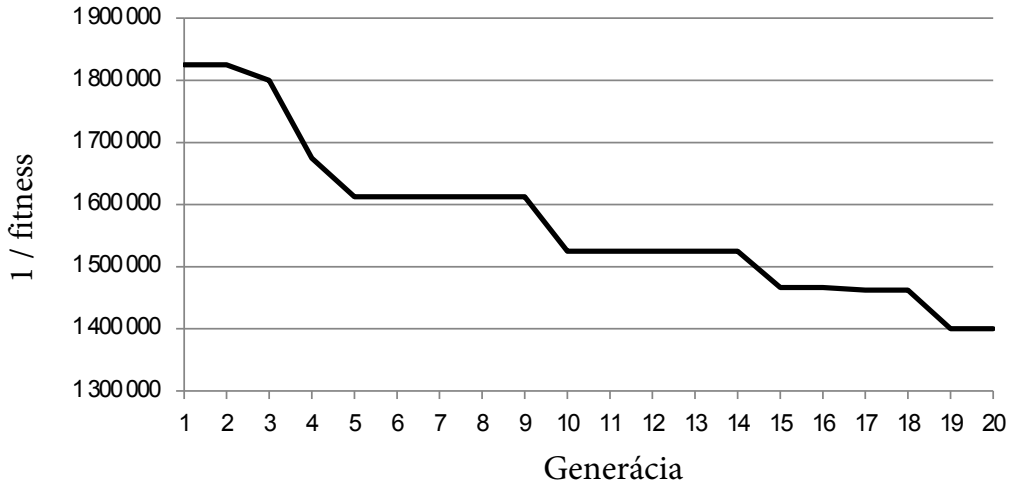
3.4 Implementácia prioritných pravidiel

Pokiaľ sa implementujú do algoritmu genetického programovania niektoré z pravidiel z kap. 1.2, možno očakávať zlepšenie výsledkov. Pravidlá napomáhajú algoritmu genetického programovania rýchlejšie sa dopracovať k priestoru s riešeniami ktoré by mohli byť lepšie, prípadne tieto riešenia vedú k iným možným lepším

Tab. 3.12: Porovnanie výpočtovej náročnosti algoritmu v testoch 41–60.

Veľkosť pop.	20 chromozómov			200 chromozómov			2000 chromozómov		
Vlákien	1	4	**Zlepšenie**	1	4	**Zlepšenie**	1	4	**Zlepšenie**
Test 41	18,9	18,8	0,3%	180,4	148,8	17,5%	2164,7	1884,8	12,9%
Test 42	16,0	14,5	9,6%	160,1	132,1	17,5%	1874,0	1621,3	13,5%
Test 43	16,3	17,0	-4,3%	182,5	155,4	14,9%	2605,6	2251,3	13,6%
Test 44	17,1	15,9	7,3%	184,3	150,1	18,6%	2493,8	2052,4	17,7%
Test 45	17,0	16,2	4,6%	180,1	154,3	14,3%	2353,2	2018,9	14,2%
Test 46	16,6	17,8	-7,2%	192,5	157,9	18,0%	2620,3	2302,8	12,1%
Test 47	18,7	18,3	2,1%	206,5	190,3	7,8%	2862,3	2411,2	15,8%
Test 48	19,8	17,6	10,9%	212,6	179,0	15,8%	2603,2	2055,7	**21,0%**
Test 49	16,8	15,9	5,6%	185,9	155,1	16,6%	2279,3	1891,7	17,0%
Test 50	21,4	20,0	6,5%	219,7	184,5	16,0%	2763,0	2209,4	**20,0%**
Test 51	29,7	23,3	**21,4%**	357,5	244,4	**31,6%**	3599,7	3598,1	0,0%
Test 52	24,3	23,8	2,2%	325,8	269,7	17,2%	3695,5	3336,7	9,7%
Test 53	25,1	22,0	12,3%	322,3	280,6	12,9%	3524,9	3359,6	4,7%
Test 54	17,2	16,8	2,1%	187,2	151,4	19,1%	2027,6	1868,0	7,9%
Test 55	21,0	24,2	-15,2%	285,1	227,2	**20,3%**	3165,6	3082,0	2,6%
Test 56	19,3	18,8	2,2%	213,6	175,6	17,8%	2292,7	2130,3	7,1%
Test 57	18,7	17,3	7,5%	193,1	160,3	17,0%	2077,1	1937,7	6,7%
Test 58	22,0	25,5	-16,0%	282,4	216,2	**23,5%**	3052,5	2844,7	6,8%
Test 59	16,3	17,4	-6,9%	187,1	159,1	15,0%	2068,3	1908,3	7,7%
Test 60	21,9	23,8	-8,7%	221,8	190,9	13,9%	2471,5	2265,6	8,3%

riešeniam. Pravidlá boli navrhnuté tak, aby boli schopné pracovať s aktuálnymi dostupnými dátami, aby nevyžadovali žiadne ďalšie informácie k jednotlivým úlohám (ako napr. dôležitosť jednotlivých úloh) a mali jednotné rozhranie, ktoré bude možné využiť pre implementáciu ďalších pravidiel v budúcnosti. Pre možnosti testovania, ako aj pre zistenie najvhodnejšieho miesta aplikovania pravidiel na aktuálnu populáciu, boli v algoritme implementované rôzne možnosti zaobchádzania s pravidlami:



Obr. 3.8: Vývoj hodnôt fitness funkcie pre 2000 chromozómov.

1. použitie samostatných pravidiel bez evolúcie – najmä pre účely testovania,
2. aplikovanie pravidiel na finálnu populáciu po skončení evolúcie,
3. rozšírenie populácie pravidlami každý n-tý krok evolúcie, pričom n je možné definovať v konfigurácií.

Štruktúra pravidiel bola inšpirovaná návrhovým vzorom *Specification*, pomocou ktorého je zabezpečená možnosť reťazenia pravidiel. Diagram jednotlivých tried a väzieb prioritných pravidiel sa nachádza na obr. A.4.

3.4.1 Pravidlo obmedzenia zmien

Ide o pravidlo RS bližšie teoreticky popísané v kap. 1.2.3. Toto pravidlo z pohľadu algoritmu genetického programovania kompletne mení poradie úloh v pracovných plánoch. Tieto vstupujú do ďalšej evolúcie, pričom je snaha ako prvé spracovať úlohy ktoré majú najväčšiu vzdialenosť od štartu k cieľu. Pravidlo postupne prechádza všetky chromozómy a vytvára klony s upraveným poradím úloh, takže nedochádza k strate žiadnych už nájdených vhodných jedincov. Nový upravený chromozóm je vložený do populácie iba v tom prípade, že je vhodnejší ako najlepší chromozóm z aktuálnej populácie. Takto rozšírená populácia je podrobená evaluácií a turnajovej selekcií. Rovnaký postup platí aj pre ostatné pravidlá.

Pre zistenie skutočného dopadu pravidla na výslednú populáciu bol vytvorený test, ktorý ihneď po inicializácií populácie aplikuje pravidlo a porovná výsledný čas. Pre tento test bola použitá sada reálnych testov z kap. 3.1, pričom každá hodnota bola vypočítaná ako priemer z piatich výsledných časov jedného testu. Hodnoty boli porovnané s časom, ktorý bol dosiahnutý pre daný test ihneď po inicializácií. V tab. 3.13 sú vypísané testy, v ktorých bolo dosiahnuté zníženie výsledného času

Tab. 3.13: Porovnanie výsledného času pravidla RS.

Test	Inicializácia	Pravidlo	Zlepšenie
21	11,11	10,88	2,03%
25	13,23	12,80	3,21%
28	28,72	27,87	2,96%
29	25,95	25,89	0,22%
30	16,80	16,63	1,04%
37	28,45	27,48	3,43%
39	26,65	26,50	0,56%
40	17,78	17,68	0,61%
55	28,10	27,70	1,42%

potrebného na splnenie úloh. Maximálne zlepšenie bolo dosiahnuté pri teste č. 37. V testoch 1 až 20 nebolo dosiahnuté žiadne zlepšenie, výsledok po prvotnej inicializácií bol najlepší možný a teda lepšie riešenie vzhľadom na výsledný čas už nie je možné nájsť.

3.4.2 Pravidlo najkratšieho času spracovania

Ďalším aplikovaným pravidlom na problém riadenia práce v logistickom sklade je SPT z kap. 1.2. Pravidlo má za úlohu uprednostniť úlohy, ktoré majú celkový potrebný čas na spracovanie čo najmenší. Dochádza teda k uprednostneniu krátkych úloh ktoré sa nachádzajú v danom okamihu najbližšie k pracovníkovi a vyžadujú najmenej času na úspešné dokončenie.

Tab. 3.14: Porovnanie výsledného času pravidla SPT.

Test	Inicializácia	Pravidlo	Zlepšenie
26	12,05	12,00	0,41%
28	29,83	29,43	1,34%
34	16,13	16,10	0,21%
36	16,58	16,49	0,50%
47	17,03	16,93	0,59%
56	15,43	15,38	0,32%

V tab. 3.14 sú zaznamenané testy, na ktoré malo toto pravidlo priaznivý vplyv. Výsledné zlepšenie je oproti predošlému pravidlu nižšie. Môžeme však očakávať, že v kombinácií s evolúciou budú dosiahnuté lepšie výsledky.

3.4.3 Pravidlo rýchleho vozíka

Posledným implementovaným pravidlom do algoritmu genetického programovania je pravidlo, ktoré vychádza z praxe a v teoretickej časti práce nebolo spomenuté. Pravidlo má na starosti úlohy s dlhou trasou priradiť rýchlym vozíkom, pričom je zabezpečené, aby nedochádzalo k preťaženiu jedného rýchleho vozíka, ale aby sa záťaž rovnomerne rozdelila na vozíky s vyššou rýchlosťou. Aplikáciou pravidla by bolo teda zabezpečené, aby ručné vozíky s nízkou rýchlosťou prevážali tovar len na kratšie vzdialenosti.

Pravidlo dosahuje najlepšie výsledky spomedzi implementovaných pravidiel. Jednoduchým použitím pravidla ihneď po inicializácií populácie je možné dosiahnuť zlepšenie v priemere o 5%, pričom v niektorých testoch bolo dosiahnuté zlepšenie výsledného času až o takmer 15% nižšie. V tab. 3.15 sú testy, v ktorých bolo dosiahnuté zlepšenie výsledných časov.

3.4.4 Výsledky syntetických testov

Testovanie účinku použitia pravidiel v algoritme genetického programovania prebiehalo v troch možných nastaveniach. Prvým bolo použitie všetkých pravidiel po skončení algoritmu, teda v 20-tej generácií, druhým nastavením bolo použitie pravidiel v každej piatej generácií, no a tretím je použitie v každej druhej generácií. Testy

Tab. 3.15: Porovnanie výsledného času pravidla FFL.

Test	Inicializácia	Pravidlo	Zlepšenie
3	12,50	12,17	2,67%
13	13,27	12,30	7,29%
19	8,08	7,10	12,07%
20	7,40	7,33	0,90%
22	11,90	11,67	1,96%
23	15,82	15,33	3,11%
29	27,32	26,23	3,97%
31	17,37	16,77	3,45%
32	20,61	20,09	2,51%
33	19,94	19,54	2,01%
34	16,13	15,40	4,55%
38	23,95	23,73	0,94%
39	26,65	25,60	3,94%
40	17,65	15,19	13,93%
41	14,25	12,75	10,53%
43	8,70	8,68	0,29%
47	17,03	16,78	1,47%
50	14,60	12,45	14,73%
53	31,55	30,33	3,88%
57	15,17	14,00	7,69%
60	15,10	14,03	7,12%

boli spustené s optimálnym nastavením pravdepodobností genetických operátorov pre danú veľkosť problému a s kombináciou nastavení pravdepodobností ktoré dosahovalo v predošlých testoch horšie výsledky.[4]

Tab. 3.16: Vplyv prioritných pravidiel na malé úlohy pri optimálnych nastaveniach.

Prioritné	na konci evolúcie		každú 5. generáciu		každú 2. generáciu	
pravidlá	Čas	Kolízie	Čas	Kolízie	Čas	Kolízie
RS + SPT + FFL	22,15	14,20	19,79	13,00	21,00	15,20
SPT + FFL	20,18	20,60	21,38	11,20	21,61	16,60
RS + FFL	21,03	11,80	20,83	11,20	20,78	11,40
FFL	21,08	18,40	21,02	15,80	21,05	15,80
RS + SPT	20,78	11,20	22,02	13,20	21,68	12,00
SPT	22,43	12,20	23,27	13,80	19,53	13,20
RS	20,19	11,60	21,84	14,20	22,03	11,80
Bez pravidiel	19,83	14,00	20,43	15,40	20,75	13,00
MIN:	19,83	11,20	19,79	11,20	19,53	11,40
AVG:	20,96	14,25	21,32	13,48	21,05	13,63
MED:	**20,91**	13,10	21,20	**13,50**	21,03	13,10

V tab. 3.16 sa nachádzajú výsledné priemerné časy a počty kolízií vždy z piatich náhodne vygenerovaných syntetických testov. Optimálnym nastavením parametrov sa rozumie nastavenie pravdepodobností genetických operátorov na kombináciu

[4]Výsledky všetkých kombinácií sú uvedené v kap. 3.2.2.

60/60/60/60/20 v prípade menšieho problému, v tomto konkrétnom prípade 10 úloh na 5 pracovníkov skladu. Z výsledných hodnôt je vidno, že vhodným nastavením bude použitie prioritných pravidiel vo väčšom rozostupe evolučných cyklov. Zníženie počtu kolízií je možné dosiahnuť použitím pravidla FFL v kombinácií s aspoň jedným ďalším pravidlom pri aplikovaní pravidiel aspoň každú piatu generáciu.

Tab. 3.17: Vplyv prioritných pravidiel na malé úlohy.

Prioritné	**na konci evolúcie**		**každú 5. generáciu**		**každú 2. generáciu**	
pravidlá	**Čas**	**Kolízie**	**Čas**	**Kolízie**	**Čas**	**Kolízie**
RS + SPT + FFL	22,91	14,00	20,53	11,40	21,09	14,80
SPT + FFL	21,36	13,00	20,69	13,60	20,35	14,00
RS + FFL	20,48	11,80	19,91	12,60	21,13	9,80
FFL	21,98	17,60	20,66	13,60	19,99	**8,20**
RS + SPT	22,08	10,80	22,10	11,20	20,40	16,40
SPT	21,37	11,00	21,77	14,80	21,85	15,40
RS	21,27	10,80	20,33	12,00	20,03	14,80
Bez pravidiel	21,54	11,20	21,79	14,60	20,57	13,40
MIN:	20,48	10,80	19,91	11,20	19,99	8,20
AVG:	21,62	**12,53**	20,97	12,98	**20,68**	13,35
MED:	21,45	**11,50**	20,68	13,10	**20,48**	14,40

Pokiaľ však pravidlá aplikujeme pri nevhodnom nastavení pravdepodobností genetických operátorov, začína mať ich častejšie volanie priaznivý vplyv na celkový výsledný čas potrebný na spracovanie úloh pracovníkmi. V tab. 3.17 sú namerané hodnoty pri nastavení pravdepodobností 60/20/20/20/60 a opäť pre problém nižšej zložitosti. Výsledný čas potrebný na spracovanie všetkých úloh je najnižší v prípade aplikácie pravidiel pri každom druhom cykle evolúcie. Opäť je najvhodnejšie zvoliť kombináciu aspoň dvoch prioritných pravidiel alebo použitie samotného pravidla FFL pre zníženie počtu kolízií. Nízky počet kolízií je možné dosiahnuť aplikáciou pravidiel iba na konci evolúcie, takýmto nastavením sa však nezabezpečí dostatočná rýchlosť iterácie a výsledná vhodnosť môže byť omnoho nižšia, vzhľadom na celkový čas potrebný na spracovanie úloh, ktorý má najväčšiu váhu.

V praxi sa však algoritmus bude využívať hlavne pri väčšom počte úloh, kedy riadenie a rozvrhovanie práce je pre človeka omnoho náročnejšie. Tab. 3.18 zachytáva hodnoty namerané pri optimálnom nastavení pravdepodobností genetických operátorov pre danú veľkosť problému, teda nastavenie 60/20/60/60/20. Z nameraných hodnôt je zrejmé, že pri väčších problémoch je pomáha časté volanie použitia prioritných pravidiel. Dochádza k znižovaniu potrebného času na spracovanie úloh, ako aj znižovaniu počtu kolízií. Najlepší čas bol dosiahnutý pri použití kombinácií pravidiel RS a SPT, prípadne kombinácií SPT a FFL. Všetky testy boli opäť generované náhodne a hodnoty sú vypočítané spriemerovaním 5 testov, teda nájdenie vhodného

Tab. 3.18: Vplyv prioritných pravidiel na väčšie úlohy pri optimálnych nastaveniach.

Prioritné	**na konci evolúcie**		**každú 5. generáciu**		**každú 2. generáciu**	
pravidlá	**Čas**	**Kolízie**	**Čas**	**Kolízie**	**Čas**	**Kolízie**
RS + SPT + FFL	38,33	337,80	38,27	309,00	35,48	343,40
SPT + FFL	36,33	316,60	35,51	378,00	35,09	322,00
RS + FFL	36,04	330,00	36,58	340,00	36,18	334,40
FFL	37,68	322,40	38,69	331,80	36,99	362,60
RS + SPT	38,77	322,60	37,46	321,60	34,98	343,40
SPT	36,31	337,40	36,67	357,00	37,14	334,40
RS	37,88	305,80	38,05	300,8	38,73	294,80
Bez pravidiel	38,94	309,80	36,57	342,60	40,09	310,00
MIN:	36,04	305,80	35,51	300,80	**34,98**	**294,80**
AVG:	37,53	322,80	37,22	335,10	**36,84**	330,63
MED:	37,78	322,50	37,06	335,90	**36,59**	334,40

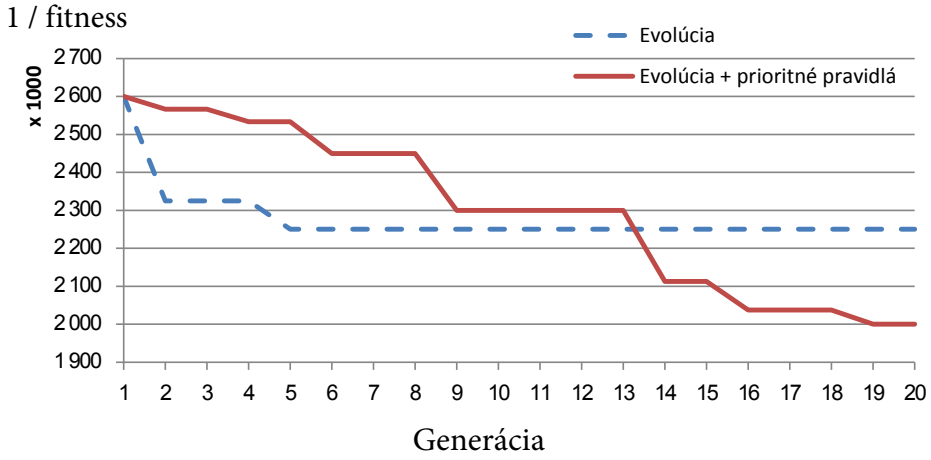
nastavenia by malo pokryť väčšinu problémov ktoré by mohli byť riešené v praxi pomocou tohto algoritmu.

Tab. 3.19: Vplyv prioritných pravidiel na väčšie úlohy.

Prioritné	**na konci evolúcie**		**každú 5. generáciu**		**každú 2. generáciu**	
pravidlá	**Čas**	**Kolízie**	**Čas**	**Kolízie**	**Čas**	**Kolízie**
RS + SPT + FFL	39,78	296,80	39,11	318,8	36,49	343,40
SPT + FFL	36,46	320,20	34,32	380	35,42	323,20
RS + FFL	38,13	312,60	38,68	328,4	39,23	341,60
FFL	39,10	348,40	39,03	319	36,03	319,60
RS + SPT	39,57	333,60	39,81	337,2	39,22	322,60
SPT	39,03	301,20	37,38	297,2	37,58	323,20
RS	38,04	352,20	38,98	334,8	37,32	311,20
Bez pravidiel	41,43	322,60	39,4	293	40,63	335,60
MIN:	36,46	296,80	34,32	293,00	35,42	311,20
AVG:	38,94	323,45	38,34	326,05	**37,74**	327,55
MED:	39,07	321,40	39,01	323,70	**37,45**	323,20

Pokiaľ však použijeme nastavenie pravdepodobností 20/20/20/60/60, ktoré nebolo určené ako vhodné nastavenie pre riešenie problémov tohto rozsahu, vplyv použitia prioritných pravidiel sa takmer nezmení. Z hodnôt v tab. 3.19 je vidno, že je opäť vhodné volať prioritné pravidlá pravidelne, čo najčastejšie. Tentokrát nám však nespôsobujú také zníženie kolízií ako v predošlom prípade.

Z toho možno vyvodiť záver, že v praxi bude ideálne použiť prioritné pravidlá každú druhú generáciu, pričom prepracovať sa čo najrýchlejšie k čo najlepším hodnotám bude možné len pri optimálnom nastavení pravdepodobností genetických operátorov, iba vtedy sa dosahuje najrýchlejšia iterácia k optimálnym riešeniam.



Obr. 3.9: Porovnanie priebehu vhodnosti pri použití prioritných pravidiel.

Na obr. 3.9 sú zachytené hodnoty vhodnosti počas priebehu evolúcie v jednotlivých generáciách. Ide o náhodne vygenerovaný syntetický test, pričom populácia bola ihneď po inicializácií naklonovaná, aby výsledok a samotný priebeh nemohol byť ovplyvnený počiatočnou inicializáciou. Najskôr bol spustený test so samotnou evolúciou, pričom na priebehu (modrý čiarkovaný) vidíme že pri tomto konkrétnom teste už pri 5. generácií samotný genetický algoritmus nebol schopný nájsť ďalšie vhodnejšie riešenie. Použitím prioritných pravidiel je však zaistené prehľadávanie väčšieho priestoru v danú chvíľu.

S naklonovanou populáciou bol spustený druhý test, kedy v priebehu evolúcie boli použité prioritné pravidlá, a síce každú druhú generáciu. Na priebehu (červený) vidíme že k zníženiu prevrátenej hodnoty vhodnosti dochádza pravidelne. Síce priebeh nie je tak strmý ako v predošlom prípade, výsledná hodnota je omnoho lepšia, no a v prípade použitia tohto riešenia je možné dosiahnuť omnoho nižší čas potrebný na spracovanie všetkých úloh v sklade.

3.5 Budúcnosť

Algoritmus je do budúcnosti možné rozšíriť o ďalšie prioritné pravidlá a tým zabezpečiť prehľadávanie väčšieho priestoru súčasne, pričom sa vychádza z určitej logiky danej pravidlom. Algoritmus genetického programovania tieto lepšie riešenia následne zahrnie do populácie a pracuje s nimi. Štruktúra pravidiel je na ďalšie rozširovanie pripravená, popisuje príloha A.4 s jednoduchým UML diagramom.

V logistických skladoch majú jednotlivé úlohy tiež danú váhu, teda dôležitosť jednotlivých úloh. Na toto je nutné pamätať a algoritmus genetického programovania

by bolo dobré o toto rozšíriť a použiť tieto hodnoty váh pri počítaní vhodnosti, ako aj pri prioritných pravidlách (takéto pravidlá popisuje napr. [4], [5] alebo [26]).

V praxi je nutné dáta dostať v reálnom čase k jednotlivým pracovníkom. K tomu je nutné prispôsobiť určité periférie k spolupráci s algoritmom riadenia a rozvrhovania práce. Algoritmus by mal pravidelne aktualizovať možné riešenia a pracovať v reálnom čase s dynamickým problémom (bližšie popisuje kap. 1.1.2), no a v prípade zmeny prostredníctvom týchto periférií pracovníkov informovať.

4 ZÁVER

Súčasťou teoretickej časti tejto práce bola štúdia týkajúca sa rozvrhovacích problémov, ako aj súčasného stavu vedy a techniky v tomto smere. Jednotlivé možné riešenia problému rozvrhovania práce boli teoreticky porovnané, čím sa potvrdilo, že použitie gramatikou riadeného genetického programovania je pre tento problém najvhodnejšie.

Pôvodný algoritmus riešenia rozvrhovania práce s dodanými hotovými genetickými operátormi a základom výpočtu fitness funkcie bol rozšírený o počítanie kolízií[1], čím sa docielilo optimálnejších riešení problému z praktického hľadiska. Týmto je zabezpečené zníženie počtu problémov pri nasadzovaní algoritmu do reálneho produkčného prostredia.

Algoritmus riadenia a rozvrhovania práce bol následne otestovaný 60 navrhnutými testami inšpirovanými reálnymi dátami, pričom sme pri každom teste porovnali čas potrebný na úspešné vykonanie úloh tak ako to navrhol algoritmus, s časom tak ako by to navrhol človek. Algoritmus dosahuje vysokú úspešnosť hlavne v ťažších prípadoch. Pri jednoduchších prípadoch je porovnateľný s človekom. Celková dosiahnutá úspešnosť algoritmu bola 123%.

Následne bol algoritmus podrobený testu na vygenerovaných úlohách, kedy sa jednalo o väčšie problémy s vyšším počtom úloh. Výsledkom bolo porovnanie nastavení pravdepodobností jednotlivých genetických operátorov a určenie najvhodnejších kombinácií vzhľadom na veľkosť problému.

Pre zníženie výpočtovej náročnosti boli časti algoritmu paralelizované a účinok paralelizácie opäť zmeraný. Priemerné zníženie náročnosti algorimu bolo pri veľkosti populácie 200 jedincov zhruba 18% pri behu na danom testovacom zariadení.

Do algoritmu bola zapracovaná ďalšia heuristika vo forme prioritných pravidiel. Použitie jednotlivých pravidiel bolo podrobené testom a úspešnosť každého pravidla bola samostatne vyhodnotená na reálnych ako aj syntetických testoch. Opäť boli nájdené optimálne parametre pre použitie na menších a väčších problémoch. Rovnako bol porovnaný aj samotný priebeh iterácie v priebehu cyklov evolúcie.

[1] Tomuto sa podrobne venuje [27].

LITERATÚRA

[1] MOON, I.; LEE, J. Genetic algorithm application to the job shop scheduling problem with alternative routings. *Brain Korea 21 Logistics Team*, 2000.

[2] JONES, A.; RABELO, L. C.; SHARAWI, A. T. Survey of job shop scheduling techniques. *Wiley Encyclopedia of Electrical and Electronics Engineering*, 1999.

[3] BEHAM, A.; WINKLER, S.; WAGNER, S.; AFFENZELLER, M. A genetic programming approach to solve scheduling problems with parallel simulation. In: *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on.* IEEE, 2008. str. 1–5.

[4] CANBOLAT, Y. B.; GUNDOGAR, E. Fuzzy priority rule for job shop scheduling In: *Journal of Intelligent Manufacturing*, August 2004, vol. 5, str. 527–533.

[5] KAMRUL HASAN, S. M.; SARKER, R.; CORNFORTH, D. GA with priority rules for solving job-shop scheduling problems. In: *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on.* IEEE, 2008. str. 1913-1920.

[6] MACH, M. *Evolučné algoritmy: prvky a princípy.* Elfa, Košice, 2009. 250 strán. ISBN 978-80-8086-123-0.

[7] PELIKAN, M. Evolučné algoritmy. *Umelá inteligencia a kognitívna veta I*, 2009, str. 335–353.

[8] STOCKWELL, D. R. B.; PETERS D. G. The GARP modelling system: Problems and solutions to automated spatial prediction. *International Journal of Geographic Information Systems 13*, 1999, str. 143–158.

[9] MAIMON, O.; BRAHA, D. A genetic algorithm approach to scheduling PCBs on a single machine. *International Journal of Production Research*, 1998, 36.3: str. 761–784.

[10] PECINOVSKÝ, R. *Myslíme objektově v jazyku Java - kompletní učebnice pro začátečníky*, 2. aktualizované a rozšírené vydanie. Grada Publishing a.s., 2009. 570 strán.

[11] DAVIS, W.; JONES, A. A real-time production sheduler for a stochastic manufacturing environment. *International Journal of Computer Integrated Manufacturing*, 1988, str. 101–112.

[12] SHAPIRO J. A survey of Lagrangian techniques for discrete optimization. *Annals of Discrete Mathematics*, 1979, 5.1979: str. 113–138.

[13] AGIN, N. Optimum seeking with branch and bound. *Management Science*, 1966, 13.4: str. 176–185.

[14] LAWLER, E. L.; WOOD, D. E. Branch and bound methods: a survey. *Operations research*, 1966, 14.4: str. 699–719.

[15] FOX, M. S. *Constraint-directed search*. 1983. PhD Thesis. Carnegie-Mellon University.

[16] SHEN, W.; WANG, L.; HAO, Q. Agent-based distributed manufacturing process planning and scheduling: a state-of-the-art survey. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 2006, 36.4: str. 563–577.

[17] RABELO, L. C. A hybrid artificial neural networks and knowledge-based expert systems approach to flexible manufacturing system scheduling. 1990. PhD Thesis. University of Missoury-Rolla.

[18] PORTO, S. C.; RIBEIRO, C. C. A tabu search approach to task scheduling on heterogeneous processors under precedence constraints. *International Journal of high speed computing*, 1995, 7.1: str. 45–71.

[19] GENDREAU, M.; POTVIN, J. Y. Tabu search. *Search methodologies*. Springer US, 2005. str. 165–186.

[20] BUSETTI, F. Simulated annealing overview, 2003.

[21] KEANE, M. A. *Genetic programming IV: Routine human-competitive machine intelligence*. Springer, 2005.

[22] KOZA, J. R. Survey of genetic algorithms and genetic programming. In: *WESCON/'95. Conference record.'Microelectronics Communications Technology Producing Quality Products Mobile and Portable Power Emerging Technologies'*. IEEE, 1995. p. 589.

[23] GRUAU, F. On using syntactic constraints with genetic programming. In: *Advances in genetic programming*. MIT Press, 1996. str. 377–394.

[24] HOAI, N. X.; MCKAY, R. I.; ESSAM, D. Some experimental results with tree adjunct grammar guided genetic programming. In: *Genetic Programming*. Springer Berlin Heidelberg, 2002. str. 228–237.

[25] MCKAY, R. I. Grammar-based genetic programming: a survey. *Genetic Programming and Evolvable Machines*, 2010, 11.3–4: str. 365–396.

[26] VINOD, V.; SRIDHARAN, R. Simulation modeling and analysis of due-date assignment methods and scheduling decision rules in a dynamic job shop production system. *International Journal of Production Economics*, 2011, 129.1: 127-146.

[27] JURČÍK, M. *Modelování elastických a neelastických kolizí*. Brno, 2013. Diplomová práca, Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikácií.

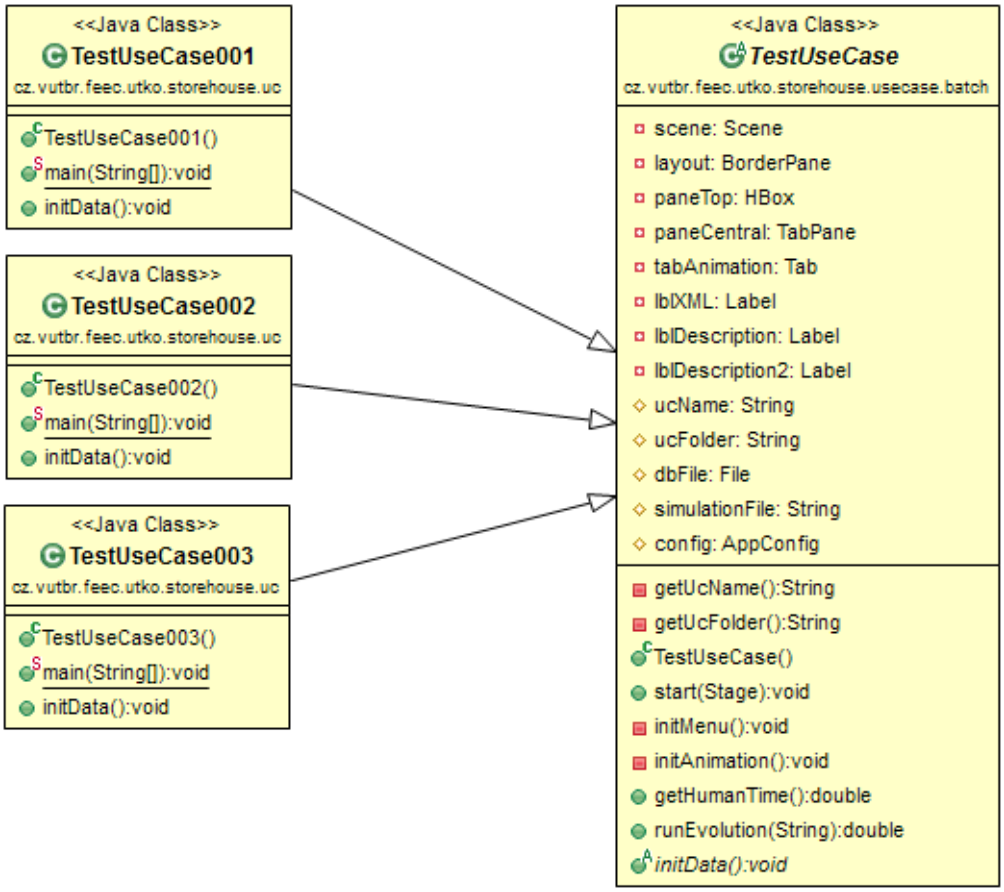
ZOZNAM SYMBOLOV, VELIČÍN A SKRATIEK

BGA	Breeder Genetic Algorithm
COVERT	Cost Over Time
CR	Critical Ratio
EDD	Earliest Due Date
FCFS	First Come, First Served
GR	Gap Reduction
JVM	Java Virtual Machine
MINSOP	Minimum Slack Time Per Operation
OOP	Objektovo orientované programovanie
PR	Partial Reordering
RS	Restricted Swappings
SOMA	Self-Organizing Migrating Algorithm
SPT	Shortest Processing Time
UML	Unified Modeling Language
XML	eXtensible Markup Language

ZOZNAM PRÍLOH

A UML DIAGRAMY

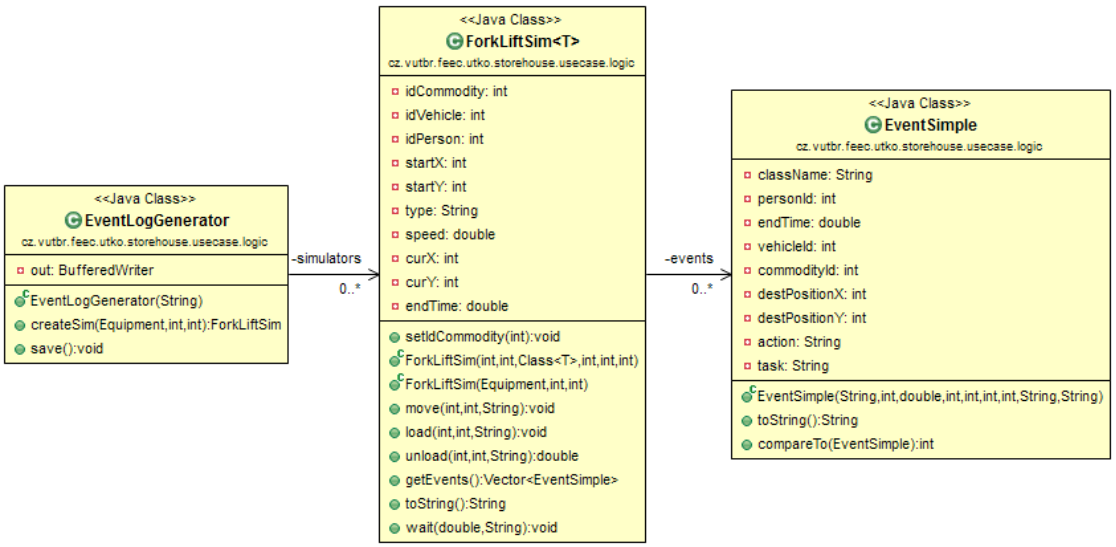
A.1 Sada testov



Obr. A.1: UML diagram sady testov, prvé 3 testy.

Všetkých 60 reálnych testov je navrhnutých ako samostatné triedy, ktoré dedia od abstraktnej triedy `TestUseCase`. V tejto sú definované komponenty užívateľského rozhrania, v triedach jednotlivých testoch dochádza iba k inicializácií testovacích dát, teda každý test je možné spustiť samostatne s podporou užívateľského rozhrania.

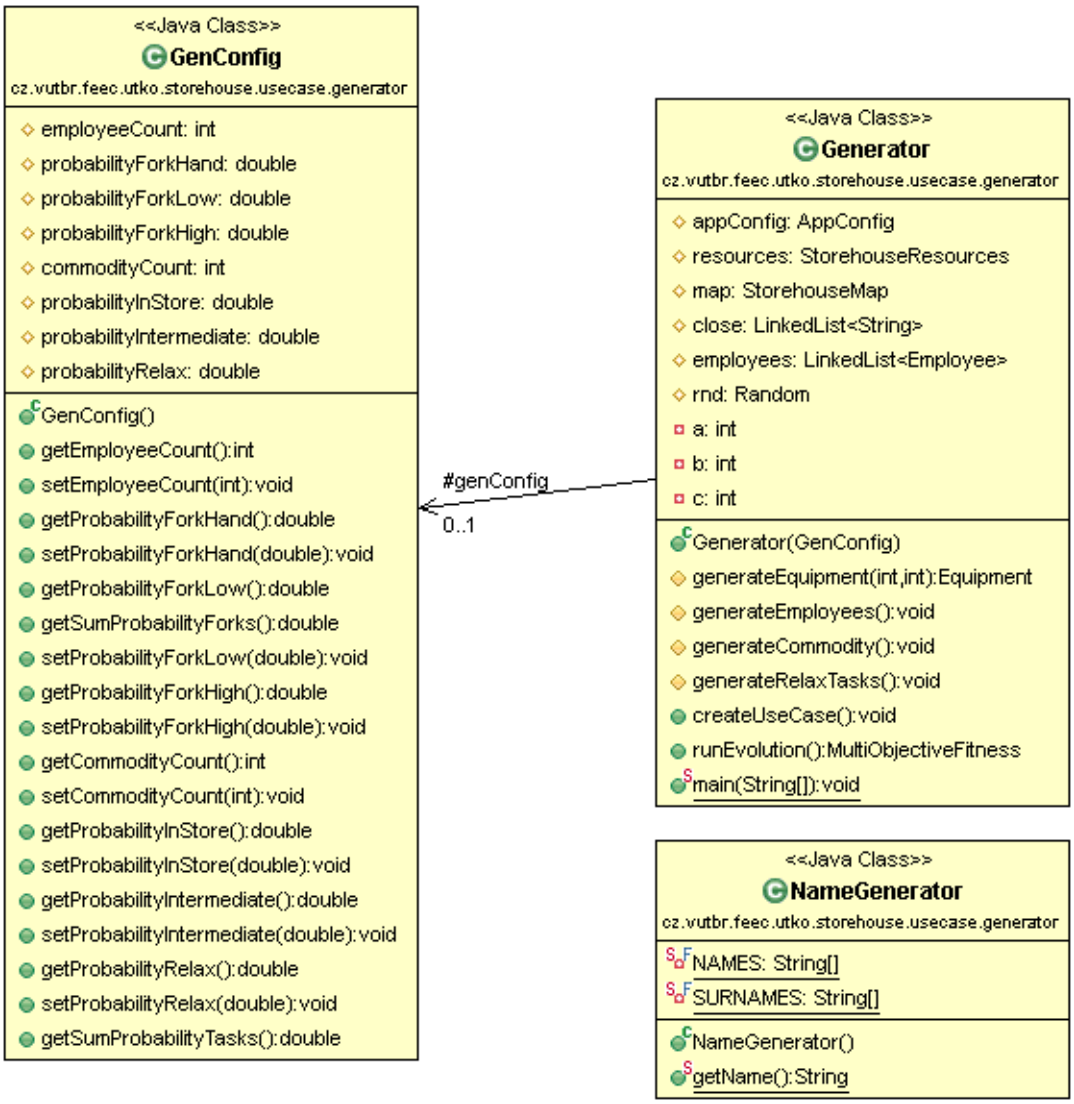
A.2 Logika testu



Obr. A.2: Zobrazenie vzťahov medzi objektami simulácie.

Jadrom logiky reálnych testov je trieda `ForkLiftSim`, prostredníctvom ktorej je možné jednoducho definovať jednotlivé udalosti `EventSimple`. Triedy pracujú s objektom typu `EventLogGenerator`, pomocou ktorého dôjde k uloženiu zoznamu udalostí do formátu XML. Objekt typu `ForkLiftSim` sa vytvára pre každý objekt vozíku v sklade.

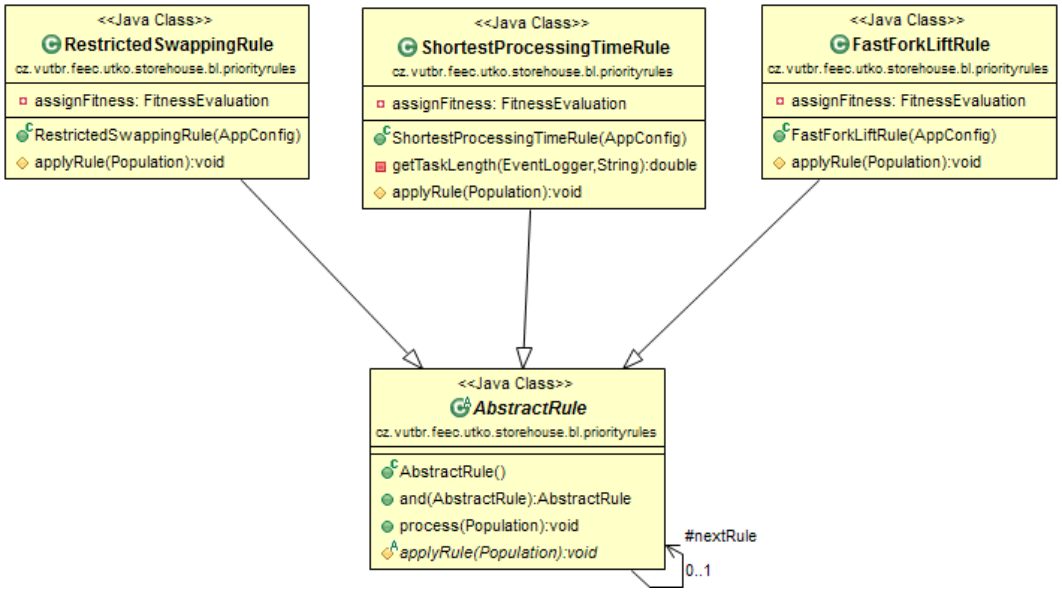
A.3 Generátor testov



Obr. A.3: Štruktúra generátoru testov.

Trieda `GenConfig` obsahuje všetky nastaviteľné parametri generátoru, objekt tohto typu je pradávaný samoznému generátoru v konštruktore. Generátor ďalej využíva statickú triedu `NameGenerator`, ktorá slúži pre generovanie náhodných mien pracovníkov.

A.4 Prioritné pravidlá



Obr. A.4: Štruktúra pravidiel.

Štruktúra prioritných pravidiel je inšpirovaná návrhovým vzorom *Specification* pre budúce jednoduché rozširovanie a naväzovanie pravidiel. Je možné tak jednoducho vytvoriť pravidlo, ktoré bude pozostávať už z existujúcich pravidiel, prípadne ich nejak rozširovať.