VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

VYUŽITÍ SIMULAČNÍHO MODELU NA VÝVOJ AUTOMATICKÉHO ALGORITMU PRO TVORBU ROUTOVACÍ TABULKY A OHODNOCENÍ CESTY V DOPRAVNÍKOVÉM SYSTÉMU

USE OF A SIMULATION MODEL FOR THE DEVELOPMENT OF AN AUTOMATIC ALGORITHM FOR CREATING A ROUTING TABLE AND PATH EVALUATION IN A CONVEYOR SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Dominika Weyrová**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Ivana Hromková, Ph.D.**
SUPERVISOR

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav automatizace a informatiky
Studentka:	**Bc. Dominika Weyrová**
Studijní program:	Strojní inženýrství
Studijní obor:	Aplikovaná informatika a řízení
Vedoucí práce:	**Ing. Ivana Hromková, Ph.D.**
Akademický rok:	2020/21

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Využití simulačního modelu na vývoj automatického algoritmu pro tvorbu routovací tabulky a ohodnocení cesty v dopravníkovém systému

Stručná charakteristika problematiky úkolu:

Pro práci bude nutné nastudování problematiky modelování a simulace výroby a rovněž nastudování problematiky prohledávání stavového prostoru (metody prohledávání, routovací tabulka, kriteria prohledávání, atp. – vypracování rešerše). Vlastní realizace bude v software Tecnomatix PlantSimulation. Jeden z úkolů pak bude vývoj algoritmu pro automatickou tvorbu routovací tabulky, včetně optimalizace pro ohodnocení tras dopravníkového systému.

Cíle diplomové práce:

Rešerše problematiky modelování a simulace výroby a problematiky prohledávání stavového prostoru.
Analýza dostupných metod pro prohledávání stavového prostoru.
Tvorba simulačního modelu dopravníkového systému v software Tecnomatix PlantSimulation.
Vývoj algoritmu a jeho otestování pro automatickou tvorbu routovací tabulky pro směrování a ohodnocení tras dle statických kriterií pomocí simulačního modelu dopravníkového systému.
Návrh principu algoritmu a jeho optimalizace pro ohodnocení tras dopravníkového systému se zahrnutím dynamických kriterií.

Seznam doporučené literatury:

PROUD, John F. Master scheduling: a practical guide to competitive manufacturing. 3rd ed. Hoboken: WILEY, John, 2007, xxviii, p. 657, ISBN 978-0-471-75727-6.

ADNAN, A., AMIT, P. "4. Algorithms on Graphs". Algorithms for Interviews 2010, ISBN 1453792996.

SKIENA, Steven. The Algorithm Design Manual. Springer, p. 480, 2008, doi:10.1007/978-1-848-0-070-4_4.

BRABAZONB, A., O'NEILL, M., McGARRAGHY, S.,Natural Computing Algorithms. Springer-Verlag Berlin Heidelberg, 2015, ISBN 978-3-662-43630-1.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2020/21

V Brně, dne

L. S.

doc. Ing. Radomil Matoušek, Ph.D. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

ABSTRAKT

Diplomová práce se zabývá využitím simulačního modelu na vývoj automatického algoritmu pro tvorbu routovací tabulky a ohodnocení cesty v dopravníkovém systému. Zahrnuje rešerši problematiky modelování a simulace výroby a problematiku prohledávání stavového prostoru s analýzou dostupných prohledávacích metod. Simulační model dopravníkového systému je vytvořen v sofware Tecnomatix Plant Simulation, kde je následně vyvíjen a testován algoritmus pro automatickou tvorbu routovací tabulky pro směrování a ohodnocení tras dle statických kriterií. V práci je uveden návrh principu algoritmu pro ohodnocení tras dopravníkového systému se zahrnutím dynamických kriterií a jeho optimalizace.

ABSTRACT

The diploma thesis deals with the use of a simulation model for the development of an automatic algorithm for the creation of a routing table and route evaluation in a transport system. It includes a search of modeling and simulation issues and state-space search issues with an analysis of available search methods. The simulation model of the transport system is created in the software Tecnomatix Plant Simulation, where an algorithm for automatic creation of routing tables for routing and evaluation of routes according to static criteria is subsequently developed and tested. The work presents a proposal for the principle of the algorithm for evaluating the routes of the transport system, including dynamic criteria and their optimization.

KLÍČOVÁ SLOVA

Model, systém, výrobní systém, dopravníkový systém, simulace, optimalizace, stavový prostor, prohledávaní stavového prostoru, složitost algoritmu, prohledávací algoritmy, směrování, routovací tabulka, Plant Simulation.

.

KEYWORDS

Model, system, production system, conveyor system, simulation, optimization, state-space, state-space search, algorithm complexity, search algorithms, routing, routing table, Plant Simulation.

ÚSTAV AUTOMATIZACE A INFORMATIKY

2021

BIBLIOGRAFICKÁ CITACE

WEYROVÁ, Dominika. Využití simulačního modelu na vývoj automatického algoritmu pro tvorbu routovací tabulky a ohodnocení cesty v dopravníkovém systému. Brno, 2021. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/135779. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky. Vedoucí práce Ivana Hromková.

PODĚKOVÁNÍ

Tímto chci poděkovat své vedoucí paní Ing. Ivaně Hromkové, Ph.D. za ochotu a vedení práce. Dále rodině a přátelům za podporu.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že, že tato práce je mým původním dílem, vypracoval jsem ji samostatně pod vedením vedoucího práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury.

Jako autor uvedené práce dále prohlašuji, že v souvislosti s vytvořením této práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následku porušení ustanovení § 11 a následujících autorského zákona c. 121/2000 Sb., včetně možných trestně právních důsledků.

V Brně dne 22. 5. 2021 ……

Dominika Weyrová

OBSAH

1 ÚVOD

Simulace se používá v mnoha souvislostech. Ekonomové mohou simulovat vývoj finančních toků a sociální simulace aplikuje výpočetní metody na studium témat ve společenských vědách. Existuje také spousta zdravotních simulací, simulace případových studií, simulátory určené pro výcvik a přípravu na různé pracovní pozice, simulátory připravenosti na katastrofy a mnoho dalších. V neposlední řadě je také simulace v zábavním průmyslu, ať už počítačové hry, kde se simuluje určité prostředí a děj, grafické efekty ve filmech, nebo celkově virtuální realita. Ve strojírenských systémech je simulace nedílnou součástí, kdy je během různých simulovaných procesů možné zaznamenávat matematické výpočty a provádět počítačový výzkum.

Rešeršní část práce začíná popisem modelu. V práci byl model a následná simulace vytvořen pomocí programu Tecnomatix Plant Simulation, což je efektivní nástroj pro dynamickou simulaci diskrétních událostí. V programu se vytvoří tzv. *digitální dvojče*, model reálného výrobního systému, na němž je možné simulovat jeho chování, komunikaci mezi jednotlivými složkami apod.

V další části je obecně popsán systém, jeho modelování a také výrobní systém a typy dopravníků. Dále jsou popsány typy simulace, její výhody a nevýhody a poté výklad optimalizace.

Následuje analýza dostupných metod pro prohledávání, protože některé simulační nástroje byly rozšířeny o prohledávající algoritmy. Modelované výrobní systémy jsou orientované grafy, jejichž uzly jsou pracovní stanice, které konají nějakou akci. Hrany mezi uzly jsou tvořeny dopravníky, které slouží k přemístění subjektů mezi stanicemi.

V praktické části jsou nejprve popsány implementace automatických směrování v simulačních nástrojích, popis vlastností Plant Simulation a potom vlastní vývoj algoritmu na vytvoření routovací tabulky a algoritmu pro směrování ve výrobě. Pro vytvoření routovací tabulky bylo využito automatického směrování Plant Simulation. Podle tabulky pak algoritmus uložený ve zdroji posílá materiál po trase dané pro určitý technologický potup, nejrychlejší trasu a nejrychlejší trasu pro určitou stanici. Na závěr je uveden návrh směrování se zahrnutím dynamických vlastností systému.

2 MODELOVÁNÍ A SIMULACE VÝROBY

Model je definován jako fyzikální nebo abstraktní, tj. matematické, slovní i grafické zobrazení reálného systému. Slovo model je dnes pojem využívaný v mnoha okruzích, a to v běžném životě, technice a vědě či třeba v umění. Tento pojem širokého významu lišícího se v různých oborech souvisí s řešením problémů různého typu, a to od nejběžnějších lidských činností a zájmů po složité vědecké a technické problémy. Pro všechny tyto obory je společný význam cílené přiřazení přídavného objektu k původnímu objektu zájmu a následná činnost s tímto objektem, modelem, z nějž lze získat potřebné informace. [1, 3]

2.1 Model obecně

Máme-li reálný systém jakožto soubor prvků s jejich vzájemnými vazbami, modelací pak můžeme získat jeho napodobeninu s nižšími náklady na realizaci, přičemž z modelu lze pak efektivně získávat informace. Před samostatnou modelací je nutné kompletní porozumění daného systému, znát vlastnosti jak celého systému, tak i jeho jednotlivých komponent a stejně tak i jeho účel a funkčnost. Pro různé systémy a jejich reprezentace jsou odlišné i metody modelování. Způsob modelování se tedy volí s ohledem na rozpočtové a časové možnosti, provedení musí být praktické a ekonomické. Model by měl být dostatečně zřejmý, komplexní a přesný, tedy spolehlivý při demonstraci funkcí systému, jeho analýze a předpovídání budoucího chování systému reálného. Modely lze klasifikovat do čtyř hlavních kategorií: fyzické modely, grafické modely, matematické modely a počítačové modely. [2]

Fyzický model je snadno pochopitelné a hmatatelné znázornění reálného objektu nebo systému, prototyp, který může být vyroben v měřítku 1:1 či jakémkoliv jiném. Je to přímá hmotná reprezentace systému, která demonstruje jeho strukturu, prvky a funkce. Nevýhodou fyzického modelu jsou jeho obrovské náklady na výrobu a příliš náročná realizace složitých systémů. [2]

Grafický model je abstrakce skutečného produktu či procesu pomocí grafických nástrojů počínaje kresbou na papíře, pokračujíce přes technické výkresy a konče u obrázků a filmů zobrazujících daný systém. Znázornit lze rozložení systému, vývojové a blokové diagramy, síťové diagramy, procesní a provozní mapy atd. Absence dynamické funkčnosti v grafických reprezentacích je vynahrazena animací a videoklipy, tato skutečnost však výrazně omezuje univerzálnost znázornění, systém je zjednodušený. Z tohoto důvodu se stala většina grafických nástrojů součástí či prerekvizitou ostatních typů modelů. [2]

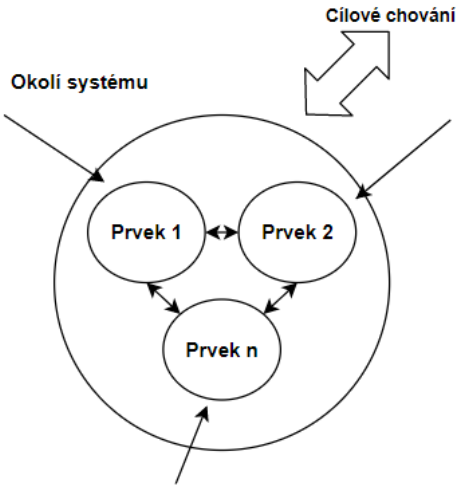
Matematický model znázorňuje chování systému pomocí matematických rovnic nebo vzorců. Model symbolicky vyobrazuje funkčnost systému, řídicích proměnných, odezvy a omezující podmínky. Formuluje návrh systému řešitelného pomocí grafických a výpočetních metod užitím matematických rovnic, pravděpodobnostních modelů a statistických metod. [2]

Počítačový model reprezentuje systém číselně, graficky a logicky, počítač je schopen rychlých složitých výpočtů s vysokou přesností. Reálný systém a jeho procesy jsou zobrazeny virtuálně. Podobně jako u matematických modelů lze modelovat složité matematické operace, řídicí systémy, mechaniku tekutin, počítačové algoritmy atd. Díky softwarovým nástrojům je umožněno vyvíjet statické i dynamické průmyslové procesy. Počítačový model umožňuje rychlé změny, snadné testování, hodnocení výkonu a optimalizaci a nabízí vyšší efektivnost a realističnost oproti matematickým modelům. Nevýhodou počítačového modelu je limitní výkon samotného počítače a možná omezení softwaru a jeho kompatibility. [2]

2.2 Systém

Systém je definován jako samostatná sada komponent, které mezi sebou komunikují a vzájemně na sebe působí tak, aby bylo dosaženo požadovaného logického výsledku. V praxi definice závisí na typu daného systému, jednotlivé subjekty můžou být lidé nebo třeba stroje atd. Záleží na cílech konkrétní studie. Množina entit, která tvoří systém pro jednu určitou studii, může být pouze podmnožinou dalšího systémového celku v určitém čase ve vztahu k cílům studie. [4]

Složitý otevřený systém musí být především vymezený vůči svému okolí, musí mít specifickou vnitřní strukturu (tj. jedinečné vazby mezi prvky). Dalším klíčovým znakem je jeho cílové chování. [5]



Obrázek 1 Schéma systému podle [5]

2.2.1 Modelování systému

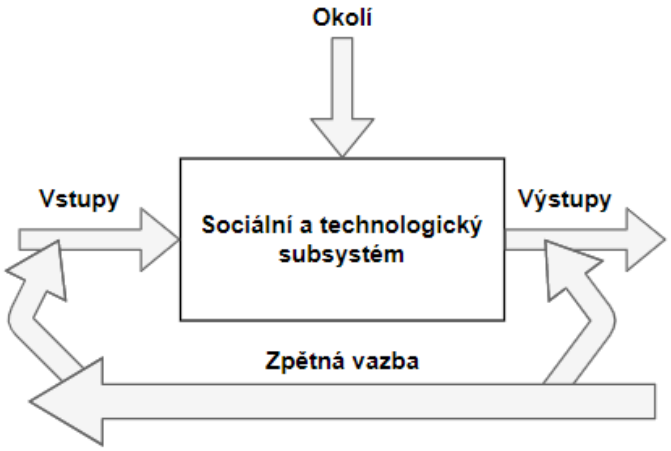
Proces modelování systémů je rozdělen do tří etap, nejprve je nutné zformulovat účelový a zjednodušený popis zkoumaného systému, následně zapsat abstraktní model formou programu a poté již lze z reprezentací modelu experimentovat.

Pro správnou charakteristiku je nutné vymezit základní časové pojmy. *Reálný čas,* tj. čas, ve kterém probíhá děj ve skutečném systému, dále *modelový čas,* který tvoří časovou osu modelu a může být reálný, zrychlený či zpomalený a čas potřebný na výpočet programu, strojový čas. Strojový čas nesouvisí s hodnotami času modelového, ale závisí na složitosti systému a vlastnostech programu. [6]

2.2.2 Výrobní systém

Výrobní systém zahrnuje jak technologické prvky (stroje a nástroje), tak organizační chování (dělba práce a tok informací). Výrobní systém definujeme jako soubor technik průmyslového inženýrství, nástrojů managmentu a metod „štíhlé“ výroby, při nichž se snažíme dosáhnout podnikatelských cílů firmy. Je to systém vzájemného propojení výrobních a pomocných prostředků s výrobními silami a vstupním materiálem. Cílíme na vysokou produktivitu a pružnost výroby a nejvýhodnější poměr mezi časem výroby a využitím zařízení. [5]

Výrobní systém realizuje výrobu – proces přeměny a přizpůsobování zdrojů vstupujících do výrobního systému a směřující k tvorbě hmotných statků nebo služeb. Architektura výrobního systému sestávající z hardwaru a softwaru určuje výkonnost celého systému, přičemž správná integrace jeho součástí (materiálů, lidí, informací atd.) rozhodne o konkurenčních výhodách daného výrobního podniku. [5]



Obrázek 2: Schéma výrobního systému podle [5]

Na vstupu do systému může být například materiál, nebo služby pro procesy, fyzický kapitál jako jsou stroje, zařízení, nástroje apod., dále informace technického nebo procesního charakteru, v neposlední řadě také lidská pracovní síla.

Pro řízení výroby je důležitou vazbou poptávka. Okolí podniku lze rozdělit podle různých hledisek, mikrookolí je tvořeno např. zákazníky, konkurenty, dodavateli atd., do makrookolí potom patří legislativa, činnost bank, ekonomické, politické, technologické, ekologické, kulturní a sociální vazby. [5]

2.2.3 Dopravníkový systém

Dopravníkový systém je rychlé a efektivní mechanické manipulační zařízení pro automatickou přepravu nákladu a materiálu v dané oblasti. Tento systém mimo jiné minimalizuje lidské chyby, snižuje rizika na pracovišti a snižuje náklady na pracovní sílu.

Systémy jsou užitečné při přemisťování objemných nebo těžkých předmětů z jednoho bodu do druhého. Dopravníkový systém může k přepravě předmětů používat pás, kola, válečky nebo řetěz. [7] Dopravníky lze rozdělit na dopravníky s nebo bez tažného elementu a na dopravníky vibrační dopravy. [8]

Dopravníky s tažným elementem:

- Pásové.
- Článkové.
- Hrnoucí.
- Korečkové elevátory.
- Závěsné.

Dopravníky bez tažného elementu:

- Šnekové.
- Dopravní skluzy.
- Válečkové tratě.

Vibrační doprava:

- Pohyblivé dopravní žlaby.
- Třasadla.
- Dopravníky s mikrovrhem. [8]

Pásové dopravníky slouží ke kontinuální dopravě sypkých látek i kusového zboží, a to převážně ve vodorovném, případně mírně šikmém směru. Podle typu materiálu a druhu pásu se volí maximální úhel stoupání. Pásové dopravníky patří k nejrozšířenějším prostředkům dopravy sypkých látek, a to díky vysokém dopravním výkonu a velké dopravní vzdálenosti. Další výhody jsou jejich jednoduchá údržba, malá spotřeba energie, možnost nakládání a vykládání materiálu v libovolném místě. Pásové dopravníky se dělí podle tažného elementu (dopravního pásu), tvaru dopravníku a podle provedení nosné konstrukce. [8]

Charakteristickým znakem **článkových dopravníků** jsou jejich tažné prvky. Tyto dopravníky jsou tvořeny řetězy s nosnými orgány, kdy podle druhu podpěry je umožněno buď valivé nebo kluzné uložení. Článkové dopravníky nacházejí své uplatnění převážně při vodorovné nebo šikmé dopravě těžkých, hrubozrnných, ostrohranných, abrazivních a horkých materiálů, tam, kde nelze použít dopravníků pásových. [8]

Hrnoucí dopravníky jsou určeny pro dopravu sypkých a nelepivých hmot. Materiál je posouván v plechovém žlabu unašeči připevněnými k tažným řetězům. Pro přepravu sypkých látek ve svislém až strmém směru se využívají **korečkové elevátory**, u kterých je možné dosáhnout vysokého dopravního výkonu i výšky.

Plechové nádoby, korečky, jsou pevně připojené k tažnému orgánu, který může být tvořen řetězy nebo gumovým dopravní pásem. [8]

Pro spojitou, výjimečně přerušovanou dopravu slouží **závěsné dopravníky** tvořené soustavou vozíků, které pojíždí po visuté dráze, která je tažena řetězy nebo lanem. Dopravují především kusový materiál a výjimečně sypké hmoty. Závěsné dopravníky uvolňují podlahové plochy, umožňují prostorové vedení dráhy a jsou ekonomicky nepříliš nákladné díky nízké energetické náročnosti, nízkým pořizovacím nákladům, jednoduché obsluze a údržbě a v mezioperační dopravě umožňují automatizaci a vytváření pohyblivých meziskladů. [8]

Šnekové dopravníky patřící k nejstarším dopravníkům jsou velmi jednoduché, skládají se pouze ze tří hlavních částí, a to z dopravního žlabu, šneku a poháněcí jednotky. Tímto dopravníkem je možno dopravovat sypké látky ve vodorovném či mírně nakloněném směru. Materiál je posouván žlabem díky otáčejícímu šneku. Třecí síly mezi šnekem a materiálem musí být nižší, než mezi materiálem a žlabem, díky tomu dochází k transportu a promíchávání materiálu. [8]

Konstrukce **dopravních skluzů** je jednoduchá s minimálními investičními i provozními náklady, protože k dopravě materiálu je využito pouze účinků tíhy při pohybu na nakloněné rovině. Tvar dráhy může být přímý, s oblouky nebo ve tvaru šroubovice, podle typu se pak volí materiál dopravníku, nejčastěji dřevo, plasty, ocelové plechy, beton apod. Skluzy se snadno kombinují s ostatními typy dopravníků za použití jednoduchých výhybek. [8]

Dopravním orgánem **válečkových,** případně **kladičkových dopravníků** jsou válečky nebo kladičky otočně uložené v rámu, kterými jsou tvořeny tratě potřebných délek. Válečkové a kladičkové dopravníky jsou určeny k přepravě kusových břemen, a to především v mezioperační dopravě ve skladech. Konstrukce válečkových dopravníků je tvořena rámem a v něm uloženými normalizovanými válečky, jejichž vzájemná vzdálenost se volí tak, aby předmět spočíval nejméně na dvou válečcích. Tratě můžou být dvou druhů, gravitační nebo poháněné. U tratí gravitačních se dopravované předměty pohybují vlivem tíhové síly nebo tlačením po nepoháněných válečcích. V poháněných tratích se předměty pohybují vlivem nuceného otáčení válečků. [8]

U **pohyblivých dopravních žlabů** se dopravní žlab uložený na kuličkách nebo válečcích pohybuje přímočaře a vratně ve směru své osy. Žlab je poháněn nesymetrickým klikovým mechanismem. Dopravní žlaby jsou vhodné pro dopravu zrnitých a kusovitých materiál, a to i horkých a silně abrazivních a doprava se uskutečňuje ve vodorovném i šikmém směru. [8]

Třasadla patří mezi impulsní dopravníky. Dopravní žlab třasadla je uložen na kyvných vzpěrách nebo listových pružinách a poháněn klikovým mechanismem, jehož osa je kolmá k osám vzpěrných ramen.

Vzpěry jsou postaveny šikmo a s daným odklonem od vertikální roviny, kmitavý pohyb žlabu má tedy vodorovnou i svislou složku. [8]

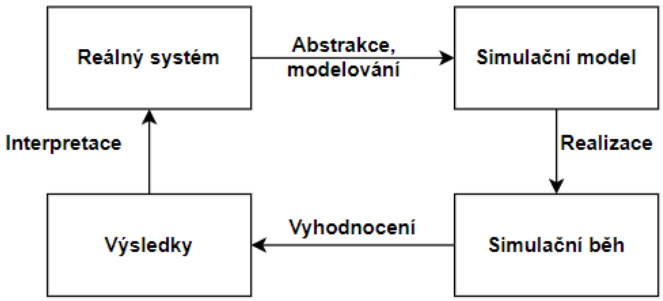
Dopravníky s mikrovrhem jsou nejrozšířenější v oblasti vibrační dopravy jsou vhodné pro odběr sypkých látek ze zásobníku a je možné je použít i na delší vzdálenosti nebo jako prosévací zařízení. Pro jejich pohon se využívá elektromagnetismus, mechanický dynamický budič nebo klikový mechanismus. V oblasti nižších frekvencí s pohonem klikovým mechanismem se konstrukčně podobají třasadlům. Jejich výhoda oproti dopravním žlabům a třasadlům spočívá v menším opotřebení žlabu a nižších potřebných příkonech hnacích motorů, jsou ale více citlivé na dopravu i mírně lepivých materiálů. [8]

2.3 Simulace

Simulace je podle Bangsowa reprodukce modelu reálného systému včetně jeho dynamických procesů. Jejím cílem je dosáhnout přenositelných závěrů pro děje modelovaného systému. Je to příprava, implementace a vyhodnocení konkrétních experimentů se simulačním modelem. [9]

Simulaci můžeme členit podle různých hledisek. Podle způsobu modelování procesů ji dělíme na **simulaci diskrétních** nebo **spojitých** procesů. Při simulaci diskrétních procesů mohou proměnné nabývat pouze předem stanovených hodnot, simulace spojitých procesů pracuje s proměnnými, které nabývají všech hodnot ze stanoveného intervalu. Podle přístupu k zahrnutí **faktoru času** rozlišujeme simulaci na **statickou** a **dynamickou**. Při statické simulaci generujeme stav systému v konkrétním časovém okamžiku, při dynamické simulaci modelujeme jeho vývoj v čase. Podle způsobu práce s náhodnými vlivy pak simulace může být **deterministická**, kdy neuvažujeme náhodné jevy, a **stochastická**, ve které modelujeme náhodné faktory zahrnutím náhodných proměnných do simulačního modelu. [10]

Pro rozvrhování výroby v čase a disponibilních kapacitách se používá **simulační model hmotného toku ve výrobě**. V algoritmech simulačního modelu se nachází komplexní logistika výrobního procesu v simulované trajektorii veškerých zásob polotovarů i mezioperačních součástí a vnitropodnikové dopravy.[5]



Obrázek 3: Proces tvorby simulačního modelu podle [10]

Na obrázku 3 je uveden obecný postup simulačního modelu. Na modelu reálného systému jsou něm prováděny experimenty za účelem lepšího pochopení studovaného systému či posouzení různých variant jeho činnosti. [10]

Simulace je jednou z nejpoužívanějších technik výzkumu a managmentu. Oblasti použití simulace jsou např.:

- Návrh a analýza výrobních systémů.
- Hodnocení vojenských zbraňových systémů nebo jejich logistických požadavků.
- Stanovení hardwarových požadavků nebo protokolů pro komunikační sítě.
- Stanovení hardwarových a softwarových požadavků na počítačový systém.
- Navrhování a provozování dopravních systémů, jako jsou letiště, dálnice, přístavy a podchody.
- Hodnocení návrhů pro servisní organizace, jako jsou call centra, restaurace. rychlého občerstvení, nemocnice a pošty.
- Reengineering obchodních procesů.
- Analýza dodavatelských řetězců.
- Stanovení zásad objednávání pro inventární systém.
- Analýza těžebních operací. [4]

Výhody a nevýhody simulace

Vysoká konkurence v počítačovém průmyslu vedla k technologickým průlomům a hardwarové společnosti směřují k neustálé výrobě lepších produktů. Probíhá zvyšování výkonu, paměti a zlepšování všech možností včetně grafických. Díky tomuto vývoji v počítačovém průmyslu dochází i k vylepšování simulačního softwarového průmyslu. S růstem výkonnosti hardwaru vzniká také přesný, rychlejší a snadněji použitelný simulační software. Roste také počet podniků využívajících simulaci a spousta manažerů si uvědomuje výhody využití simulace pro více než jen jednorázové představení zařízení. Kvůli pokroku v softwaru tak manažeři začleňují simulaci v jejich každodenním provozu stále častěji. U většiny společností výhody používání simulace přesahují pouhé poskytnutí pohledu do budoucnosti. [11]

Výhody simulace:

1. **Možnost volby.** Díky simulaci lze získat výsledky testováním všech aspektů navrhované změny bez nutnosti přidělování prostředků. Simulace tedy umožňuje otestovat všechny návrhy, ať už vhodné či ne, aniž by docházelo k nákladným opravám nevhodných dopadů zvolené změny.
2. **Manipulace s časem.** Zkracování či prodlužování doby simulace umožňuje zrychlit či zpomalit jevy pro důkladné prošetření.
3. **Porozumění.** Ve chvíli, kdy je nutné pochopit, proč v reálném systému dochází k nějakým jevům, je možné rekonstruovat, přiblížit a prozkoumat děje a určit vysvětlení, kterého bychom nemohli u reálného systému dosáhnout, protože jej nemůžeme ovládat ani vidět jako celek.
4. **Zkoumání možností.** Jakmile je vytvořen platný simulační model, je možné prozkoumat nové provozní postupy nebo metody bez nákladů a narušení skutečného systému.

5. **Diagnostika problémů.** Ve složitém systému, kde je nemožné vzít v úvahu všechny interakce, ke kterým dochází v daný okamžik, simulace umožní lépe porozumět vztahům mezi proměnnými. Diagnostikuje problém a získá přehled o důležitosti určitých proměnných, díky čemuž lze pochopit jejich vliv na výkon celého systému.
6. **Identifikace omezení.** Díky simulaci lze nalézt úzká místa, tj. místa, která ovlivňují průtok celým systémem. Při analýze těchto míst je možné zjistit příčinu zpoždění v nedokončené výrobě, informacích, přenosu materiálu a ostatních procesů.
7. **Rozvoj porozumění.** Simulační studie umožňují lepší pochopení skutečného fungování systému než předpovědi a analýzy expertů.
8. **Vizualizace plánu.** Podle použitého softwaru je možné prohlížet operace z různých úhlů a úrovní zvětšení a trojrozměrně. Díky tomu lze detekovat konstrukční chyby, které nebyly patrné při dvojrozměrném modelu na papíře.
9. **Vytvoření konsenzu.** Pozorování konstrukčních změn při simulaci dopomůže k vytvoření objektivního názoru. Vybrány jsou návrhy a úpravy, které poskytly ty nejžádanější výsledky, ať už jde o zvýšenou výrobu nebo zkrácení čekací doby na servis. Simulace podává spolehlivé a přesvědčivé výsledky, které byly modelovány, testovány, ověřeny a vizuálně reprezentovány na rozdíl od případných názorů na návrh.
10. **Příprava na změny.** Je téměř jisté, že budoucnost přinese změny. Uvažování o budoucích možnostech je užitečné jak pro návrh nových systému, tak po redesign systémů stávajících.
11. **Náklady.** Typické náklady na simulační studii jsou podstatně nižší než 1 % z celkové částky vynakládané na implementaci designu nebo redesignu. Protože jsou tedy náklady na změnu nebo úpravu systému po instalaci tak vysoké, investice do simulace je rozumná. Časové náklady oproti experimentu s reálným systémem jsou také mnohonásobně nižší.
12. **Bezpečnost.** V některých případech není možné experimentovat přímo se systémem, a přitom je potřeba znát jednotlivé fáze jeho vývoje.
13. **Školení týmu.** Simulace může být také navržena tak, aby poskytla trénink členům operačního týmu, což je mnohem levnější a méně rušivé než učení na pracovišti.
14. **Specifikace požadavků.** V případě, že např. nejsou známy vhodné specifikace konkrétního typu stroje v komplexu, simulací různých vlastností stroje lze tyto požadavky stanovit. [11]

Nevýhody simulace:

1. **Model musí být vytvářen zkušeným expertem.** Schopnost vytvořit simulační model vyžaduje zkušenosti získávané časem.
2. **Nevšestrannost.** Model nemá obecnou platnost, simulační model lze využít jen pro systém, pro který byl modelován. Pro jiný je nutné vytvořit nový simulační model.
3. **Obtížně interpretovatelné výsledky.** Nelze zjistit závislost vstupních a výstupních dat. [11]

3 OPTIMALIZACE

Optimalizace je důležitým rozhodovacím nástrojem ve vědě a při analýze fyzikálních systémů. Při optimalizaci je nutno mít určitý cíl, tj. kvantitativní měřítko výkonu studovaného systému. Tímto cílem může být třeba zisk, využitý čas, potenciál energie nebo jakékoliv množství či kombinace dalších veličin, které je možné reprezentovat jedním číslem. Výsledek optimalizace závisí na charakteristikách systému, a to na známých i neznámých proměnných. Cílem je najít takové proměnné, při kterých bude systém dosahovat nejlepších možných výsledků. Proměnné jsou často nějakým způsobem omezené, jako třeba hustota hmoty nebo úroková sazba půjčky, které nemohou mít nikdy zápornou hodnotu. [12]

3.1 Optimalizační model

Vytvoření vhodného modelu je nejdůležitějším krokem procesu optimalizace. Při příliš velkém zjednodušení může být model nedostačující a nedokáže poskytnout potřebné informace, naopak u příliš složitého modelu bývá řešení velmi obtížné. Jakmile je vytvořen model, je na něm použit optimalizační algoritmus, který většinou vykonává počítač. Neexistuje žádný univerzální algoritmus pro všechna řešení, každý optimalizační problém má jeden a více vlastních algoritmů. Správná volba algoritmu určí rychlost řešení, při špatné volbě může dojít k obrovskému zpomalení, nebo také že řešení nebude nalezeno vůbec.

Po aplikaci algoritmu na model musíme být schopni rozpoznat, zda došlo k nalezení řešení. Ke kontrole optimality se často používají *podmínky optimality,* což jsou matematické výrazy, které určí, jestli je aktuální skupina proměnných skutečně řešením problému. V případě, že tyto podmínky nejsou splněny, mohou navíc poskytnout informace, jak aktuální řešení vylepšit. Model je vylepšován aplikací technik, jako je třeba *analýza citlivosti*, která ukazuje vliv změny v modelu a datech na řešení úlohy. Ke zlepšování a opravám modelu také přispívá vhodná interpretace řešení z hlediska dané aplikace. Po provedení změn se algoritmus znovu aplikuje a celý proces se opakuje. [12]

Matematický model

Obecnou strukturu matematického modelu lze vyjádřit následovně:

Najdi x pro:

$$\begin{array}{lll} & \text{Maximalizuj} & f(x) \\ (1) & \text{Při} & g_i(x) \leq gb_i, i = 1, \dots, m \\ & & h_j(x) = hb_j, j = 1, \dots, p \\ & & x \geq 0 \end{array}$$

Kde účelová funkce f je funkce jedné proměnné x a omezovací funkce g_i a h_j jsou obecné funkce proměnné (respektive neznámé, rozhodující veličiny, někdy také parametr funkce) x, která patří do oboru reálných čísel. [13]

Pravé strany podmínek gb_i a hb_j jsou obvykle konstanty deterministických problémů. Podmínka nezápornosti je nezbytná pro mnoho praktických problémů, jelikož spousta proměnných nemůže být záporná a pro mnoho typů řešení je nezápornost výchozí předpoklad. Místo omezení negativity může mít x horní a dolní hranici, dále může model obsahovat více proměnných ať už s omezením nebo bez. [13]
Za předpokladu, že $\bar{x}$ představuje množinu proměnných, kde $\bar{x} = x_{1,}x_2 \dots x_n$, pak lze výše uvedený model přepsat pro více proměnných následujícím způsobem:

(2)
$$\begin{aligned} &\text{Maximum} && f(\bar{x}) \\ &\text{Při} && g_i(\bar{x}) \leq gb_i, i = 1, \dots, m \\ & && h_j(\bar{x}) = hb_j, j = 1, \dots, p \\ & && \bar{x} \geq 0 \end{aligned}$$

Obecné vlastnosti matematického modelu jsou:

- Množství zdrojů, které je většinou reprezentováno pravou stranou omezující podmínky, je popsáno parametrem.
- Zdroje využité pro nějakou aktivitu, jako je třeba výroba nebo poskytnutí služby, jsou obvykle zastoupeny rozhodovací proměnnou.
- Existuje řada alternativních způsobů, jakými lze zdroje použít.
- Každá aktivita využívající prostředky míří ke splnění stanoveného cíle.
- Alokace zdrojů je obvykle limitována několika omezeními. [13]

Za předpokladu, že $g_i(\bar{x})$ a $f(\bar{x})$ v modelu (2) jsou lineární funkce a je možné je vyjádřit tímto způsobem:

$$f(\bar{x}) = c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

a

(3)
$$\begin{aligned} g_1(\bar{x}) &= a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \leq gb_1 \\ g_2(\bar{x}) &= a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \leq gb_2 \\ &\vdots \end{aligned}$$

Při omezení $g_1(\bar{x})$ je a_{11} potřebný zdroj z gb_1 pro každou jednotku aktivity x_1, stejně jako je a_{12} potřebný z gb_1 pro každou jednotku aktivity x_2 atd. V účelové funkci $f(\bar{x})$ je c_1 výnos na jednotku aktivity x_1, c_2 pro aktivitu x_2 a dál. Koeficienty c_i a a_{in} jsou známé jako koeficienty účelové a omezovací funkce. [13]

Obecné předpoklady pro formulování matematického modelu jsou:

- Výnosy z různých alokací zdrojů lze měřit běžnou jednotkou (jako jsou třeba dolary, kilogramy nebo užitné hodnoty) a lze je porovnat.
- Zdroje se mají využívat co nejekonomičtěji.
- Všechna data jsou známa pro deterministické problémy.
- Rozhodovací proměnné jsou buď reálná čísla, celá, nebo kombinace obojího.
- Funkce je obecná, není omezená na žádný konkrétní typ. [13]

3.2 Optimalizační problém

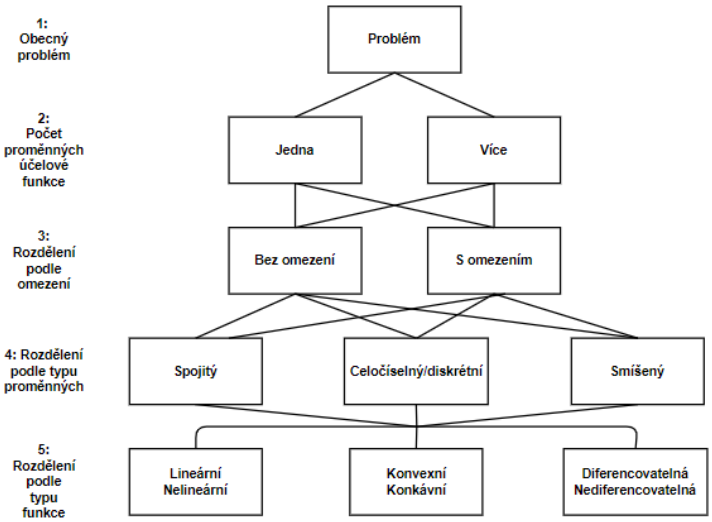
Problémy, které se snaží maximalizovat nebo minimalizovat matematickou funkci s určitým počtem proměnných a omezení, tvoří jedinečnou třídu optimalizačních problémů. V tomto obecném rámci je možné modelovat mnoho problémů jak skutečných, tak teoretických. Často tedy termín optimalizace nahrazuje právě výrazy maximalizovat či minimalizovat. Optimalizovaná funkce, známá jako funkce *účelová,* obsahuje obvykle několik proměnných. Optimalizační problémy mohou také obsahovat víc než jednu účelovou funkci. [13]

Podle povahy problému proměnné v modelu nabývají hodnot reálných nebo celých čísel a čísel binárních, nebo jejich kombinací. Optimalizační problém také může mít omezení, v matematickém modelu je levá strana omezovací funkce (případně nějaká proměnná) rovna, větší nebo rovna, či menší nebo rovna hodnotě pravé strany. [13]

3.2.1 Dělení optimalizačních problémů

Na obrázku 4 je obecné rozdělení optimalizačních problémů podle Sarkera (2007). Jak je psáno výše, proměnná účelové funkce může být jednoduchá i vícenásobná, dále lze funkci maximalizovat nebo minimalizovat podle zvoleného cíle. V reálném světě pravděpodobně neexistují žádné neomezené optimalizační problémy, proto má každý problém omezující funkce, proměnné meze (horní nebo dolní), nebo obojí. [13]

Proměnné mohou nabývat reálných, celých, binárních či smíšených hodnot, mnoho odborníků často proměnné klasifikuje jako spojité, celočíselné, diskrétní nebo smíšené. V problémech se spojitými proměnnými obecně hledáme množinu reálných čísel. Úlohy s celočíselnými či diskrétními proměnnými jsou úlohami kombinatorickými, kdy z konečné nebo nekonečné množiny hledáme obvykle celé číslo, množinu nebo graf. [13



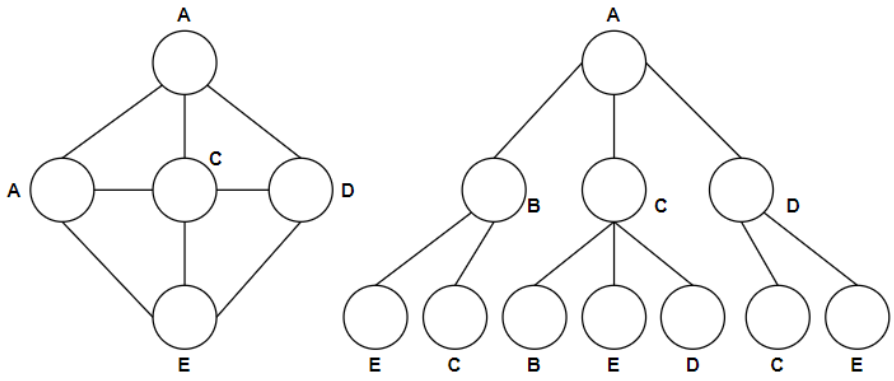
Obrázek 4: Dělení optimalizačních problémů podle [13]

4 PROHLEDÁVÁNÍ STAVOVÉHO PROSTORU

Prohledávání je primární technika používaná v informatice a operačním výzkumu pro řešení a optimalizaci složitých kombinatorických problémů. Cílem prohledávání je najít optimální, nebo alespoň téměř optimální řešení z konečné či nekonečné, ale spočetné množiny řešení. Počet řešení při hledání optima často nabývá až exponenciálních hodnot. Algoritmy, jako je třeba algoritmus *dynamického programování* nebo *metoda větví a mezí,* nacházejí optimální řešení. Algoritmy, které naleznou téměř optimální nebo přibližné řešení, se nazývají *aproximační* algoritmy. [14]

Stavový prostor sestává z množiny stavů a operátorů. Je to konfigurace stavů, které slouží k vyřešení problému. Operátory jsou akce, které propojují stavy dohromady. Obecně platí, že stavový prostor je graf, ve kterém uzly představují stavy a hrany představují operátory nebo přechody stavů. Prohledávání stavového prostoru je jeho systematické zkoumání za účelem nalezení jednoho nebo více řešení se specifikovanými vlastnostmi. Požadovaným řešením může být cílový uzel nebo cesta z počátečního stavu do uzlu cílového. Cílový stav nemusí být nemusí být popsán explicitně, může být popsán nutnými podmínkami a cílových stavů může být i více. [14,17]

Stromový graf neobsahuje žádné cykly, jeho koncové uzly se nazývají listy a náklady k jejich dosažení jsou různé. Inkluze a exkluze je obecný princip, který lze aplikovat na většinu kombinačních problémů, proto je stromový graf realistickým a obecným modelem pro řešení. Zatímco graf s cykly je obecnějším modelem stavového prostoru, lze jej převést při prohledávání na graf stromový za cenu vytvoření duplicitních uzlů.[14]



Obrázek 5: Přepis grafu do stromového tvaru podle [14]

Na obrázku 5 je patrné, že identický graf s cykly prohledávaný do hloubky přepsaný do stromového vykazuje lepší vlastnosti pro analýzu, a to zejména pro analýzu průměrných případů. Počet duplicitních uzlů, které lze vygenerovat, závisí na konektivitě grafu. [14]

4.1 Složitost algoritmů

Algoritmy jsou nejdůležitější součástí počítačových věd, protože je lze studovat nezávisle jazykově i strojově. Lze tedy porovnat účinnost algoritmů bez jejich přímé implementace. [16]

Pro výběr vhodného algoritmu pro daný účel je nutné vyhodnotit jeho efektivitu. Rozhodující charakteristikou je čas výpočtu (running time nebo execution time) závislý na počtu elementárních operací, jako je především sčítání, odčítání, násobení, porovnávání hodnot nebo přesun dat. *Paměťová složitost (space complexity)* odhaduje paměťové požadavky algoritmu. *Časová složitost (time complexity)* neboli délka práce je nejdelší možná část výpočtu. Vhodný algoritmus nalezne řešení za nejkratší čas nebo s nejmenšími nároky na paměť. [15]

Dva nejdůležitější nástroje podle (16) jsou model výpočtu RAM *(Random Access Machine)* a asymptotická složitost nejhoršího možného případu. Ke zhodnocení algoritmického výkonu se využívá O-notace (*Big Oh Notation*), která se ukazuje jako nezbytná pro porovnání a navrhování efektivnějších algoritmů.

Asymptotickou horní a dolní mez funkce vyjadřujeme pomocí θ-notace, pouze horní mez pomocí O-notace a pouze dolní mez pomocí Ω. [15]

4.1.1 Výpočet podle modelu RAM

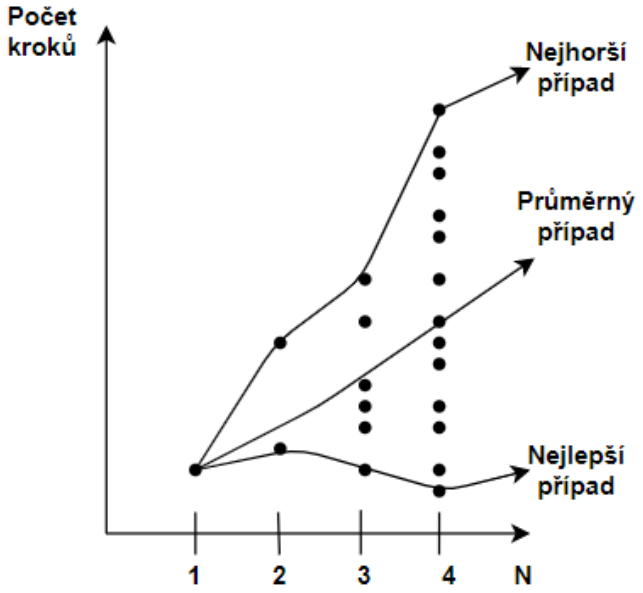
Návrh algoritmu nezávislého na stroji je možný díky teoretickému modelu počítače s názvem *Random Access Machine* neboli RAM. U tohoto modelu se setkáme s komponenty jako:

- Jednoduché operace, jako +, *, -, =, if, call, které trvají přesně jeden krok.
- Smyčky a podprogramy se nepovažují za jednoduché operace, je to uskupení mnoha jednostupňových operací. Nemá smysl, aby třídění probíhalo jako jednokroková operace, protože třídění 1 000 000 položek bude určitě trvat mnohem déle než třídění 10 položek. Čas potřebný k provedení smyčky nebo provedení podprogramu závisí na počtu iterací smyčky nebo na konkrétní povaze podprogramu.
- Každý přístup do paměti trvá přesně jeden časový krok. Máme tolik paměti, kolik potřebujeme. Model RAM nerozeznává, zda je položka v mezipaměti nebo na disku. [16]

V rámci modelu RAM měříme dobu běhu počítáním kroků, které algoritmus provede na dané instanci problému. Za předpokladu, že naše RAM provede daný počet kroků za sekundu, tento počet operací se přirozeně převede na skutečnou dobu chodu. Přestože je model RAM velmi zjednodušený při srovnání s výkonem počítače, třeba vynásobení dvou čísel zabere ve většině procesorů více času než přidání dvou čísel a stejně tak se liší časy přístupu do paměti v závislosti na tom, zda jsou data uložena v mezipaměti nebo na disku, i tak se RAM ukazuje jako vynikající model pro pochopení toho, jak bude algoritmus fungovat na skutečném počítači. [16]

4.1.2 Nejlepší, nejhorší a průměrná složitost

Pomocí modelu výpočtu RAM můžeme spočítat, kolik kroků náš algoritmus vykoná provedením s daným vstupem. Abychom však pochopili, jak dobrý nebo špatný je algoritmus obecně, musíme vědět, jak funguje ve všech případech. Abychom dosáhli možnosti porozumět pojmům nejlepší, nejhorší a průměrné složitosti, musíme popřemýšlet o spuštění algoritmu ve všech možných případech dat, která mohou být použita.



Obrázek 6: Nejlepší, nejhorší a průměrný případ složitosti podle [16]

Množina vstupů pro klasifikační problém obsahuje všechna možná uspořádání klíčů, přes všechny možné hodnoty n. Každou vstupní instanci můžeme reprezentovat jako bod v grafu (obr. 6), kde osa x představuje velikost vstupního problému (pro klasifikaci počet položek ke třídění) a osa y označuje počet kroků provedených algoritmem. Tyto body jsou přirozeně zarovnány do sloupců, protože pouze celá čísla představují možnou velikost vstupu, nemá smysl třídit třeba 0,57 položek. [16]

Na grafu na obrázku 6 lze definovat tři zajímavé funkce:

- Nejhorší případ je maximální počet kroků provedených v dané instanci velikosti *n*, proto funkci představuje křivka procházející nejvyšším bodem v každém sloupci.
- Nejlepší složitost algoritmu je definovaná minimálním počtem kroků v dané instanci o velikosti *n*, tedy křivka procházející nejnižším bodem každého sloupce.
- Průměrná složitost algoritmu je definovaná průměrným počtem kroků všech instancí o velikosti *n*. [16]

V praxi se ukazuje jako nejužitečnější nejhorší možný případ. [16]

4.1.3 O-notace

Jestliže je počet operací dán číslem n, pak je čas výpočtu určitou funkcí f čísla n, jejíž hodnota je $f(n)$. Tato hodnota může být vyjádřena natolik složitým výrazem, že její přesný výpočet může být obtížný. Proto se určí dominantní složka v $f(n)$, která pro velká n reprezentuje téměř celý čas výpočtu a potom lze zanedbat nepodstatné detaily. [15]
Rychleji rostoucí funkce je dominantní oproti funkci, která roste pomaleji. Jestliže f a g jsou funkce jiného typu, pak můžeme říct, že g dominuje f když $f(n) = O(g(n))$. [16]

Základní funkce seřazené podle rostoucí dominance:

- *Konstanty,* $f(n) = 1$: Funkce měří třeba náklady na přidání dvou čísel, výpis textu, nebo třeba realizaci funkce jako třeba $f(n) = \min(n, 100)$. Celkově nemá žádný vliv na parametr n.
- *Logaritmické funkce,* $f(n) = \log n$: Logaritmická složitost se projevuje v algoritmech jako je binární vyhledávání. Takové funkce rostou sice pomalu se zvětšováním n, nicméně rychleji než konstantní funkce, která stojí.
- *Lineární funkce,* $f(n) = n$: Takové funkce měří náklady na prohlížení každé položky jednou či vícekrát v n-prvkovém poli třeba k identifikaci největší či nejmenší položky nebo k výpočtu průměrné hodnoty.
- *Lineárně logaritmické funkce,* $f(n) = n \lg n$: Typická složitost řadicích algoritmů. Roste jen trochu rychleji než lineární funkce, ovšem dost na to, aby byla ve vyšším řádu dominance.
- *Kvadratická funkce,* $f(n) = n^2$: Funkce pro průchod všech dvojic universa s *n* prvky.
- *Kubická funkce,* $f(n) = n^3$: Funkce pro průchod všech trojic universa s *n* prvky.
- *Exponenciální funkce,* $f(n) = c^n \; s \; konstantou \; c > 1$ – Funkce jako 2^n vznikají při propočtu všech podmnožin *n* položek.
- *Faktoriálová funkce,* $f(n) = n!$: Funkce s faktoriálem vyhodnocují všechny možné permutace n prvků. [16]

Jiný zápis dominance funkcí:

$n! \gg 2^n \gg n^3 \gg n^2 \gg n \log n \gg n \gg \log n \gg 1$ [16]

5 METODY PROHLEDÁVÁNÍ STAVOVÉHO PROSTORU

Každá řídicí strategie, má-li být úspěšnou, musí splňovat dvě základní vlastnosti (Rich, Knight, 1991):

1. Musí *vést k prohledávání*, tj. způsobovat pohyb a zabraňovat cyklům v posloupnosti pravidel.
2. Musí být *systematická*. [17]

Při značné velikosti stavového prostoru může při jeho prohledávání dojít ke zbytečnému prohledávání části, která nevede k cíli. Proto je prohledávání omezeno využitím znalostí o problému. Jsou to znalosti často empirického rázu, nebo nepříliš přesné znalosti, o kterých víme, že jsou pro řešení užitečné, přestože nezaručují nalezení řešení. Tyto znalosti se nazývají **heuristikami** a používají se právě tam, kde není k dispozici přesný algoritmus. Problém, který je vybaven lepším souborem heuristik prohledá menší část stavového prostoru, postupuje přímočařeji k cíli a jeho způsob řešení se jeví inteligentněji. Algoritmy úloh, u kterých využíváme dané znalosti, se nazývají **informované**, naopak úlohy bez využití znalostí nazýváme **neinformované**. [17]

5.1 Neinformované metody prohledávání

Neinformované metody dělíme podle pořadí, ve kterém jsou uzly expandovány, a to na prohledávání do šířky *(breadth-first search, BFS)* a do hloubky *(depth-first search, DFS)*. Hloubkou uzlu rozumíme počet hran na cestě od počátečního k danému uzlu. [17]

Výhodou prohledávání do hloubky jsou nižší nároky na paměť, protože se do ní ukládají jen uzly na cestě od počátečního uzlu k uzlu právě expandovanému, prohledávání do šířky zase vždy vede k nalezení nejkratší cesty. [17]

Hlavní rozdíly mezi BFS a DFS je tedy pořadí, ve kterém prozkoumávají vrcholy. Toto pořadí souvisí s datovou strukturou použitou k uložení již objevených, ale nezpracovaných vrcholů.

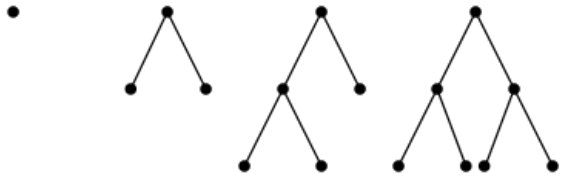
- *Fronta* – Při ukládání vrcholů do *fronty (FIFO, first-in, first-out)* nejprve prohledáváme nejstarší ještě neprohledané vrcholy. Prohledávání tedy pomalu postupuje od výchozího bodu, čímž je definováno *prohledávání do šířky*. [16]
- *Zásobník* – Při vkládání vrcholů do zásobníku *(LIFO, last-in, last-out)* postupujeme tak, že kdykoliv má daný stav k dispozici následovníka, pohybujeme se dál, hlouběji a zálohujeme pouze když jsme obklopeni dříve objevenými vrcholy. Prohledávání se tak rychle vzdaluje od výchozího bodu a definuje tak *prohledávání do hloubky*. [16]

5.1.1 Prohledávání do šířky (Breadth-first search)

Při prohledání do šířky je nejprve expandován kořenový uzel, tedy uzel s minimální hloubkou, potom jeho následovníci, pak následovníci následovníků a tak dále.

Obecně jsou všechny uzly v hloubce d expandovány před uzly v hloubce $d+1$. BFS lze implementovat pomocí funkce *fronty*, která nově generované stavy umístí na její konec, za všechny již generované stavy a je využívána pro dočasné ukládání nenavštívených uzlů. [17,18]

Prohledávání pomocí BFS probíhá systematicky, nejprve prohledá všechny uzly v hloubce 1, následně 2 a dál, což je zobrazeno na jednoduchém binárním grafu na obrázku 7.



Obrázek 7: Prohledávání do šířky po 0, 1, 2 a 3 expanzích podle [18]

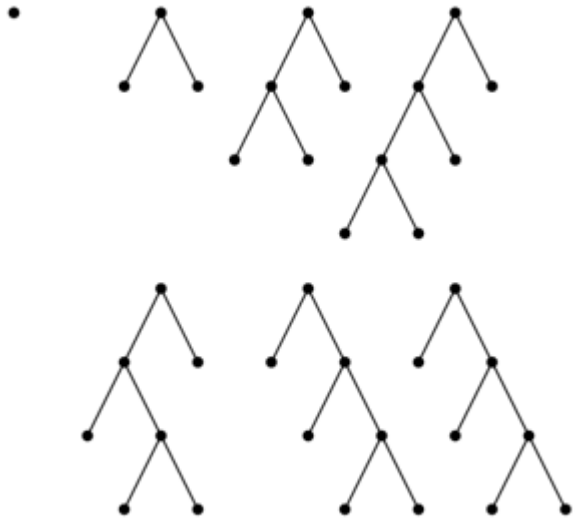
Pokud existuje řešení, BFS garantuje jeho nalezení a v případě, že je možných řešení více, algoritmus vždy najde nejdříve řešení s nejmenší hloubkou. [18]

Složitost algoritmu je $O(b^d)$, kde b je počet uzlů a $1 + b + b^2 + b^3 + \cdots + b^d$ je maximální počet expandovaných uzlů do nalezení řešení, avšak řešení může být nalezeno na kterékoli d. úrovni, takže v nejlepším případě může být tento počet i menší.

Časová i paměťová složitost je stejná, protože všechny listové stromy musí být v paměti zachovány současně. [18]

5.1.2 Prohledávání do hloubky (Depth-first search)

Prohledávání do hloubky vždy nejdříve expanduje uzel v nejhlubší úrovni stromu. Takto postupuje dál až do chvíle, kdy narazí na koncový uzel, tedy uzel, který již nelze expandovat. Tehdy se vrací zpět a stejným způsobem a expanduje uzly v hlubší úrovni.



Obrázek 8: Prohledávání do hloubky stromového binárního grafu. V hloubce 3 již nejsou žádní potomci. Podle [18]

Strategii lze implementovat pomocí funkce *zásobníku*, kdy je každý nově generovaný stav umístěn na jeho začátek, před všechny již generované stavy. [18]
Časová složitost prohledávání do hloubky j $O(b^m)$e. U problémů, které mají více řešení, může být prohledávání do hloubky rychlejší než prohledávání do šířky, protože je zde šance nalezení řešení po prozkoumání jen malé části celého prostoru. Prohledání do šířky musí vždy zkontrolovat všechny cesty délky $d-1$ než prohlédne délku d. Složitost prohledávání do hloubky je v nejhorším možném případě vždy $O(b^m)$. [18]

5.2 Informované metody prohledávání

Neinformované strategie jsou bohužel ve většině případů neúčinné. Informovaná metoda, která využívá znalosti specifické pro daný problém, hledá řešení efektivněji. Pomocí informované metody lze vyřešit i problémy s optimalizací. [18]

U informovaných metod jsou podle charakteru úlohy voleny hodnotící funkce, které se používají pro výběr uzlu k expanzi. Funkce, která vypočítává odhady nákladů, se nazývá heuristická funkce a obvykle se označuje písmenem h. Pokud hodnotící funkce dobře postihuje charakter a vlastnosti úlohy, budou vždy expandovány nejperspektivnější uzly a nedojde ke zbytečnému prohledávání cest, které nevedou do cíle. Kvalita hodnotící funkce tedy ovlivní efektivitu prohledávání. [17, 18]

5.2.1 Gradientní algoritmus (hill-climbing algorithm)

Nejjednodušší verzí neinformovaného prohledávání je podle Pearla (1984) *gradientní algoritmus* (hill-climbing algorithm), který vždy expanduje ten uzel, který byl podle hodnotící funkce (maximální nebo minimální hodnota následovníků) vybrán jako nejlepší. Nejlepší z následovníků je vybrán k další expanzi, „rodič“ i „sourozenci“ jsou smazáni, algoritmus uchovává v paměti jen právě rozvíjený uzel. Prohledávání je zastaveno, když je dosaženo stavu, který má lepší hodnocení podle hodnotící funkce než jeho následovníci. Nevýhodou gradientní strategie je, že může skončit v lokálním extrému. [15, 17]

5.2.2 Algoritmus upořádaného prohledávání (best-first search)

Tento algoritmus vznikl rozšířením gradientního algoritmu o paměť. Je tedy znám předchůdce, následovník i hodnota dané hrany (cena průchodu, postih). Algoritmus prozkoumá nejdříve políčka, o kterých heuristika říká, že jsou nejblíže cíli. [15, 17]

5.2.3 Greedy algoritmus

Greedy algoritmus je jedním z nejjednodušších vyhledávacích strategií typu *best-first search*. Minimalizuje odhadované náklady na dosažení cíle tím, že uzel, jehož stav je považován za nejblíže cílovému stavu, je vždy rozšířen jako první. U většiny problémů lze odhadnout náklady na dosažení cíle z konkrétního stavu, ale nelze je určit přesně, proto je hodnotící funkce heuristická. [18]

Greedy metoda pracuje v iteracích a v každé iteraci je vybrán jeden kandidát jako možné optimální řešení. V případě, že nějaký kandidát nemůže být zařazen jako optimální řešení, je z množiny kandidátů definitivně vyřazen (Kubincová, 2007). [15]

5.3 Hledání nejkratší cesty

Dopravní sítě mohou být modelovány pomocí grafů obsahujících uzly propojené hranami. Pro hledání nejkratších cest mezi uzly v grafu existuje mnoho různých algoritmů. Všechny algoritmy uvedené níže pracují s ohodnocenými grafy. Síť tvoří množina N s uzly n a množina E s hranami m. Každá hrana propojuje dva uzly (i, j). Váha hrany je pro dané přímé spojení označena $d(i, j)$, pro ostatní spojení značí nejkratší vzdálenost. [19]

5.3.1 Dijkstrův algoritmus

Pomocí Dijkstrova algoritmu lze najít nejkratší cestu z daného uzlu i do cílového uzlu, případně do ostatních uzlů grafu s nezáporně ohodnocenými hranami (Dijkstra 1959). Nejprve je vytvořen seznam dosud nenavštívených uzlů a orientační vzdálenost z uzlu i do ostatních uzlů j, zvaná *Dist[j]*.

Seznam *Prev[j]* ukládá uzly již navštívené při cestě z uzlu i do uzlu j. Všechny vzdálenosti jsou primárně nastaveny na hodnotu nekonečno. Pro aktuální uzel, nejprve

počáteční, uzel *i* jsou všichni jeho sousedé (všechny uzly propojené hranou s uzlem *i*) uvažováni jako další možný krok cesty a jsou zohledněny jejich vzdálenosti od uzlu *i*. Jakmile jsou uváženi všichni sousedé příslušného uzlu, uzel je smazán ze seznamu nenavštívených uzlů. Novým aktuálním uzlem se stává uzel s nejmenší vzdáleností od uzlu *i* (uzel, jehož vzdálenost od *i* je nejmenší hodnota v *Dist[j]*). Algoritmus končí, jakmile je smazán cílový uzel ze seznamu nenavštívených uzlů, nebo když je tento seznam prázdný. [19]

Nejjednodušší implementace Dijkstrova algoritmu ukládá uzly do propojeného seznamu nebo pole a operace k nalezení minimální hodnoty v seznamu *Dist* je lineární vyhledávání ve všech uzlech v *Dist*. V tomto případě je časová složitost $O(n^2)$, kde n je počet uzlů. Složitost však lze snížit u řídkých grafů, tj. u grafů, které mají méně než n^2 hran, díky inteligentnější strategii ukládání grafu ve formě seznamů sousedství a použití různých typů hald k provedení operace k nalezení minimální hodnoty v *Dis*t. Se strategií, která použije Fibonacciho haldy, lze dobu chodu zkrátit na $O\ (m\ +\ n\ \times\ log\ (n))$ (Cormen a spol. 2001; Fredman a Tarjan 1987). [19]

5.3.2 A * algoritmus

Algoritmus A * nabízí pro nalezení nejkratší cesty mezi dvěma uzly v grafu obecnější přístup než algoritmus Dijkstrův. Jak je uvedeno výše, princip Dijkstrova algoritmu je založen na prohledávání do šířky, kdy další zkoumaný uzel je ten s minimální vzdáleností od počátečního uzlu (uvedený v *Dist*).

Algoritmus A * zavádí heuristiku k určení pořadí, ve kterém jsou uzly vybrány v procesu vyhledávání. Tato heuristika je součtem vzdálenosti k aktuálnímu uzlu k (udává se jako *Dist [k]*) a odhadu vzdálenosti k cílovému uzlu *j*, obvykle označovanému jako *h (k).* Ve většině implementací se *h (k)* počítá jako euklidovská vzdálenost od uvažovaného uzlu k cílovému uzlu. Pokud jsou k dispozici souřadnice všech uzlů, lze vzdálenost vypočítat například pomocí Pythagorovy věty. [19]

5.3.3 Floyd-Warshallův algoritmus

Algoritmus vyvinuli nezávisle na sobě Floyd (1962) a Warshall (1962). Místo výpočtu cesty z daného počátečního uzlu do všech ostatních uzlů (nebo jediného cílového uzlu) se všechny nejkratší cesty, tj. z každého uzlu do všech ostatních, počítají v rámci jedné smyčky. Ve výsledku získáme matici *Dist,* kde *Dist [i, j*] označuje vzdálenost od uzlu *i* k uzlu *j*. Dále lze vypočítat matici *Next, kde Next [i, j]* představuje nástupce uzlu *i* na nejkratší cestě od uzlu *i* k uzlu *j*. [19]

Časová složitost algoritmu je $O(n^3)$, což je *n*-krát ekvivalentní provedení Dijkstrova algoritmu. Provedení algoritmu je obvykle rychlejší než provedení Dijkstrova algoritmu pro každý uzel. [19]

6 ROUTOVÁNÍ V DOPRAVNÍKOVÉM SYSTÉMU

Pro směrování (routování) dopravníků nebo personálu ve složitých logistických systémech se používají aplikace, které detailně popisují systém pomocí modelu přepravních sítí a pracovních stanic. Počet relevantních uzlů tak může snadno přesáhnout i 10 000. Základní úlohou je často nalezení nejkratší cesty ze startovního uzlu do uzlu cílového. V poslední době byly některé simulační nástroje rozšířeny o prohledávající algoritmy. Doba provedení simulačního modelu může významně záviset na počtu uzlů v síti. [19]

6.1 Implementace automatického směrování v simulačních nástrojích

Existuje řada aplikací, jejichž součástí jsou simulační modely, ve kterých lze směrovat personál a dopravníky. Základní úloha je najít nejkratší cestu z aktuální pozice do cíle, k čemuž se využívá řada výše uvedených známých algoritmů. Některé simulační systémy nabízí automatické směrování vozidel tak, že se přemístí tou nejkratší cestou do cílového uzlu. Výkon simulačních nástrojů se může lišit v některých důležitých aspektech:

- Omezená velikost přepravních sítí (počet uzlů) které lze simulovat.
- Průměrný výpočetní čas pro splnění určitého počtu přepravních objednávek během simulačního běhu.
- Průměrný výpočetní čas a výsledná velikost modelu pro vytvoření odpovídajícího grafu a matice, ve které jsou uloženy nejkratší cesty. [19]

V oblasti diskrétní simulace jsou k dispozici různé softwarové nástroje (průzkum trhu viz Lindemann a Schmid (2007)). Gutenschwager a spol. (2012) porovnali tři simulační systémy široce používané v oblasti výroby a logistiky: Tecnomatix Plant Simulation od společnosti Siemens, AutoMod od Applied Materials a Enterprise Dynamics od INCONTROL Simulation Solutions. Byl srovnáván výkon automatického směrování vozidel z aktuální pozice do cíle po nejkratší cestě v dopravní síti. [19]

Podle srovnání přístupů k hledání nejkratších cest (Zhan a Noon 1998), Gutenschwager a spol. (2012) rozlišili implementaci algoritmů nejkratší cesty v simulačních modelech na dva případy:

- Nejkratší cesty se počítají během simulace (např. Dijkstrův algoritmus nebo A *).
- Všechny nejkratší cesty jsou předem vypočítány a jsou uloženy v příslušných maticích (např. Algoritmus Floyd-Warshall). Jakmile je algoritmus propočítán, nalezení nejkratší cesty, nebo nalezení dalšího uzlu, který má být navštíven, na cestě k cílovému uzlu,se provádí vyhledáním v matici *Next(Další)*.[19]

Při porovnání simulačních nástrojů s odlišnými strategiemi automatického směrování, kdy Plant Simulation počítá nejkratší cesty během simulace, zatímco AutoMod

a Enterprise Dynamics počítají relevantní cesty předem, došlo k závěrům, že záleží na dané simulační studii a preferencích: Pokud je nutno opakovaně používat stejné cesty, předběžné výpočty AutoModu a Enterprise Dynamics se vyplatí. Při použití Floyd-Warshallova algoritmu jsou sice kratší výpočetní časy, ovšem za ceny větší spotřeby úložného prostoru. Tyto programy proto nejsou schopné simulovat velké dopravní systémy. Pouze Plant Simulation dokáže zvládnout sítě s více než 3000 uzly. [19]

Právě schopnost Plant Simulation zvládnout největší počet uzlů a dále strategie jeho prohledávacího algoritmu, kdy nedochází k žádnému předzpracování dopravní sítě před spuštěním simulačního běhu, takže je možné měnit dopravní síť za běhu simulace, byl tento program zvolen pro řešení problému této diplomové práce.

6.2 Tecnomatix Plant Simulation

Software Tecnomatix Plant Simulation umožňuje simulaci a optimalizaci výrobních systémů a procesů. Pomocí Plant Simulation lze optimalizovat tok materiálu, využití zdrojů a logistiku pro všechny úrovně plánování závodu od globálních výrobních zařízení, přes místní závody až po konkrétní linky. [9]

Plant Simulation poskytuje jednosměrné i obousměrné trasy pro modelování dopravníkových sítí (Bangsow 2010). Každá trasa má počáteční a koncový bod, které mohou být spojeny s počátečními nebo koncovými body jiných objektů trasy. Obslužnou rutinu události lze vyvolat kdykoli vozidlo dosáhne počátečního nebo koncového bodu trasy. Pomocí tohoto konceptu lze modelovat libovolné složité dopravníkové systémy. [19]

Podle Gutenschwagera a spol. (2012) Plant Simulation používá pro určení cesty vozidla z jeho aktuální polohy k danému cíli Dijkstrův algoritmus a A * nebo jejich odvozeninu. Před provedením simulačního běhu není nutné žádné předběžné zpracování dopravníkové sítě. Je dokonce možné upravit síť během simulačního běhu. Vozidla k dosažení cíle vždy použijí nejkratší cestu. [19]

Automatické vyhledání nejkratší cesty bylo vyhodnoceno jako potřebné pro aplikaci, která bude v diplomové práci vytvářet routovací tabulku. Plant Simulation také poskytuje celou řadu snadno použitelných nástrojů pro analýzu modelů se stochastickými procesy, pro výpočet distribucí hodnot vzorků, pro správu simulačních experimentů a pro stanovení optimalizovaných parametrů systému. Díky těmto nástrojům bude možné následně přidat do původní aplikace dynamické stochastické procesy s následnou optimalizací jako součást dizertační práce.

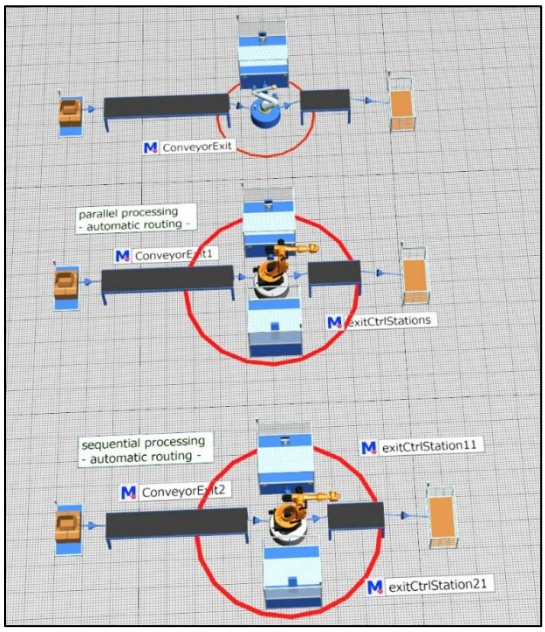
Nástroje dále umožňují simulaci různých variant řešení podle scénářů, vyhodnocení pomocí analytických, statistických a grafických nástrojů. Vizualizace kompletního modelu v prostředí Plant Simulation 3D umožňuje působivou 3D prezentaci chování systému.

Logické soubory lze použít k vizualizaci simulace v prostředí virtuální reality. Zároveň je možné optimalizovat výkon systému a eliminovat úzká místa při zachování nízkých nákladů.

Díky Plant Simulation může dojít k opodstatnění strategických rozhodnutí již ve fázi úvah o nové výrobě a vytvořit denní operativní plánování výroby s cílem maximálního využití zdrojů při změnách vstupů. [9,20]

Klíčové vlastnosti:

- Objektově orientovaný.
- Interaktivní, strukturovaný a hierarchický program.
- Možnost 2D i 3D zobrazení.
- Součástí jsou uživatelské a aplikační knihovny.
- Programování Metod, které řídí činnost projektu pomocí jazyku SimTalk (varianta C++).
- Součástí jsou generátory náhodných čísel pro přesnější přiblížení funkce modelu ke skutečnosti.
- Obsahuje moduly pro Analýzu (Sankey diagram, Analyzátor úzkých míst, Ganttův diagram, Statistický analyzátor, Manager pro řízení experimentů)
- Součástí je genetický algoritmus pro automatickou optimalizaci parametrů systému. [20]



Obrázek 9: Stroj PickAndPlace s automatickým směrováním [20]

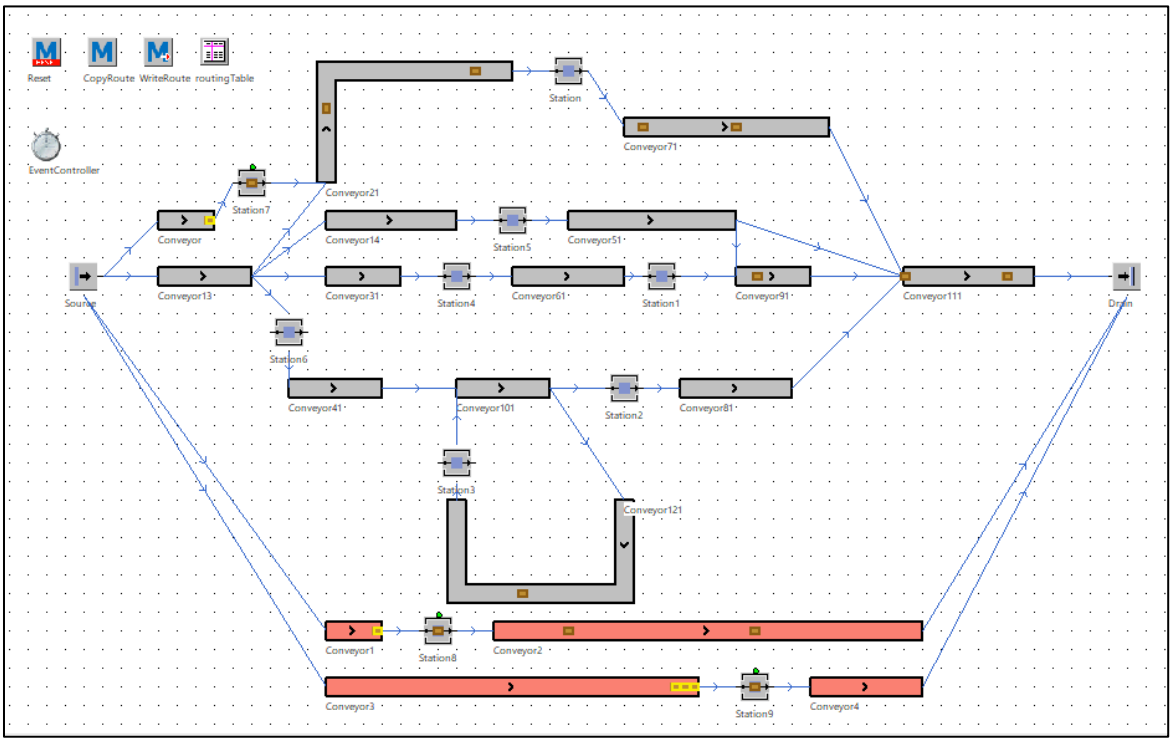
7 VÝVOJ AUTOMATICKÉHO ALGORITMU PRO TVORBU ROUTOVACÍ TABULKY

Routovací (směrovací) tabulka, známá především v informatice jako datová struktura pro směrování dat procházejících počítačovou sítí, je v aplikaci využita jako tabulka pro směrování MUs *(Mobile Units)* v dopravníkovém systému. Princip její tvorby bude založen na postupném vyplnění tabulky trasami, kterými prošly MUs.

7.1 Princip algoritmu

Algoritmus pro zápis do routovací tabulky nejdříve využívá původní algoritmus zabudovaný v Plant simulation, který podle Gutenschwagera a spol. (2012) prohledává síť pomocí Dijkstrova algoritmu, A*, případně jejich odvozenin.

Princip zápisu spočívá v tom, že ze zdroje (*Source*) je vysláno několik *MUs (Mobile Units)* které automaticky projdou celou sítí do cíle (*Drain*) a během cesty budou postupně zapisovat uzly, kterými prošly a přičítat délky dopravníků tak, aby měla každá *MU* v sobě uloženou svoji trasu a její délku.



Obrázek 10: Model dopravníkového systému pro zápis do routovací tabulky. Hnědé čtverečky znázorňují MUs, žluté MUs čekají ve frontě před obsazenou stanicí. Červeně jsou zvýrazněné úmyslně vytvořené trasy shodných délek.

Testovací model

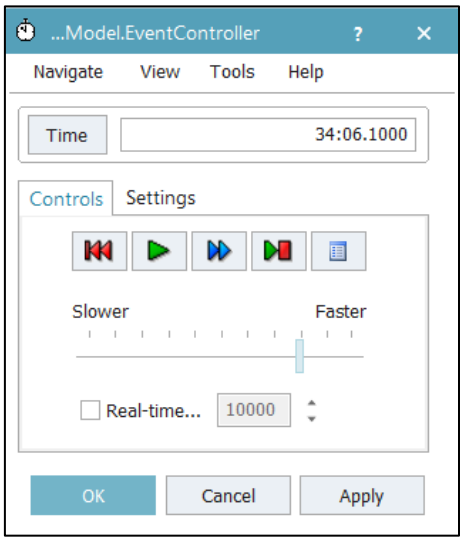
Testovací model na obrázku 10 je složen z 18 dopravníků *(Conveyor)* různých délek, z nichž jeden dopravník vytváří zpětnou vazbu, z deseti pracovních stanic *(Station)* a tří metod.

Počet MUs vyslaných ze *Source* musí být při zápisu do routovací tabulky dostatečný k projití všech možných cest v síti.

7.2 Zápis do routovací tabulky

Metoda *WriteRoute* slouží k zápisu uzlů, které postupně navštíví daná MU a také k přičítání vzdáleností, které MU již absolvovala. Metoda se zavolá vždy, když MU vstoupí do nějakého objektu v cestě. Metoda *CopyRoute* pak zapisuje cesty a jejich délku do tabulky *routingTable* a spustí se ve chvíli, kdy MU vstoupí do *Drainu,* tedy do svého cíle. Metoda *Reset* maže hodnoty v *routingTable* na začátku každé další zapisovací simulace.

Simulace se spouští v *EventControlleru,* kde je možné simulaci resetovat, spustit a zastavit, zrychlit, krokovat, debugovat její další krok, spustit v reálném čase a nastavit její rychlost.

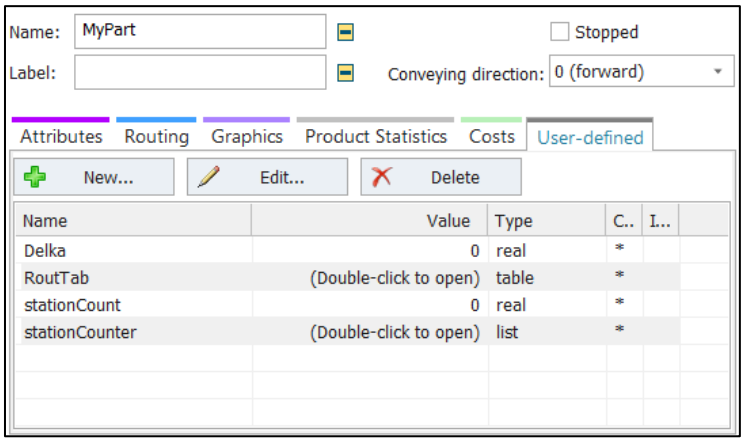


Obrázek 11: Event Controller

Objekty MUs

Objekty MUs určené pro vytvoření routovací tabulky v sobě mají čtyři parametry, které se s každým krokem v dopravníkové síti mění (obr. 12). *Delka,* v sobě postupně přičítá vzdálenost, kterou urazila ze *Source*.

RoutTab je jednořádková tabulka, která postupně připisuje do buňky následného sloupce uzel, který právě navštívila, a *stationCount* a *stationCounter* slouží zápisu všech stanic, které se v dané trase nachází.



Obrázek 12: Objekt MU, jeho název a proměnné

Zápis do tabulky

Při zapisování jednotlivých cest do *routingTable* je nutné zkontrolovat, zda nejsou duplicitní s cestou již zapsanou. K tomu slouží cyklický kód, který se spustí při vstupu MU do cíle a nachází se v metodě *CopyRoute*.

K číslování řádků již zapsaných v tabulce slouží globální proměnná *n* uložená v cíli *Drain*. Tato proměnná se zvýší pokaždé, když je do routovací tabulky uložen nový řádek, tedy cesta nově příchozího MU do cíle. Nová cesta se ukládá pouze v případě, že stejná trasa ještě nebyla v tabulce nalezena.

	real 1	list 2	string 3	string 4	string 5	string 6
string	delky	stationy				
1	26.00	list21	Conveyor1	Station8	Conveyor2	
2	26.00	list22	Conveyor3	Station9	Conveyor4	
3	28.00	list23	Station15	Conveyor13	Conveyor21	Station14
4	9.00	list24	Conveyor5	Station3	Conveyor6	Station12
5	37.00	list25	Conveyor	Station7	Conveyor21	Station
6	23.00	list26	Conveyor5	Station3	Conveyor101	Station2
7	23.00	list27	Station11	Conveyor41	Conveyor101	Station2
8	32.00	list28	Station15	Conveyor13	Conveyor14	Station5
9	26.00	list29	Conveyor	Station7	Conveyor21	Station14

Obrázek 13: Část routovací tabulky, první sloupec obsahuje délky jednotlivých tras, druhý sloupec seznamy všech navštívených stanic.

Kontrola duplicit

První MU, která dorazí do cíle je vždy unikátní, proto se rovnou zapíše do tabulky.Při každé další příchozí je nutné porovnat její posloupnost uzlů, zda není shodná s nějakou posloupností již uloženou.

Pro optimalizaci algoritmu (kontrolování všech posloupností uzlů by bylo časově i paměťově náročné) se vždy nejprve zkontroluje délka příchozí trasy. V případě, že není shodná s žádnou trasou již zapsanou v tabulce, je jisté, že ani posloupnost uzlů, které navštívila, nebude stejná.

Pro potvrzení, že se do tabulky zapíšou i cesty stejné délky, avšak odlišné trasy, byly v modelu vytvořeny dvě trasy stejné délky (obr. 10, červené trasy). Ve výsledné tabulce lze vidět, že trasy shodné délky jsou dokonce tři.

Data pro další směrování

Každá buňka druhého sloupce obsahuje seznam stanic (datový typ *List)*, které daný řádek popisující cestu obsahuje. Tento seznam si nese každá MU v sobě. Postupně si zapisuje každou stanici, kterou navštíví a následně je spolu s celou trasou zapíše do routovací tabulky (obr. 13). První a druhý sloupec slouží k dalšímu směrování různých posloupností stanic.

Opakování zpětných vazeb

Počet různých opakování u zpětné vazby roste s počtem poslaných MUs ze zdroje. S vyšším počtem MU tedy roste počet různých tras, kvůli různému počtu opakování zpětné vazby. Tomuto lze zamezit přidáním omezující metody na výstup zpětného *Conveyeru* a proměnné MU počítající požadovaný počet opakování.

```
if drain.n=0
    @.routTab.copyRangeTo({1,1}..{*,1}, routingTable, 3,routingTable.yDim+1)
    routingTable[1,routingTable.yDim] := @.Delka
    routingTable.createNestedList(2,1)
    routingTable[2,1]:=@.stationCounter
    drain.n+=1
else
    var duplicita := false
    for var i := 1 to drain.n
        if routingTable[1,i] = @.Delka
            var j:=1
            var duplicityFound := true
            repeat
                if routingTable[j+2,i] /= @.routTab[j,1]
                    duplicityFound:=False
                end
                j+=1
            until routingTable[j+2,i] /= void or @.routTab[j,1] /= void
            duplicita := duplicityFound
            if duplicita = true
                exitLoop
            end
        end
    next
    if duplicita = False
        @.routTab.copyRangeTo({1,1}..{*,1}, routingTable, 3,routingTable.yDim+1)
        routingTable[1,routingTable.yDim] := @.Delka
        routingTable.createNestedList(2,routingTable.yDim)
        routingTable[2,routingTable.yDim]:=@.stationCounter
        drain.n+=1
    end
end
```

Obrázek 14: Algoritmus pro tvorbu routovací tabulky

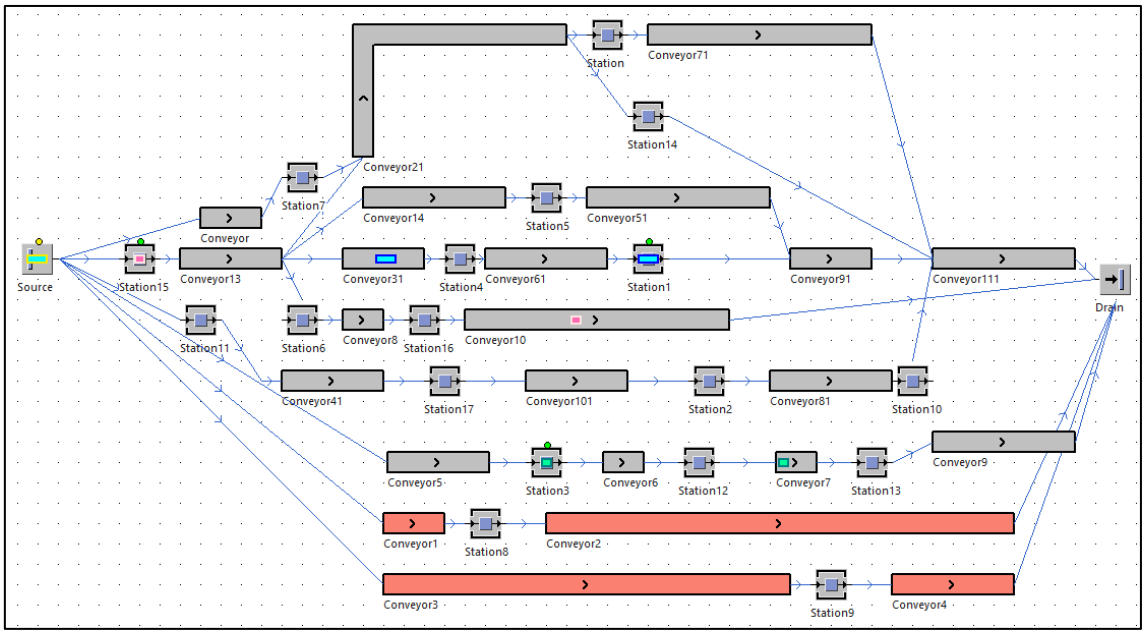
Na obrázku 14 první příchozí MU rovnou zapíše včetně svojí délky a seznamu stanicí, kterými prošla. Proměnné datového typu boolean duplicita a duplicityFound mají pravdivostní hodnotu podle nálezu shodných objektů. V případě, že v tabulce ještě není zapsána cesta o délce nové příchozí trasy, trasa se rovnou zapíše. V opačném případě se opakuje cyklus repeat, dokud nenajde odlišný objekt, nebo pokud jedna z porovnávaných cest není prázdná.

7.3 Směrování a ohodnocení tras dle statických kritérií

Jako statická kritéria byly zvoleny stanice, které musí být navštíveny, a délka trasy. Vznikly tak tři možnosti směřování chodu dané MU dle statických kritérií:

- Zadané určité stanice, kterými musí MU projít.
- MU musí projít určitou stanicí, a to nejrychlejší možnou cestou.
- MU musí projít libovolnou nejkratší cestou.

Metoda *pathChoose* se spustí pro každou MU než vyjde ze zdroje *(Source)*. Obsahuje tři základní podmínky, podle kterých vznikne trasa pro MU podle jejích parametrů.



Obrázek 15: Model dopravníkového systému, simulace se třemi typy MU

MUs s danými stanicemi, kterými musí projít

Na obrázku 14, modré MUs, trasa přes stanice *Station15, Station4* a *Station1*. Pro tento typ tras musí mít MU zadáno minimálně dvě stanice, kterými má projít. Tyto stanice jsou uloženy v proměnné MU s názvem *StationToGo*. Tato proměnná je datového typu *List*, požadované stanice jsou vypsány do sloupce.

Proměnná booleovského typu *ShortestPath* má v tomto případě hodnotu *false*. Po splnění podmínky se spustí cyklus, který nejprve prohledá, ve kterých trasách v *routingTable* se nachází první stanice zapsaná do *StationToGo*. Tyto trasy vypíše do listu *potencionalPath* jako čísla řádků *routingTable*, obsahující požadovanou stanici. Cyklus dále prohledává potenciální trasy tak, že vezme další stanici ze *stationToGo*, která porovnává už jen cesty v *potencionalPath* tak, že iterativně porovná všechny řádky a od prvního řádku listu *potentionalPath* přepisuje ty, v nichž se sama nachází.

Poslední stanice ve *StationToGo* má tedy v prvním řádku *potencionalPath* trasu, ve které se nacházejí všechny požadované stanice a tato trasa se zapíše do proměnné MU *Trace*. MU postupuje v systému podle posloupnosti uzlů uložených v *Trace* tak, že se na konci každého uzlu spustí metoda *Move*. Metoda zvýší proměnnou *collOfTrace* o 1 při každém průchodu nějakým uzlem pro získání jména následujícího uzlu v předepsané posloupnosti v *Trace*.

```
if @.stationToGo[1] /= void AND @.ShortestPath = False
 var l:=1
    repeat
        if l = 1
            i := 1
            j :=1
            k := 1
            repeat
                repeat
                    if routingTable[2,i][j] = @.stationToGo[l]
                        @.potencionalPath[k] := i
                        k+=1
                    end
                    j+=1
                until routingTable[2,i][j] = void
                i+=1
                j := 1
            until routingTable[2,i] = void
        else
            j:=1
            k:=1
            var m:=1
            repeat
                repeat
                    if routingTable[2,@.potencionalPath[k]][j] = @.stationToGo[l]
                        @.potencionalPath[m] := @.potencionalPath[k]
                        m+=1
                    end
                    j+=1
                until routingTable[2,@.potencionalPath[k]][j] = void
                k+=1
                j:=1
            until @.potencionalPath[k] = void
        end
        l+=1
    until @.stationToGo[l] = void
    routingTable.copyRangeTo({3,@.potencionalPath[1]}..{*,@.potencionalPath[1]},@.Trace,1,1)
    target:= str_to_obj(@.Trace[@.collOfTrace,1])
    @.move(target)
    @.collOfTrace+=1
end
```

Obrázek 16: Algoritmus procházející zadané stanice

Proměnná *l* (Obr. 16) iteruje přes všechny požadované stanice přes první *repeat* cyklus, další dva *repeat* cykly projedou celou routovací tabulku a zapíšou do ní trasy, ve kterých se nachází první požadovaná stanice. Za *else* už se porovnávají další požadované stanice jen s trasami potenciálními. Trasa, která obsahuje všechny žádané stanice je vždy v prvním řádku *potencionalPath*, jako trasa, kterou následně MU půjde. MU je vyslána do dalšího objektu podle posloupnosti v ní uložené.

MUs s danou stanicí, kterou musí projít nejkratší cestou

Na obrázku 14 růžové MUs, nejkratší cesta procházející stanicí *Station15*. Pravdivostní hodnota proměnné *ShortestPath* je *True* a ve *stationToGo* je zapsaná požadovaná stanice.

Nejprve se, podobně jako s předchozím typem MU, najde v *routingTable* každá trasa, ve které se daná stanice nachází a zapíše do listu *potencionalPath*.

Následně se porovnají všechny délky cest a číslo řádku nejkratší trasy bude opět v prvním řádku *potencionalPath.* Tento řádek se zkopíruje do tabulky *Trace*. Jakmile má MU svoji *Trace*, další průběh je u všech typů MU totožný.

```
if @.stationToGo[1] /= void AND @.ShortestPath = True
    var i := 1
    var j := 1
    var k := 1
        repeat
            repeat
                if routingTable[2,i][j] = @.stationToGo[1]
                    @.potencionalPath[k] := i
                    k += 1
                end
                j += 1
            until routingTable[2,i][j] = void
            i += 1
            j := 1
        until routingTable[2,i] = void
        i := 1
        var short := routingTable[1,@.potencionalPath[1]]
        repeat
            if routingTable[1,@.potencionalPath[i]] < short
                short := routingTable[1,@.potencionalPath[i]]
                @.shortest := @.potencionalPath[i]
            end
            i+=1
        until @.potencionalPath[i] = void
        routingTable.copyRangeTo({3,@.shortest}..{*,@.shortest},@.Trace,1,1)
        var target:object:= str_to_obj(@.Trace[@.collOfTrace,1])
        @.move(target)
        @.collOfTrace+=1
end
```

Obrázek 17: Algoritmus procházející určitou stanici nejkratší možnou cestou.

Algoritmus na obrázku 17 v prvních dvou *repeat* cyklech prozkoumá, ve kterých trasách v routovací tabulce se stanice nachází. Tyto trasy jsou uloženy jako potenciální trasy do listu *potencionalPath*. Následuje další *repeat* cyklus, ve kterém je vybrána trasa s nejmenší délkou a podle posloupnosti uzlů v této trase pak MU pokračuje do dalšího objektu trasy.

MUs po nejkratší cestě

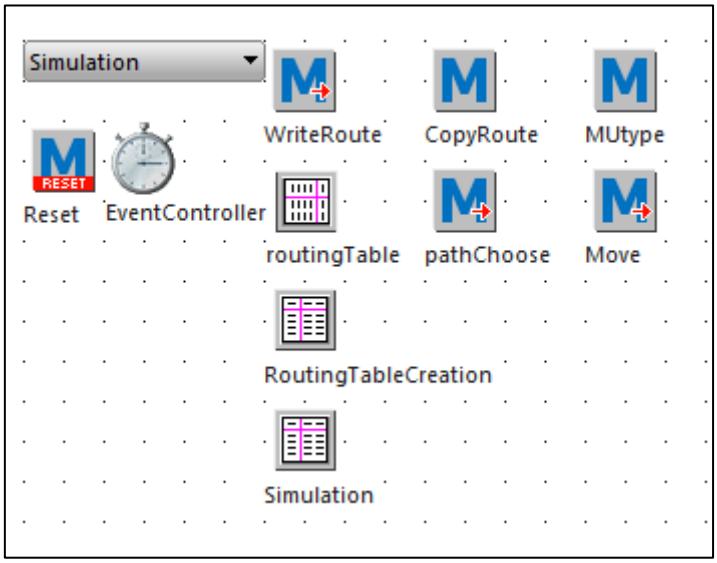
Na obrázku 14, zelené MUs, pravdivostní hodnota proměnné *ShortestPath* je *True*, seznam stanic, které musí být navštíveny, je prázdný. Algoritmus pouze porovná všechny délky v prvním sloupci *routingTable* a vybere první nejkratší zapsanou. Podle čísla řádku se doplní *Trace* a každá MU pokračuje po dané trase stejně jako ostatní typy MUs. Algoritmus je totožný s druhou částí algoritmu na obrázku 17.

```
if @.ShortestPath = True AND @.stationToGo[1] = void
    i:=1
    short := routingTable[1,1]
    repeat
        if routingTable[1,i] < short
            short := routingTable[1,i]
            @.shortest := i
        end
        i+=1
    until routingTable[1,i] = void
routingTable.copyRangeTo({3,@.shortest}..{*,@.shortest},@.Trace,1,1)
target:=str_to_obj(@.Trace[@.collOfTrace,1])
@.move(target)
@.collOfTrace+=1
end
```

Obrázek 18: Algoritmus pro nalezení nejkratší cesty z celé sítě.

7.4 Simulační model s vytvořením routovací tabulky

Výše uvedené dílčí algoritmy pro výběr trasy různých typů MUs jsou spojeny v metodě *pathChoose*, která se spustí pokaždé, když nějaká MU opouští *Source*. Pro propojení tvorby routovací tabulky a následné simulace s různými typy MUs bylo použito tlačítko, objekt *DropDownList*, které přepíná mezi simulací a tvorbou routovací tabulky. S tlačítkem je spojená metoda *MUtype*, která slouží k přiřazení jedné ze dvou tabulek určených pro uložení odlišných typů MUs. Tabulka *RoutingTableCreation* má v sobě MU s parametry určenými pro zápis do tabulky. V tabulce *Simulation* je uloženo několik různých typů MU podle jejich volby trasy.



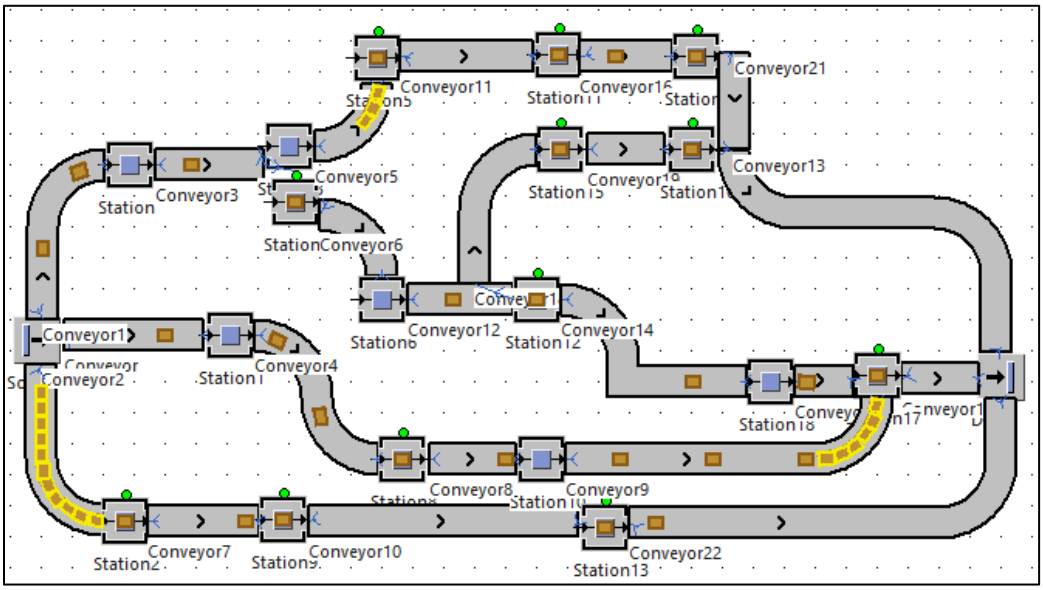
Obrázek 19: Tlačítko pro volbu mezi vytvořením směrovací tabulky, Event Controller pro ovládání simulací, objekty metody Plant Simulation, obsahující potřebné algoritmy, routovací tabulka a tabulky pro různé typy MUs.

```
if dropDownList.Value = 2
    if @.stationToGo[1] /= void AND @.ShortestPath = True v … end
    if @.stationToGo[1] /= void AND @.ShortestPath = False v … end
    if @.ShortestPath = True AND @.stationToGo[1] = void i … end
end
if dropDownList.Value = 1
    @.move
end
```

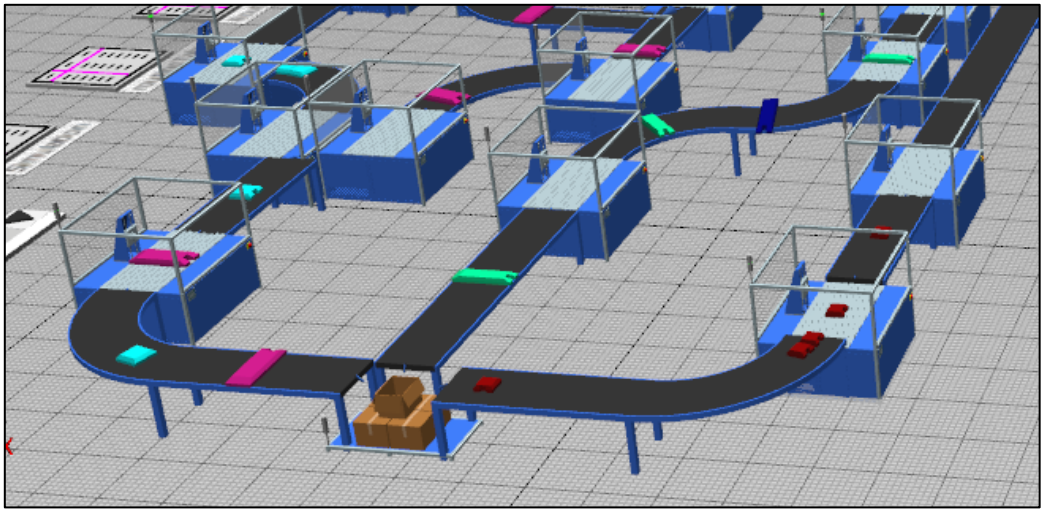
Obrázek 20: Propojení dílčích algoritmů v metodě pathChoose. Dvě možnosti tlačítka DropDownList

7.5 Použití algoritmů na nový model a zobrazení ve 3D

Doposud byly algoritmy tvořeny a testovány na dvou základních modelech. Obrázek 10 na vývoj algoritmu pro vytvoření routovací tabulky a obrázek 15 k testování různých typů MU. Následně byl vytvořen model vhodný pro důstojnější vykreslení ve 3D a přenositelnost algoritmů. Externě vypsané metody viz. obr. 19 byly uloženy do vlastních objektů, které nyní tyto metody používají napevno.



Obrázek 21: Prohledávání tras pro vytvoření routovací tabulky, 2D model.



Obrázek 22: Výřez 3D simulace systému na obrázku 21.

8 PRINCIP ALGORITMU SE ZAHRNUTÍM DYNAMICKÝCH KRITÉRIÍ

Dosavadní zpracování zahrnuje pouze statická kritéria, přičemž hlavní atributy směrování toku MUs byly délka cesty a stanice, které musely být navštíveny. Stanice i dopravníky však mají spoustu aspektů proměnných v čase a ovlivňujících průběh toku v síti. V prvé řadě deterministické prvky, což je doba trvání zadané práce na stanici, či doba průchodu dopravníkem. Dále může dojít ke stochastickým jevům, což můžou být různé poruchy, zpoždění materiálu apod.

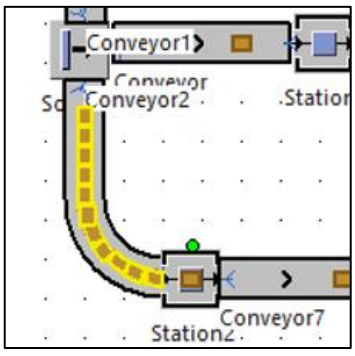
8.1 Přidání dynamických kritérií do routovací tabulky

V každé stanici je nastavitelná **doba zpracování** (*Processing time*), což je doba, po kterou se MU nachází na daném objektu a dochází k jejímu zpracování.
Také lze nastavit čas, kdy začne stanice zpracovávat jiný typ MU a čas obnovy, kdy je vyžadováno uvést stanici do definovaného stavu, než může začít zpracovávat další část. U dopravníků je nastavitelná **rychlost** (Speed) a lze přidat i **zrychlení** *(Acceleration).*

Doposud se do routovací tabulky zapisovaly jen možné trasy a jejich délky. S přidáním dynamických kritérií by každá MU měla ve svých proměnných přidané i doby průchodu přes všechny stanice a dopravníky a ty by se do routovací tabulky rovnou zapisovaly také.

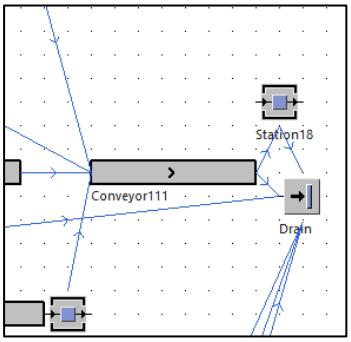
8.2 Optimalizace algoritmu

Na obrázcích simulací jsou patrné fronty, tvořící se před některými stanicemi a stejně tak i stanice prázdné, což zdržuje celou výrobu a zvyšuje náklady. Tyto fronty lze omezit vhodným vysíláním MUs ze zdroje upravenou metodou *pathChoose.* Je nutné se zaměřit na nejkritičtější místa v síti, a to křižovatky. Dynamická kritéria navíc přináší další aspekty, které ovlivňují chod toku.



Obrázek 23: Fronta MU;s tvořící se před Station2

Pro zahrnutí dynamických kritérií a následnou optimalizaci je nutné pozměnit již vytvořené algoritmy. Nynější algoritmus vypíše díky prohledávacímu algoritmu Plant Simulation při dostatečném počtu prohledávacích MUs všechny objekty, nacházející se v dopravníkové síti. Při určitých typech větvení však nenajde všechny možné trasy.



Obrázek 24: Problémové místo v síti

Na obrázku 24 je vidět situace, kdy do dopravníku *Conveyor111* vedou čtyři různé trasy. Z něj je možno jít buď rovnou do cíle, do *Drain*,nebo ještě přes *Station18*. Vzniká tak 8 různých tras přes *Conveyor111*. Prohledávácí algoritmus však některé trasy vynechá.

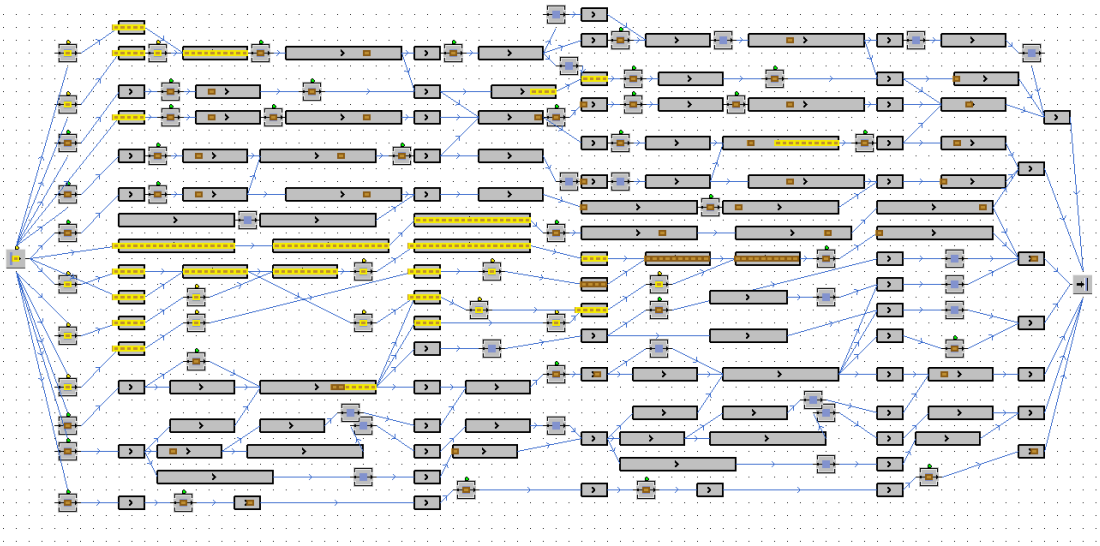
Z tohoto důvodu by pro optimalizování systémů s dynamickými kritérii bylo vhodné ze vzniklé routovací tabulky vytvořit tabulku novou. Tabulka bude obsahovat seznamy předchůdců a nasledovníků se svými atributy, data bude získávat z původní routovací tabulky. Vytvoření vhodného algoritmu může být předmětem dizertační práce.

9 ZHODNOCENÍ A DISKUZE

V praktické části práce byl nejprve vytvořen algoritmus pro tvorbu routovací tabulky. Algoritmus využívá automatické směrování Plant Simulation tak, že jsou ze zdroje vyslány objekty MUs, které si během cesty do cíle ukládají posloupnost uzlů a hran, kterými prošly, včetně délky celé trasy. V cíli byl vytvořen algoritmus, který ukládá do routovací tabulky poze unikátní trasy.

V případě, že se dopravníková síť rozvětvuje do více cest, které se posléze sejdou v jednu či více a dále se nevětví, algoritmus do tabulky zapíše všechny unikátní trasy. V opačném případě, pokud se systém větví i z dopravníku, do kterého vedlo více cest, některé trasy chybí, ačkoliv uzly jsou prozkoumané všechny. Řešením by byl algoritmus, který by z routovací tabulky vytvořil další tabulku, která by obsahovala seznamy všech předchůdců a následovníků. Díky tabulce by pak bylo možné směrovat přes libovolné trasy, nejen trasy, zapsané v routovací tabulce.

Při nízkém počtu MUs vyslaných ze zdroje, nebudou prohledány všechny uzly, naopak při příliš velkém počtu dochází ke zbytečnému navýšení času a využití paměti. Algoritmus pro routování byl otestován na modelu čítajícím téměř 200 objektů a nalezl 36 tras. Pro nalezení těchto tras bylo potřeba ze zdroje vyslat 668 MUs.



Obrázek 25: Algoritmus pro zápis do routovací tabulky na větším modelu

Na obrázku 25 je patrné, že je využit zbytečně velký počet MUs. Stanice, které mají nastavený základní čas pro zpracování výrobku, zdrží prohledávací MUs, zatímco na dopravník se posílají v pravidelných intervalech další MUs.

Pro zahrnutí dynamických vlastností a následnou optimalizaci je nutno k algoritmu přidat další atributy. Námětem na další práci je tedy vytvořit pomocí původního algoritmu s přidanými dynamickými atributy tabulku předchůdců, následovníků s jejich statickými i dynamickými vlastnostmi.

10 ZÁVĚR

Cílem diplomové práce byl vývoj algoritmu pro tvorbu routovací tabulky a ohodnocení cesty v dopravníkovém systému. V rešeršní části byla nejprve popsána problematika modelování a simulace. Do problematiky modelování bylo zahrnuto modelování výrobních systémů a popis výrobního systému s dopravníky včetně typů dopravníků. Po následujícím popisu simulace byly shrnuty její výhody a nevýhody.

S modelováním a simulací dále souvisí optimalizace, v práci byl popsán matematický optimalizační model a dělení optimalizačních problémů. Nástroje pro simulaci výroby používají prohledávací algoritmy, proto bylo zpracováno prohledávání stavového prostoru a výpočetní náročnost algoritmů. Dále byly popsány metody prohledávání stavového prostoru, rozdělení na neinformované a informované metody a algoritmy pro hledání nejkratší cesty.

Díky těmto algoritmům je možné vytvořit tabulku pro směrování v dopravníkovém systému. Na základě srovnání simulačních nástrojů (Gutenschwager a spol., 2012) byl pro tuto práci vybrán program Tecnomatix Plant Simulation, který jako jediný z porovnávaných programů dokázal zpracovat až 3000 uzlů. Plant Simulation hledá nejkratší cesty za běhu simulace, nemusí tedy dělat předchozí výpočty, což sice činí program pomalejší, na druhou stranu šetří paměť.

V praktické části byl nejprve vytvořen algoritmus, který pomocí prohledávacího algoritmu Plant Simulation vytvoří routovací tabulku všech tras, kterými je možno projít. Následně bylo možno vyslat objekty ze zdroje po trase, která navštíví zadané stanice, dále po nejkratší možné trase celé sítě a po nejkratší trase k zadané stanici.

Algoritmus pro zápis do tabulky zapíše všechny unikátní trasy v případě, že se předchozí rozvětvené trasy po spojení do jedné již dále nevětví. V opačném případě jsou sice projity všechny objekty v sítí, všechny možné trasy však nikoliv. Dále bylo zjištěno při spuštění zapisovacího algoritmu, že používá zbytečně vysoké množství testovacích objektů. Vyřešení tohoto problému a optimalizace algoritmu je námět na další práci.

Vývoj se zabýval především vytvořením algoritmu pro zápis do tabulky a jejím následným použitím pro vysílání objektů po daných typech tras.

Pro zařazení dynamických vlastností a následnou optimalizaci by bylo vhodné po zápisu tabulky vytvořit tabulku obsahující seznam předchůdců a následovníku objektů a z nich by pak bylo možné směrovat i přes trasy, které původní algoritmus neobjevil. Původní algoritmus by byl rozšířen o dynamické proměnné s následným vytvořením další tabulky.

V této tabulce by byli uloženi všichni předchůdci a následovníci, takže by bylo možno ve fázi simulace směrovat po celé síti, nejen přes předpřipravené trasy. Tímto by se také vyřešil problém stochastických jevů. Tento způsob přesahuje možnosti použitého algoritmu a může být následně zpracován v dizertační práci.

11 SEZNAM POUŽITÉ LITERATURY

[1] ONDRÁČEK, Emanuel a Přemysl JANÍČEK. *Výpočtové modely v technické praxi*. Praha: SNTL - Nakladatelství technické literatury, 1990. ISBN 80-03-00458-6.

[2] EL-HAIK, Basem a Raid AL-AOMAR. *Simulation-based lean six-sigma and design for six-sigma*. Hoboken, N.J.: Wiley-Interscience, 2006, 404 s. ISBN 978-047-1694- 908

[3] PENHAKER, Marek, Petr TIEFENBACH a František KOBZA. *Modelování a simulace biologických systémů: cvičení*. Ostrava: VŠB - Technická univerzita Ostrava, 2007. ISBN 978-80-248-1560-2.

[4] LAW, Averill M. *Simulation modeling and analysis. Fifth edition*. New York: McGraw-Hill Education, 2015. ISBN 978-1-259-25438-3.

[5] TUČEK, David a Roman BOBÁK. *Výrobní systémy*. Vyd. 2., upr. Zlín: Univerzita Tomáše Bati ve Zlíně, 2006. ISBN 80-731-8381-1.

[6] RÁBOVÁ, Zdena a Roman BOBÁK. *Modelování a simulace*. 3., přeprac. vyd. Brno: VUT, 1992. ISBN 80-214-0480-9.

[7] Monroy, Carolina. *What is a convoyor system?*. In: 6river.com [online]. 19-10-2020 [cit. 27-3-2021]. Dostupné z: https://6river.com/what-is-a-conveyor-system/.

[8] Gajdůšek, Jaroslav a Miroslav Škopán. TEORIE DOPRAVNÍCH A MANIPULAČNÍCH ZAŘÍZENÍ. Vysoké učení technické v Brně v Čs. Redakci VN MON, 1988 č.j. 164/99860/88

[9] BANGSOW, Steffen. Tecnomatix Plant Simulation: *Modeling and Programming by Means of Examples*. 2. Switzerland: Springer, 2020. ISBN 978-3-030-41543-3.

[10] HOUŠKA, Milan. *Simulační modely I.* V Praze: Česká zemědělská univerzita, Provozně ekonomická fakulta, 2005. ISBN 978-80-213-1334-7.

[11] BANKS, Jerry, ed. *HANDBOOK OF SIMULATION: Principles, Methodology, Advances, Applicatons, and Practice*. 2. Atlanta, Georgia: John Wiley, 1998. ISBN 978-0-471-13403-9.

[12] NOCEDAL, Jorge a Stephen J. WRIGHT. Numerical optimization. 2nd ed. New York: Springer, 2006. Springer series in operations research and financial engineering. ISBN 0387303030.

[13] SARKER, Ruhul A. a Charles S. NEWTON. Optimization Modelling: A Practical Approach. London: CRC Press, 2007. ISBN 1420043102.

[14] ZHANG, Weixiong. State-space search: algorithms, complexity, extensions, andApplications. New York: Springer, 1999. ISBN 978-1-4612-7183-3.

[15] ZELINKA, Ivan. Evoluční výpočetní techniky: principy a aplikace. Praha: BEN - technická literatura, 2009. ISBN 9788073002183.
[16] SKIENA, Steven S. The algorithm design manual. 2nd ed. London: Springer, c2008. ISBN 978-184-8000-704. [12] NOCEDAL, Jorge a Stephen J. WRIGHT. Numerical optimization. 2nd ed. New York: Springer, 2006. Springer series in operations research and financial engineering. ISBN 0387303030.
[17] MAŘÍK, Vladimír, Olga ŠTĚPÁNKOVÁ a Jiří LAŽANSKÝ. Umělá inteligence. Praha: Academia, 1993-. ISBN 978-80-200-2276-9.
[18] RUSSELL, Stuart J. a Peter NORVIG. Artificial intelligence: a modern approach. 3rd ed. Upper Saddle River: Prentice Hall, 2010. Prentice Hall series in artificial intelligence. ISBN 978-0136042594.
[19] GUTENSCHWAGER, Kai, Sven VOLKER, Axel RADTKE a Georg ZELLER. The shortest path: Comparison of different approaches and implementations for the automatic routing of vehicles. In: Proceedings Title: Proceedings of the 2012 Winter Simulation Conference (WSC) [online]. Berlin, Germany: IEEE, 2012, 2012, s. 1-12 [cit. 2021-4-28]. ISBN 978-1-4673-4782-2. ISSN 0891-7736. Dostupné z: https://informs-sim.org/wsc12papers/includes/files/con231.pdf.
[20] AXIOM TECH. TECNOMATIX PLANT SIMULATION. In: axiomtech.cz [online]. [cit. 2-5-2021]. Dostupné z: https://www.axiomtech.cz/25357-texnomatix-plant-simulation
[21] BANGSOW, STEFFEN. PickAndPlace machine loading using automatic routing [picture]. 11.3.2021. In: bangsow.eu [online]. [20.5.2021]. Dostupné z: https://www.bangsow.eu/detail_en.php?id=849

12 SEZNAM ZKRATEK, SYMBOLŮ, OBRÁZKŮ A TABULEK

12.1 Obrázky

12.2 Zkratky a symboly

$f(x)$	účelová funkce proměnné x
$g_i(x), h_j(x)$	omezující funkce
BFS	Prohledávání do šířky (Breadth-First Search)
DFS	Prohledávání do hloubky (Depth-First Search)
FIFO	front (First-in, First-Out)
LIFO	zásobník (Last-In, Last-Out)
MU, MUs	mobilní jednotka (Mobile Unit), mobilní jednotky (Mobile Units)

13 SEZNAM PŘÍLOH

model_3D.spp
Testovaci_model.spp
Velky_dopravnikovy_system.spp