VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MATEMATIKY

INSTITUTE OF MATHEMATICS

VIRTUÁLNÍ TOVÁRNA ZA POMOCI METOD MATEMATICKÉHO MODELOVÁNÍ

VIRTUAL FACTORY USING MATHEMATICAL PROGRAMMING METHODS

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAN SVOBODA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. TOMÁŠ MAUDER, Ph.D.

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav matematiky
Student:	**Bc. Jan Svoboda**
Studijní program:	Aplikované vědy v inženýrství
Studijní obor:	Matematické inženýrství
Vedoucí práce:	**Ing. Tomáš Mauder, 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:

Virtuální továrna za pomoci metod matematického modelování

Stručná charakteristika problematiky úkolu:

Filozofie Průmyslu/Industry 4.0 ukazuje možné směry vývoje a nastiňuje opatření, která podpoří budoucnost ekonomiky a průmyslu ve smyslu zefektivnění výroby, minimalizování ekologické stopy, automatizaci, bezpečnosti aj. Pomocí stále se rozšiřujících IT možností začíná být standardem využívat digitální kopii továrny a jednotlivých výrobních procesů. Simulace výrobního procesu pomocí matematických metod patří v dnešní době k nepostradatelným pomocníkům při plánování výroby, rozvržení servisních odstávek, k tvorbě nové výrobní linky, atd. Kombinací simulačního přístupu s heuristickou optimalizací pak dostáváme nástroj, který je sám schopen najít optimální výrobní scénáře, optimální rozvržení servisních úkonů a zvýšení výtěžnosti linky na maximum. Práce se zabývá zejména vytvořením vlastního simulačního nástroje s heuristickou nadstavbou včetně validace výsledků.

Cíle diplomové práce:

Cílem diplomové práce je matematický popis problematiky výrobního řetězce založeného na teorii front. V práci vznikne vlastní diskrétní simulační model vytvořený v programu MATLAB. Nástavbu modelu bude tvořit heuristický optimalizační algoritmus, který bude hledat optimální nastavení výrobní linky tak, aby byla dosažena maximální výrobní propustnost. Pro simulaci náhodných vlivů může být použití metody Monte Carlo. V případě, že budou k dispozici reálná výrobní data, bude vytvořený model validován s reálnou výrobou, jinak se model ověří v simulačním prostředí SIMULIK–SIMEVENTS.

Seznam doporučené literatury:

GROSS, Donald, SHORTLE, John F., THOMPSON, James M. a HARRIS, Carl M. Fundamentals of queueing theory. 4th ed. Hoboken: John Wiley & Sons, 2008. Wiley series in probability and statistics. ISBN: 9780471791270.

RAO, Singiresu S. Engineering optimization: Theory and practice. Čtvrte vydání. Hoboken: Wiley, 2009. ISBN: 978-0-47 0-18352-6.

PASCUAL, D. Galar, DEPONTE, Pasquale a KUMAR, Uday. Handbook of Industry 4.0 and SMART Systems. CRC Press, 2019. ISBN: 9781138316294.

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

V Brně, dne

L. S.

prof. RNDr. Josef Šlapal, CSc.
ředitel ústavu

doc. Ing. Jaroslav Katolický, Ph.D.
děkan fakulty

Abstrakt
Práce je zaměřena na řešení problémů v moderním odvětví Industry 4.0. Hlavní část práce se věnuje návrhu a vlastní implementaci diskrétního simulačního modelu vycházejícího z teorie front, který bude využit k heuristické optimalizaci výrobní linky. Diskrétní model je validován s provozními daty z reálné výrobní linky. Zvýšení efektivity výroby pomocí kombinace diskrétního modelu a optimalizačního algoritmu je demonstrována na modelovém příkladě výrobní linky.

Summary
This work is focused on solving problems in Industry 4.0. Main part of this work describes development of a discreet simulation model based on queuing theory. This model will be used for a heuristic optimization of a production line. Model will be validated with data from real production line. Improvement of effectivity using discrete model and heuristic optimization will be shown on virtual production line.

Klíčová slova
Heuristická optimalizace, matematický model, teorie front, průmysl 4.0, výrobní linka

Keywords
Heuristic otimization, mathematical model, queuing theory, industry 4.0, production line

SVOBODA, J. *Virtuální továrna za pomoci metod matematického modelování*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2021. 46 s. Vedoucí diplomové práce Ing. Tomáš Mauder Ph.D.

Prohlašuji, že jsem práci Virtuální továrna za pomoci metod matematického modelování vypracoval sám pod vedením Ing. Tomáše Maudera, Ph.D. a materiály uvedenými v seznamu literatury.

Bc. Jan Svoboda

Obsah

Úvod

V dnešní době se v průmyslu klade stále větší důraz na zvýšení zisku a snížení nákladů. S tím je spojena modernizace výroby. Fáze, ve které se nacházíme se označuje jako čtvrtá průmyslová revoluce a využíváme metod Industry 4.0, kde jednotlivé stroje ve výrobní lince komunikují přes síť a díky tomu dokážeme výrobu lépe organizovat a řídit.

Cílem této práce je vytvořit diskrétní simulační model, který bude modelovat výrobu. V praxi se takovýto simulační model označuje jako digitální dvojče. Díky tomuto dvojčeti dokážeme vyzkoušet implementaci změn výrobní linky bez nutnosti zasahovat do skutečné linky, která je v provozu. Největším benefitem je tedy optimalizace výroby s minimálními náklady.

Nejprve budeme modelovat tok produktů linkou složenou ze strojních zařízení. V další fázi se pokusíme vytvořit model pro linku, kde jsou jednotlivá zařízení obsluhovaná šestiosým robotem. Tento model potom validujeme na reálných datech z výroby od společnosti ALPS Electric Czech.

V poslední části zkusíme do našeho modelu implementovat systém logiky obsluhy jednotlivých strojních zařízení šestiosým robotem. Dalším krokem bude určit takovou logiku obsluhy, aby linka pracovala co možná nejefektivněji. K určení optimalní logiky využijeme metod heuristiké optimalizace. V první řadě implementujeme algoritmus Random Search. Zrychlení optimalizace nám zajistí sofistikovanější algoritmus diferenciální evoluce.

1. Industry 4.0

1.1. Základní pojmy

1.1.1. Čtvrtá průmyslová revoluce

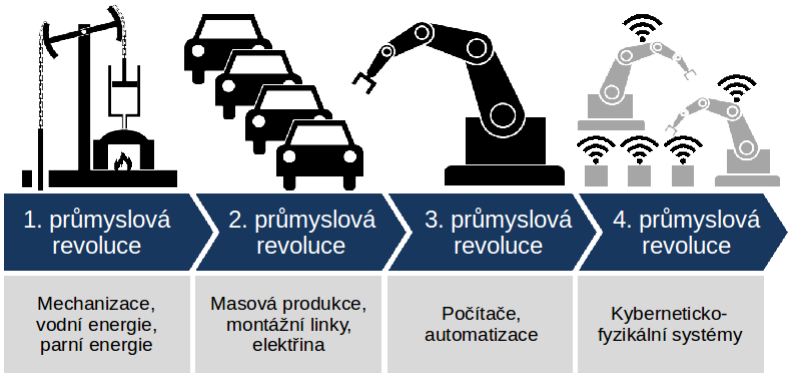
Průmyslovou revoluci lze popsat jako postupnou změnu v hospodářství a průmyslu, kdy dojde díky pokroku ve vědě a technice k ulehčení náročné práce lidí, snížení nákladů a maximalizaci možných zisků.

První z těchto revolucí probíhala v 18. a 19. století, po vynalezení parního motoru Jamesem Wattem. Začaly se využívat nové zdroje energie jako uhlí, které pohánělo parou poháněné stroje. V této době také nastal přechod výroby z manufaktur na strojovou velkovýrobu.

Druhou průmyslovou revolucí nazýváme období následující ihned po první průmyslové revoluci. Hlavním znakem tohoto období na přelomu 19. a 20. století je využití elektrické energie, využití výrobních linek a s nimi související dělbu práce pro zvýšení zisků společností.

Počátkem třetí průmyslové revoluce bývá označován rok 1969, kdy vznikl první programovatelný logický počítač (zkratka PLC). Toto období se nese v duchu aplikace informační technologie a automatizace výroby. Datovaní je ovšem spíše orientační, protože využití počítačů a automatizace v průmyslu byla spíše evoluce než označovaná revoluce.

V období čtvrté průmyslové revoluce se nacházíme právě teď a budeme ji pozorovat alespoň dalších 20 let. Je založené na komunikaci kybernetických systému přes IoT - Internet of Things, který označuje síť pro decentralizované řízení. Co to v praxi znamená je to, že zařízení ve výrobním procesu se mohou sama rozhodovat na základě dat, které mají se senzorů, kamer a cloudového úložiště. Tímto stylem je možné aby roboti a zařízení vykonávali jednoduché, opakovatelné úkony, které dřív dělali lidé. Toto ovšem vznáší dotazy, jestli kvůli nahrazovaní lidské pracovní síly roboty nenastane problém s vysokou nezaměstnaností. Na tento problém není nikdo schopen stoprocentně odpovědět, ovšem ekonomové tvrdí, že tímto postupem zaniknou pouze pracovní pozice s nízkou hodnotou a naopak vniknou nové pozice s vyšší hodnotou a vyššími požadavky na vzdělání spojené s údržbou, návrhem hardware a software pro roboty. Některé odhady dokonce tvrdí, že každé místo které bude v důsledku zavádění Industry 4.0 zrušeno, vznikne 2,5 nových pracovních pozic [1,2].



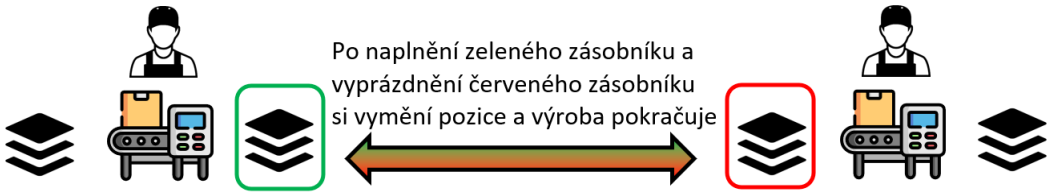
Obrázek 1.1: Schéma průmyslových revolucí [1]

1.1.2. Výrobní linka a její typy

Výrobní linku můžeme definovat jako proces, který se skládá z částí. Tyto jednotlivé části mají za úkol postupně utvářet produkt za cenu nižší než, by byla ruční výroba [3].

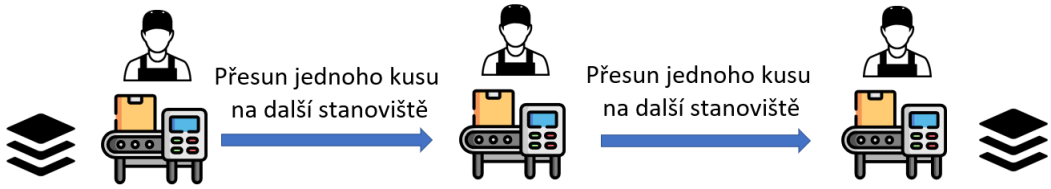
První továrny organizované jako výrobní linky zavedla firma Ford MotorCompany na počátku 20. století při výrobě modelu T. Právě díky sériové výrobě na linkách bylo možné snížit výrobní náklady a také čas výroby automobilu. Výrobní linky jako takové bych rozdělil, podle způsobu přenosu jednotlivých částí produktů mezi stanovišti. [4]

Představme si jedno stanoviště jako zařízení, které má na vstupu zásobník s produkty, které má zpracovat. Na výstupu zařízení je další zásobník, tentokrát na hotové produkty. Tento zásobník se po naplnění převeze na další stanoviště ke vstupu. Druhá možnost je že další zařízení odebírá části přímo ze zásobníku předchozího zařízení pomocí dopravníku. Tento model linky má nevýhodu v tvorbě mezizásob ve výrobě, což je nežádoucí. Můžeme si představit že po konci výrobní směny na lince zůstanou rozpracované díly, které zůstanou nehlídány. Tudíž může dojít k poškození nebo ztrátě a tedy zvýšení nákladů. Další problém nastává při přenastavení linky na jiný model tzv. ChangeOver, kdy je úkolem výroby co nejrychleji začít vyrábět nový model. To ovšem není možné, protože se nejprve musí vyrobit kusy v mezizásobě. To sníží efektivitu linky.



Obrázek 1.2: Schéma výrobní linky se zásobníky

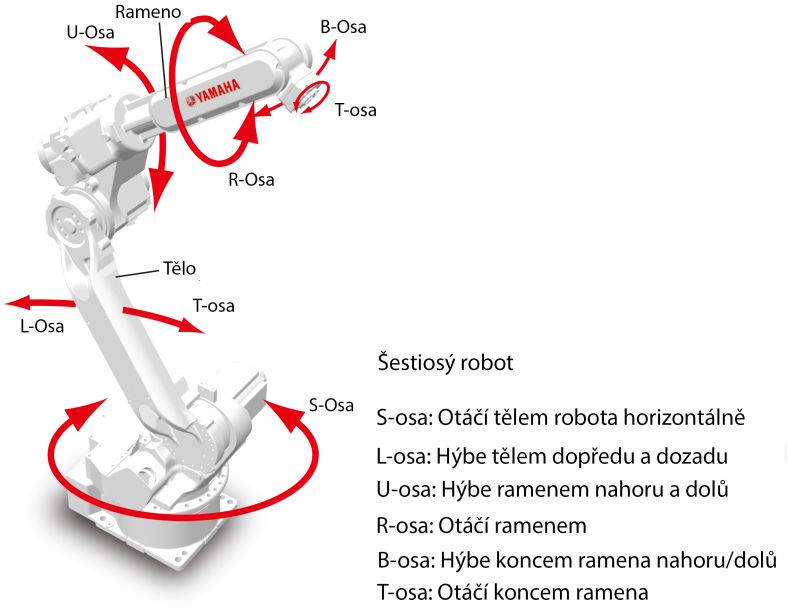
Další typ linky využívá takzvaný One Piece Flow. Na stanovišti takovéto linky nejsou žádné zásobníky. Každé zařízení zpracovává pouze jeden kus a ten poté pošle dál a z předchozího stanoviště vezme nový kus. Tento způsob výroby zamezuje tvorbě mezizásob, a výrazně zkracuje dobu na ChangeOver. Nastávají ovšem další výzvy u návrhu procesů a linky. Je třeba aby jednotlivá zařízení měla balancovaný tok produktu oproti zbytku linky. Vezměme příklad že jedno zařízení bude zpracovávat kus déle než zbytek linky. Toto zařízení by brzdilo celou výrobu a nestal se z něj tzv. Bottle Neck (tj. úzké místo výroby, proto analogie s hrdlem lahve) a proto by bylo třeba umístit například 2 takovéto zařízení vedle sebe paralelně aby neklesala efektivita výroby. Samotný přenos mezi zařízeními může vykonávat operátor nebo robot [5].



Obrázek 1.3: Schéma výrobní linky s One Piece Flow

1.1.3. 6-osý robot

Jedním ze základu moderních automatických linek je 6-axis robot. Proč potřebujeme na pohyb ve 3D prostoru 6 os? Problém je to že se potřebujeme k danému bodu prostoru dostat z různých směrů. Robotické rameno se tedy může pohybovat v šesti osách jak je na obrázku 1.4



Obrázek 1.4: Osy pohybu robota [7]

Takto pohyblivý robot má široké pole využití. Na jeho rameno se dají připevnit různé přípravky podle potřeby výroby. Nejzákladnější využití je pouze pro přenášení podsestav v prostoru výrobní linky. Dále můžeme na hlavu ramena implementovat šroubováky, svářečky, brusky a mnoho další nástavce. V této práci se ovšem budeme zabývat pouze roboty využívanými pro přenos kusů po lince.

1.1.4. PLC

PLC je zkratka z anglického výrazu Programmable Logic Controller. V podstatě je to malý průmyslový počítač, který se využívá především k řízení automatických procesů. Většino jsou konstruovány robustněji než standardní počítače, protože musí řídit celou část výroby kterou mají na starosti. Program v těchto zařízeních probíhá cyklicky po celý čas. Na PLC se většinou nachází velké množství analogových a digitálních vstupů, ke kterým se připojují nejrůznější sensory. Na základě dat ze vstupů PLC vyhodnotí stav a vyšle odpovídající signál s pokyny pro další postup. Dá se říct že PLC je mozek automatické výrobní linky. [8]

Obrázek 1.5: PLC

1.1.5. Traceabilita

Základem moderní výroby je Traceabilita. Český význam by se dal popsat jako vystopovatelnost produktu v čase. Každý kus ve výrobě musí mít jednoznačné identifikační číslo nebo kód, který je uloženo v databázi. Úplně stejným způsobem musí být označené každé zařízení a robot. Poté můžeme jednoduše ukládat informace, kde a v jaký čas se daný kus nacházel. Traceabilita může sloužit k vyhledávání informací o výrobě v případě statistického vyhodnocení výroby za definované období nebo reklamace ,ale také k ověření zda produkt nepřeskočil nějaký výrobní proces. V dnešní době se jedná o nutnost v každé moderní výrobní lince.

Diskrétní simulační model, který vytvoříme bude na Traceabilitě založen. Tedy budeme schopni jednoznačně zjistit kdy a kde se daný produkt v lince nacházel.

1.2. Teorie Front

K tvorbě simulačního programu budeme využívat vybrané pojmy a principy z teorie front. Celá teorie je založena na interakci entit (Customer) a obsluhy (Server). Naším úkolem je pomocí serveru obsloužit co nejvíce entit, které čekají na obsloužení ve frontě například v obchodě [6].

1.2.1. Vysvětlení základních principů

V simulačním modelu budou servery reprezentovat jednotlivá zařízení, které zpracovávají produkt. Jsou to například šestiosé roboty, lisy, řezačky, elektronické kontroly, atd. Před těmito stanovišti budeme mít zásobníky ve kterých budou čekat naše produkty na zpracování. Můžeme si lehce představit, že tyto zásobníky reprezentují fronty. Naším úkolem je tedy dostat produkty přes všechny zařízení v co nejmenším čase.

Existují různé typy front. Mezi základní patří FIFO, což je zkratka "First In First Out". Znamená to že produkty které vstoupí do fronty první, budou první obslouženy přesně jako ve fronty v obchodě. Druhým běžně užívaným typem je LIFO, značící "Last In First Out". V tomto případě se dostane na řadu první ten produkt, který do fronty vstoupil první. Tento případ můžeme přirovnat k hromadě papírů, kterou zpracovává úředník. Vždy vezme horní papír na hromadě, tedy ten který do systému vstoupil poslední.

Dalším typem fronty je PRI, kde každá entita ve frontě má prioritu a nejprve je obsloužena entita s nejvyšší prioritou. Posledním fronta o které se zmíníme bude SIRO. V tomto případě jsou entity z fronty voleny náhodně pomocí náhodného rozdělení pravděpodobnosti. [6]

1.2.2. Ilustrativní příklad

Máme Linku s jedním zařízením. Do linky vstupují produkty rychlostí 18ks/hod. Zařízení má výrobní kapacitu 25ks/hod. Známe tedy intenzitu vstupu $\lambda = 18$ a intenzitu obsluhy $\mu = 25$. První úkol bude zjistit průměrný čas $\bar{T}$, který stráví produkt v systému (lince). Ten spočteme ze vzorce:

$$\bar{T} = \frac{1}{\mu - \lambda} = 0,146hod = 8,6min.$$

Potom určíme průměrný čas $\bar{T_f}$, který stráví produkt ve frontě na obsluhu:

$$\bar{T_f} = \bar{T} - \frac{1}{\mu} = \frac{1}{\mu - \lambda} - \frac{1}{\mu} = -\frac{\lambda}{\mu(\mu - \lambda)} = 0,103hod = 6,2min.$$

Další určíme průměrný počet kusů v systému $\bar{N}$:

$$\bar{N} = \frac{\lambda}{\mu - \lambda} = 2,57ks.$$

Poslední spočteme průměrný počet kusů ve frontě $\bar{N_f}$:

$$\bar{N_f} = \frac{\lambda^2}{\mu(\mu - \lambda)} = 1,85ks.$$

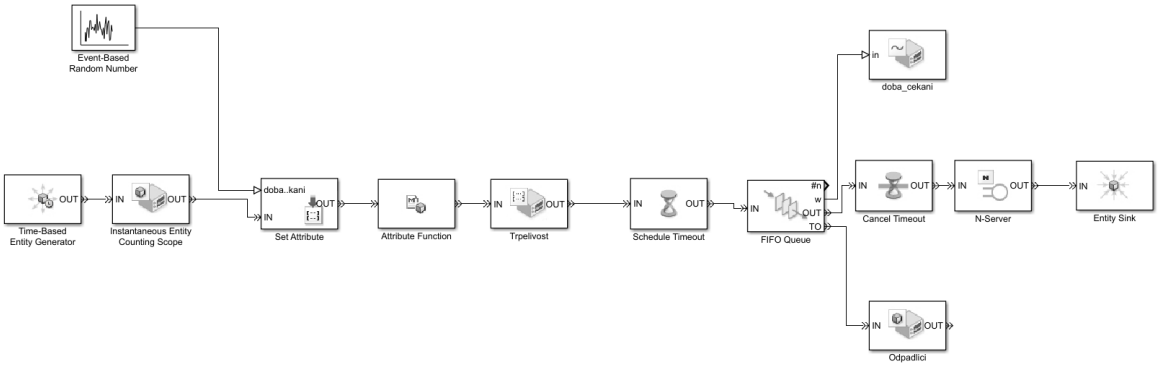
2. Simulační model

V této kapitole se bude zabývat výstavbou simulačního modelu, který popisuje tok produktů výrobní linkou s určitým nastavením. Pomocí heuristické optimalizace se poté pokusíme určit nejlepší nastavení linky, tedy minimální čas za který vyrobí daný počet produktů. V první části si ukážeme že takovýto model je možné vytvořit i pomocí modulu SimEvents v MatLabu. Toto prostředí nám potom poslouží jako verifikace vlastního modelu. Za zmínku stojí i další simulační programy jako Arena a Siemens Smart Factory.

2.1. MatLab SimEvents

SimEvents je nástroj pro diskrétní simulaci událostí vyvinutý společností MathWorks. Přidává knihovnu grafických stavebních bloků pro modelování systémů front do prostředí Simulink. Přidává také simulační modul založený na událostech k časovému simulačnímu modulu v Simulinku.

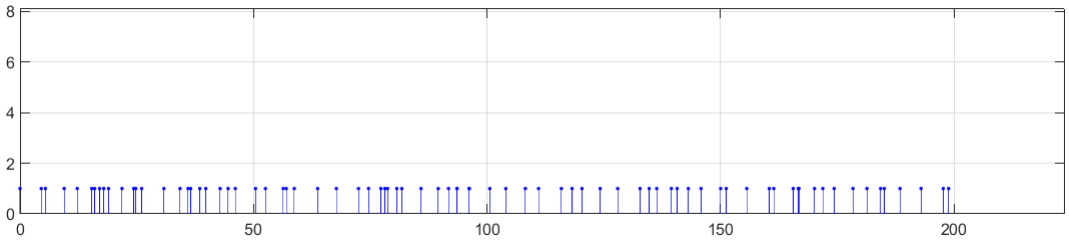
Díky jednoduchému grafickému prostředí, kde skládáme odpovídající bloky do schématy jsme schopni relativně jednoduše sestrojit simulační model. Funkcionalitu SimEvents si ukážeme na jednoduchém příkladu výrobní linky.



Obrázek 2.1: Schéma simulace výrobní linky

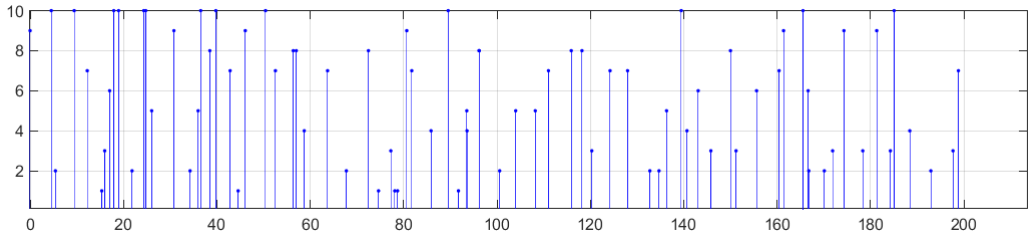
V tomto modelu popíšeme jednotlivé bloky zleva. První blok je generátor entit (produktů). Entity do systému vstupují s rovnoměrným rozdělením. Druhý blok slouží jak čítač generovaných entit, abychom věděli kolik entit vstoupilo do systému. další 2 z bloků přiřazují každé entitě náhodné číslo mezi čísly 1 a 10, které označuje maximální trvanlivost entity. To je čas po kterou vydrží entita čekat na obsluhu než dojde k její expiraci. V praxi bychom tuto entitu mohli popsat jako produkt, kterém zasychá lepidlo a musí být včas zpracován. Pokud se tak nestane, produkt je vyřazen se systému. Pátý blok slouží pouze k vykreslení grafu trvanlivosti jednotlivých entit. V šestém bloku nastavujeme timeout, tedy dobu pro výstup entity ze systému kvůli dlouhému čekání. Další blok už označuje samotnou FIFO frontu, ve které entity čekají na obsluhu. Z tohoto bloku zapisujeme entity opouštějící systém kvůli trvanlivosti pomocí spodního bloku. Horní blok zapisuje doby čekání jednotlivých entit do grafu. Pokud se entita dostane na řadu, projde blokem na zrušení nastavení "timeoutu"a dojde k bloku samotné obsluhy. Po obsloužení přejde do konečného kontejneru na entity.

Nyní když jsme si popsali tvorbu simulačního modelu v prostředí SimEvents nezbývá nic jiného než ho spustit a podívat se na výsledky.
V prvním grafu se můžeme podívat v kterých časech do systému vstupovali jednotlivé entity. Vzpomeňme že entity jsou generovány s normálním rozdělením.



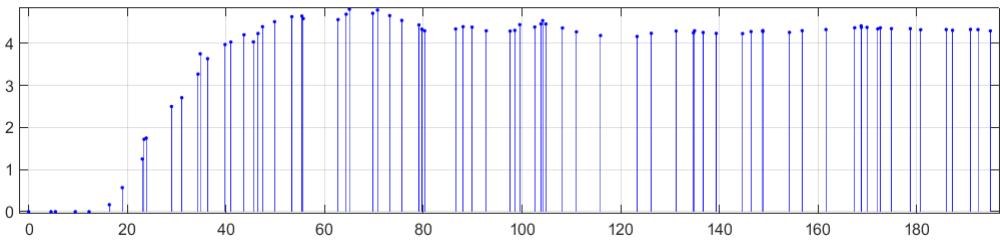
Obrázek 2.2: Počet vygenerovaných entit v čase

V druhém grafu se podíváme na trvanlivosti entit vygenerovaných v čase na odpovídající pozici z grafu minulého.



Obrázek 2.3: Trvanlivosti jednotlivých entit

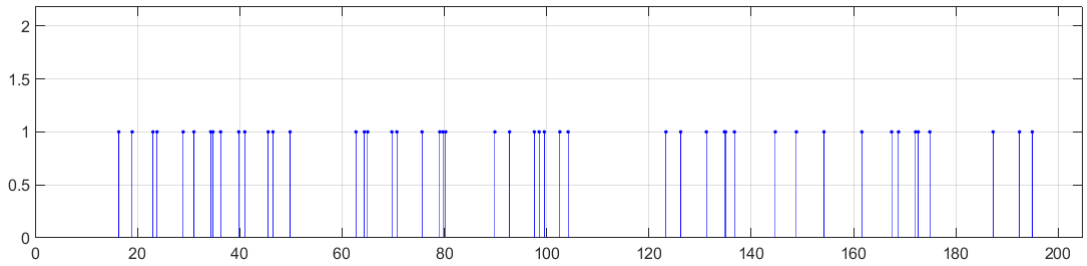
Nyní se dostáváme k samotné FIFO frontě. Podíváme se na doby čekání jednotlivých entit z předchozích grafů.



Obrázek 2.4: Doba čekání ve frontě

V tomto grafu můžeme vidět určitý náběh na dobu čekání přes 4 časové jednotky a následné ustálení. Tedy tím že je fronta na začátku prázdná, jsou entity obslouženy dřív. Po naplnění fronty se časy opět stabilizují do rovnovážného stavu.

V posledním grafu se podíváme na odchody entit ze systému kvůli překročení doby trvanlivosti.



Obrázek 2.5: Entity odcházející ze systému kvůli vypršení trvanlivosti

Jak můžeme vidět, tak SimEvents je velice silný a univerzální nástroj pro diskrétní simulace. Asi největší nevýhodou je samotná cena modulu, která se pohybuje v řádu statisíců. Dále je ke spuštění třeba naistalovaný MatLab a Simulink, což způsobuje špatnou přenositelnost. Také tu není možnost navázat na předchozí simulaci. Proto se v následující kapitole pokusíme vytvořit vlastní diskrétní model jenom pomoci skriptu v Matlabu.

2.2. Vlastní simulační model

V našem modelu budeme chtít simulovat průchod produktů skrz výrobní linku, která se bude skládat z jednotlivých zařízení. Přenos mezi těmito zařízeními budeme řešit buď pomocí zásobníku, tedy fronty a nebo šesti-osým robotem. Zařízení a fronty mají podobné parametry. Proto využijeme možnost objektově orientovaného programovaní v prostředí Matlab. V následující kapitole popíšeme jednotlivé třídy.

2.2.1. Jednotlivé užité struktury a jejich popis

První třídu označíme GOODS. Popisuje jednotlivé produkty prostupující linkou. Každý produkt má následující atributy:

- *SN* - Seriové číslo, které jednoznačně označuje produkt.
- *Model* - Slouží pro rozpoznání typu produktu při výrobě více modelů.
- *Trace* - Je matice do které zapisujeme zařízení a čas, ve kterém produkt daným zařízením prošel.

Tabulka 2.1: Tabulka Traceability

Legenda	Zápis č.1	Zápis č.2	Zápis č.3	Zápis č.4
ID zařízení	1	2	3	4
Čas vstupu do zařízení	4	13	27	52
Čas výstupu ze zařízení	9	22	45	59

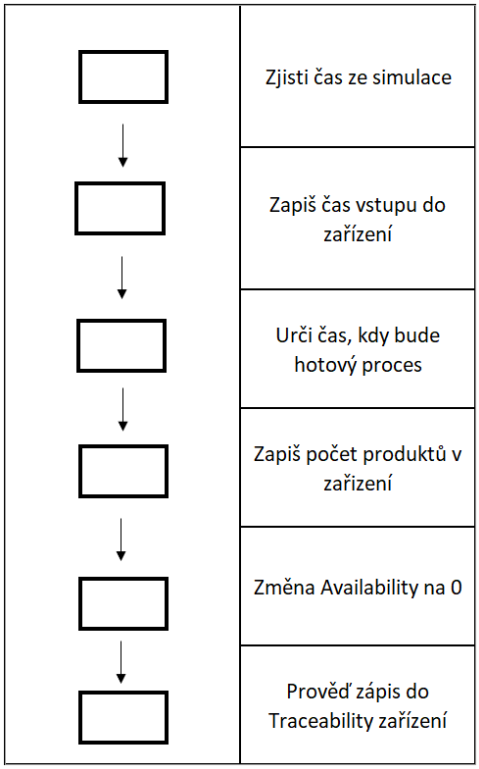
- *ID* - Slouží jako identifikátor rozpracovanosti produktu. Na začátku linky má *ID* nula a pokud projde například devíti zařízeními, bude *ID* rovno devíti. Hlavní využití tato proměnná má při obsluze šestiosým robotem. Robot díky ní pozná, které stanoviště má následovat.

Druhou třídu označíme STATION Bude popisovat zařízení a zásobníky v lince. Opět si popíšeme jednotlivé atributy:

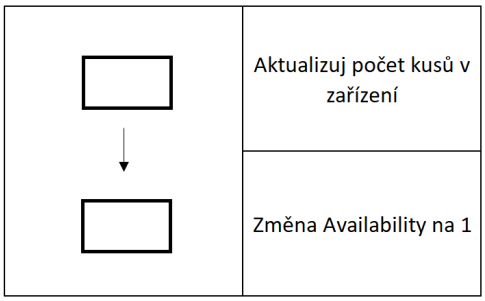
- *Name* - Označuje název daného zařízení
- *ID* - Proměnná která se zapisuje do ID produktu při průchodu zařízením.
- *ProcessTime* - Čas, který zařízení potřebuje na zpracování produktu. U front je nulový.
- *Capacity* - Značí počet produktů které může zařízení nebo fronta pojmout.
- *Occupied* - Počet produktů aktuálně se nacházejících se v zařízení nebo frontě.
- *Availability* - Pokud je zařízení připraveno na výrobu tak je rovna jedné. Pokud jsou v zařízení produkty a další nemůže zpracovávat, je rovna nule
- *Batch* - Pokud zařízení musí produkty zpracovávat v dávkách, například po dvou nebo třech kusech, značí tato proměnná onu dávku.
- *Tout* - Čas ve kterém má produkt opustit zařízení, protože je hotový proces.

- *Tent* - Čas ve kterém produkt vstoupil do zařízení a zahájil proces.
- *Trace* - Matice do které zapisujeme *Tent* a *Tout* jednotlivých produktů procházejících zařízením. Dále taky zapisuje *ID* produktu, tedy fázi zpracování. To slouží pro kontrolu přeskočení stanoviště.

Součástí třídy STATION jsou i 2 procedury na vstup a výstup produktu ze zařízení nebo fronty. Proceduru pro vstup nazveme ENTER. Nejprve určíme *Tent* pouze odečtením času ze simulace, potom $Tout = Tent + ProcessTime$. Následně se provede zápis do Traceability zařízení i produktu a do *Occupied* zařízení se přičte počet vstupujících produktů. Poslední krok je změna *Availability* zařízení. Při výstupu využijeme proceduru označenou LEAVE. Zde se pouze odečte počet vystupujících kusů ze zařízení z *Occupied* a změní se *Availability*.



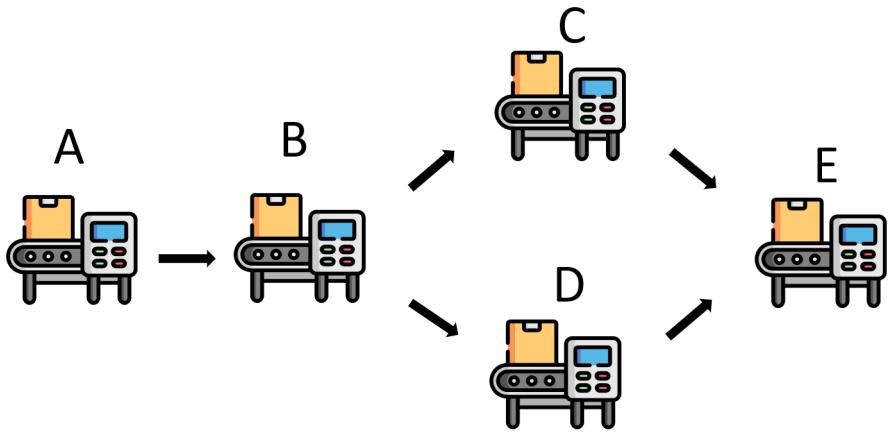
Obrázek 2.6: Flow Chart procedury ENTER



Obrázek 2.7: Flow Chart procedury LEAVE

2.2.2. Incidenční matice

Nyní již víme z jakých struktur budeme utvářet naší linku. Potřebujeme z daných zařízení tedy instancí třídy STATION vytvořit nějakou posloupnost reprezentující danou výrobní linku. Vezmeme si tedy modelovou linku. Můžeme vidět že jde o velice jednoduchou výrobu



Obrázek 2.8: Modelová Výrobní linka

s jedním zdvojeným zařízením. V první řadě vytvoříme vektor zařízení ST, na který se budeme později odkazovat.

$$ST = (A, B, C, D, E)$$

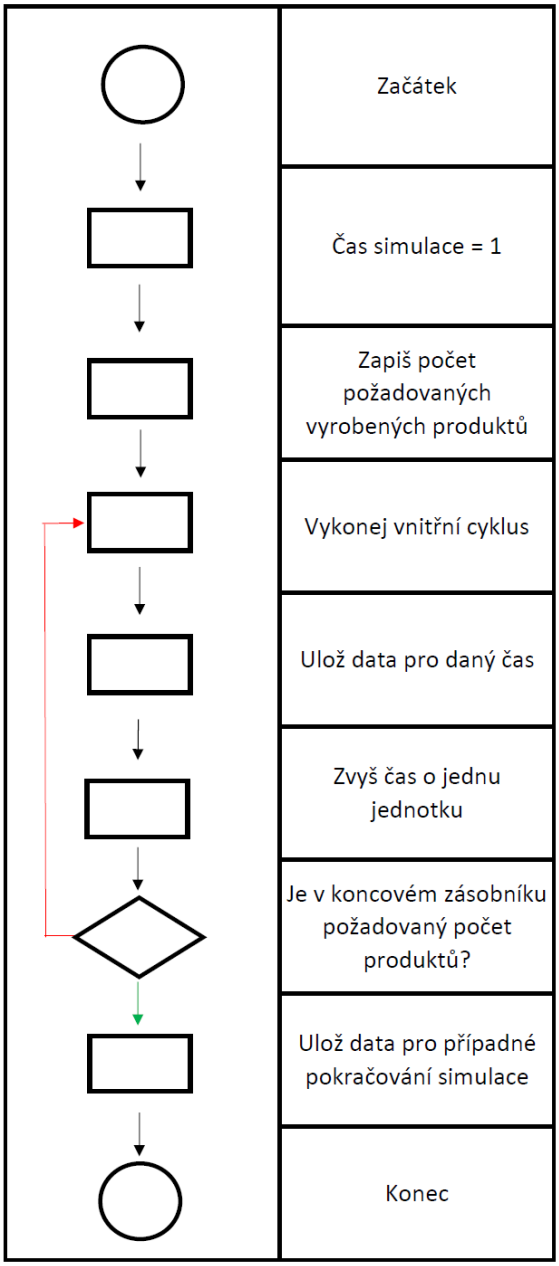
Nyní chceme jednoznačně popsat z jakého zařízení má produkt vystoupit a do kterého následně vstoupit. K tomu slouží Incidenční matice M, kde $(m_{i,j}) = 1$ značí že produkt vystupující z i-tého zařízení vstupuje do j-tého, kde koeficienty i, j odpovídají pořadí zařízení ve vektoru ST. Můžeme zde vidět analogií s orientovaným grafem který představuje schéma linky a jeho maticí, kterou reprezentuje incidenční matice.

$$M = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Tuto matici budeme procházet postupně po řádcích od prvního řádku. Jakmile se dostaneme do pozice $(m_{1,2}) = 1$, dostaneme že ze zařízení A můžeme přejít do zařízení B. Další postup je analogický.

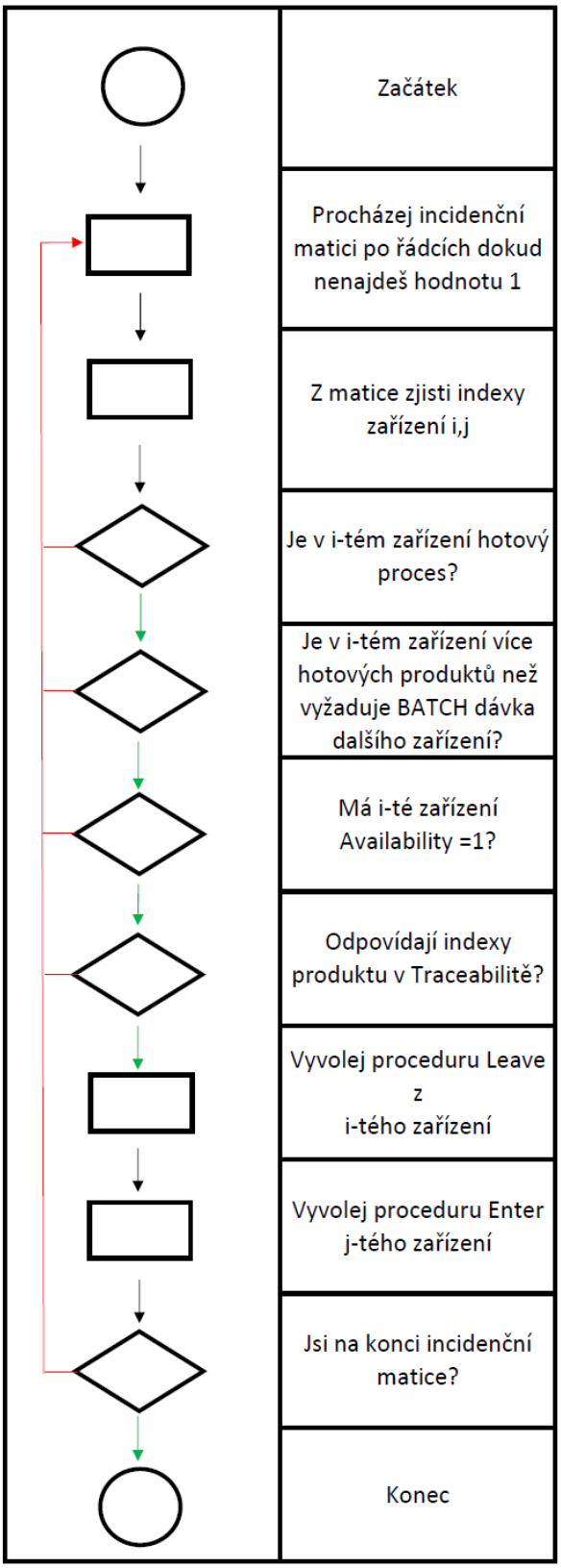
2.2.3. Flow Chart simulace

Máme tedy nějakou posloupnost zařízení, kterou má produkt projít. Potřebujeme navrhnout nějaký algoritmus, aby přechod mezi zařízeními byl uskutečněn pouze pokud nastanou vhodné podmínky v daném čase.



Obrázek 2.9: Flow Chart Simulace

V dalším obrázku se podíváme na vnitřní cyklus simulace, kde dochází k samotnému průběhu přes incidenční matici a ověřování podmínek pro přechod produktu.



Obrázek 2.10: Flow Chart vnitřního cyklu simulace

Výhodou tohoto algoritmu oproti MatLab SimEvents je to, že po skončení simulace zůstanou uložena poslední data o každém za zařízení a je tedy možné plynule navázat na další časový úsek.

Další užitečná vlastnost, která byla z Flow vynechána kvůli jednoduchosti je přeskakování časových úseků, kde nenastanou žádné akce. K tomu využijeme proměnnou $Tout$, kterou má každá instance třídy STATION. Na konci simulace v daném čase jednoduše vybereme nejmenší z časů $Tout$ každého zařízení a označíme jej $Tout_{min}$. Poté tento čas porovnáme s aktuálním časem T simulace. Pokud je $Tout_{min} > T + 1$ tak místo zvýšení T o jednotku času rovnou zvýšíme na $Tout_{min}$. Tímto způsobem jsme schopni ušetřit mnoho cyklů, kdy procházíme dvojitým cyklem velkou matici.

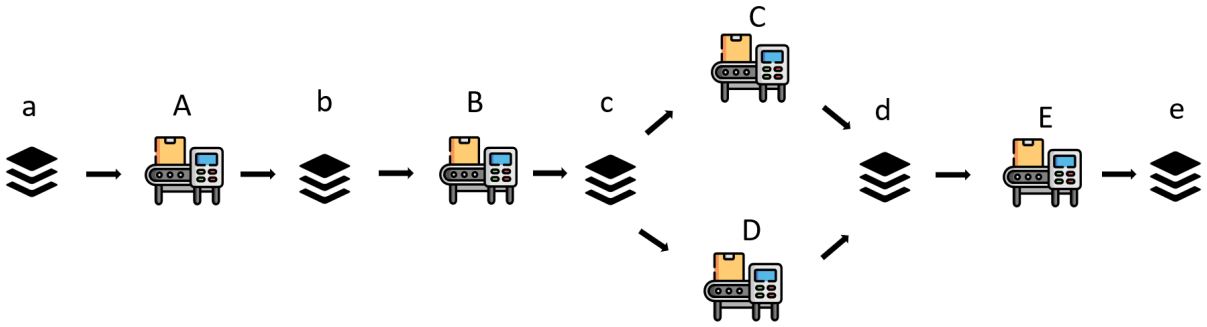
2.2.4. Modelová výrobní linka bez 6-axis robota

Dalším krokem bude vytvořit model výrobní linky, kde je pohyb mezi zařízeními vykonáván automatickými zásobníky. Tyto zásobníky reprezentují fronty. Jedná se také o instanci třídy STATION, ovšem co ji odlišuje od zařízení je to že její $ProcessTime = 0$. To znamená, že produkt může ze zásobníku vystoupit ihned. Zavedeme si tedy nový symbol pro zásobník.

Obrázek 2.11: Symbol zásobníku

Schéma a tok produktu

Jak již bylo řečeno, naše linka se bude skládat ze zařízení a zásobníků. Zařízení budeme označovat velkými písmeny a zásobníky malými.



Obrázek 2.12: Modelová linka s automatickými zásobníky

Pro danou linku potřebujeme vektor zařízení ST a incidenční matici, kterou nyní označíme FAC. Zvolíme se tedy vektor zařízení takto:

$$ST = (a, A, b, B, c, C, d, D, e, E).$$

Z tohoto vektoru a schématu 2.12 reprezentující orientovaný graf dostaneme incidenční matici:

$$FAC = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Víme tedy jak bude vypadat tok produktu skrz výrobní linku. Dále potřebujeme vědět vlastnosti a parametry jednotlivých zařízení a front z kapitoly 2.2.1. Údaje jsou zapsány v následující tabulce.

Tabulka 2.2: Tabulka parametrů stanic výrobní linky

Legenda	a	A	b	B	c	C	d	D	e	E
Name	101	001	102	002	103	003	104	004	105	005
Capacity	1000	1	50	3	60	1	50	2	1000	1
Batch	1	1	1	3	1	1	1	2	1	1
ProcessTime	1	4	1	9	1	30	1	20	1	7

Výsledky simulace

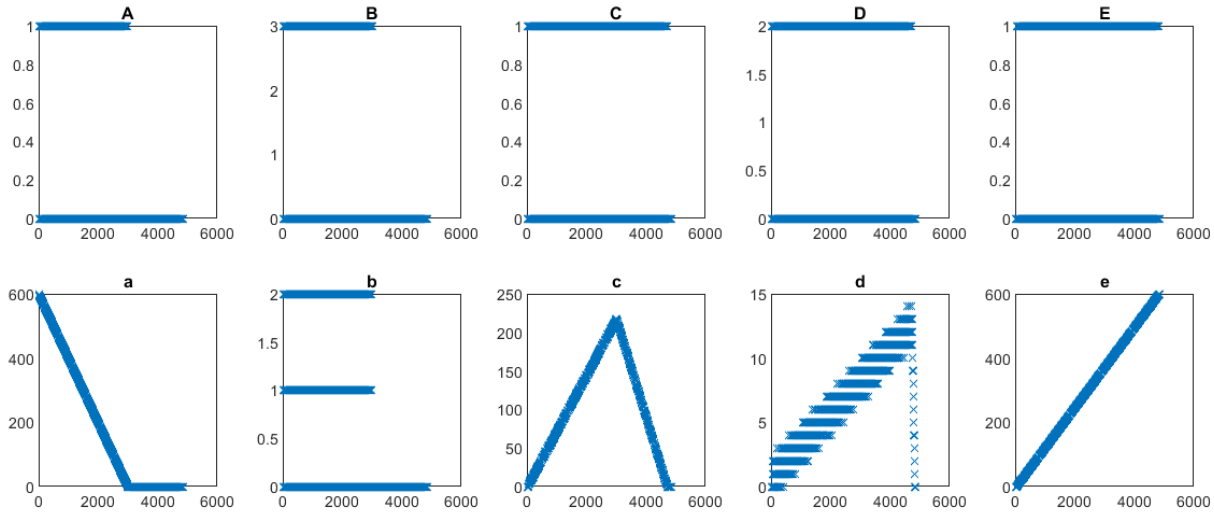
Nyní se musíme zamyslet, jakým způsobem budeme chtít reprezentovat výsledky simulace. Jako první nás bude zajímat čas výrobní dávky T. Dalším faktorem bude vytíženost jednotlivých zařízení k každém časovém okamžiku. Budeme tedy ukládat data v každém čase do tabulky, kde bude zaznamenán počet produktů v zařízení nebo zásobníku v každém časovém kroku.

Tabulka 2.3: Tabulka vytíženosti stanovišť

Station/T	1	2	1000	1001	1002	3000	3001	3020	4841	4842
a	599	599	400	399	399	0	0	0	0	0
A	1	1	0	1	1	0	0	0	0	0
b	0	0	2	2	2	0	0	0	0	0
B	0	0	0	0	0	3	3	0	0	0
c	0	0	72	72	72	216	216	217	0	0
C	0	0	1	1	1	1	1	1	0	0
d	0	0	3	3	3	8	8	8	0	0
D	0	0	2	2	2	2	2	2	0	0
e	0	0	119	119	120	369	369	372	599	600
E	0	0	1	1	0	1	1	0	1	0

Tabulka má tedy T sloupců. Pro ilustraci jsme vypsali tabulku jen pro náhodně zvolené časy. Můžeme vidět že výrobní dávka 600ks byla dokončena za 4842 časových jednotek, protože v tomto čase vstoupil poslední produkt do posledního zásobníku, tedy $T = 4842$.

Výsledky můžeme jednoduše znázornit v grafu závislosti proměnné *Occupied* každé stanice na čase T.



Obrázek 2.13: Závislost *Occupied* na T každé stanice

Díky této vizualizaci dat vidíme která zařízení tvoří tzv. bottleneck, což je nejpomalejší místo výroby. Z grafu je patrné že fronta se tvoří v zásobníku c, tedy před zařízeními C,D. Dostáváme se tedy k závěru, že buď musíme zrychlit zařízení C nebo D. Další možností by bylo přidat další zařízení paralelně.

Dalším využití simulačního modelu najdeme u tvorby tvz. Man-Machine Chart, která se standartně tvoří ručně nebo s pomocí videozáznamů. Tvorba MM Chartu je velice zdlouhává a dá se v ní jednoduše ztratit. Proto je mnohem jednodušší vložit linku do modelu a vytvářet simulace pro různá nastavení.

2.2.5. Modelová výrobní linka s 6-axis robotem

Dalším linka kterou budeme modelovat bude mít stejná zařízení, ale přesun mezi nimi nebudou zajišťovat zásobníky, nýbrž jeden šestiosý robot.

Obrázek 2.14: Symbol šestiosého robota

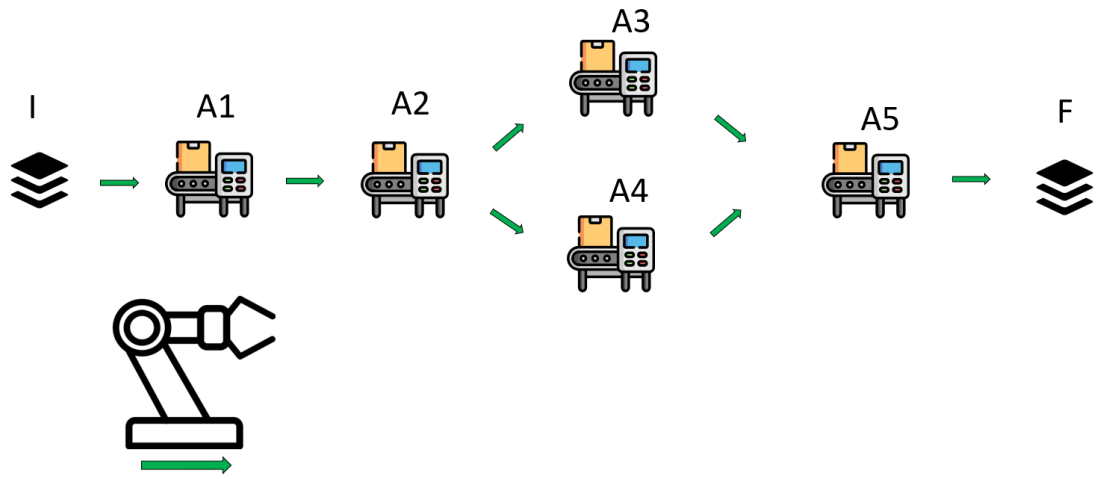
V tomto modelu budeme potřebovat plně funkční Traceabilitu. Protože každý produkt vstupuje do robota opakovaně, musíme mu jednoznačně říct, do kterého zařízení má daný produkt vložit. K tomu poslouží proměnná ID produktu popsáná v kapitole 2.2.1. Díky ní robot pozná v jaké fázi zpracovaní produkt je a vloží je do následující stanice. Poslední odlišnost od normálního zařízení je ten, že se mění časy zpracování $ProcessTime$ v závislosti na tom, ze kterého zařízení daný produkt vyjímá. Časy zapíšeme do vektoru

$$rTime = (time1, time2, time3, ...),$$

kde čas na první pozici odpovídá času ne přenesení z prvního zařízení vektoru ST, druhý čas odpovídá času přenesení z druhého zařízení atd.

Schéma a tok produktu

Výrobní linka bude mít pouze 2 zásobníky. První na začátku výroby, kde bude na začátku nachystaná výrobní dávka. Označíme jej *Initial*, zkratka *I*. Druhý zásobník bude na hotové produkty na konci linky. Označíme jej *Finish*, zkratka *F*. Všechny přesuny mezi stanicemi jsou vykonávány šestiosým robotem. V grafu tyto přesuny označíme zelenou šipkou. Můžeme tedy uvést schéma výrobní linky.



Obrázek 2.15: Modelová linka s šestiosým robotem

Vektor stanic, bude bez šestiosého robota, protože víme, každý přenos mezi stanicemi vykonává právě on. Vektor tedy zavedeme následně:

$$ST = (I, A1, A2, A3, A4, A5, F).$$

K tomuto vektoru a schématu vytvoříme incidenční matici. Díky absenci zásobníků bude matice menší.

$$FAC = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Opět si zapíšeme vlastnosti jednotlivých stanovišť do tabulky. Jako poslední uvedeme šestiosého robota kterého označíme R. Robot nemá pevně daný *ProcessTime* ani *ID*. Proto tyto proměnné označíme křížkem.

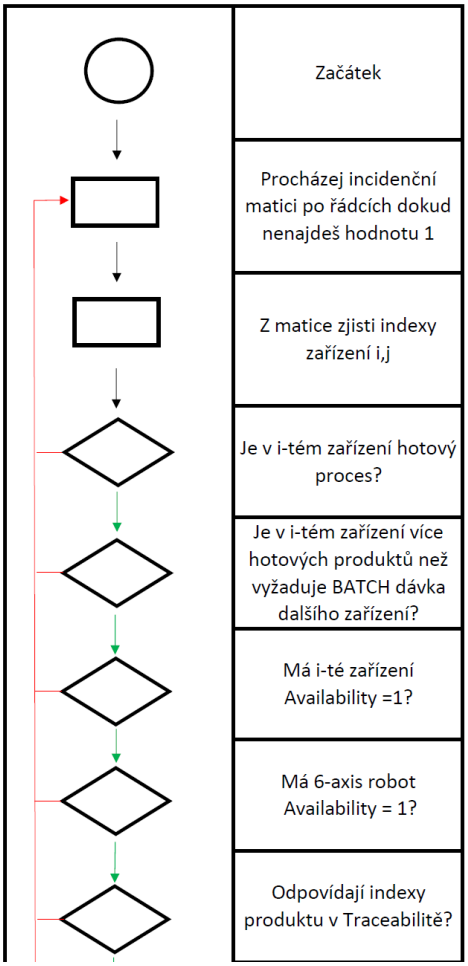
Tabulka 2.4: Parametry stanic pro linku se šestiosým robotem

Legenda	I	A1	A2	A3	A4	A5	F	R
Name	101	001	002	003	004	005	999	501
Capacity	1000	1	1	1	1	1	1000	1
ProcessTime	0	5	9	18	19	7	0	x
ID	1	2	3	4	5	6	7	x

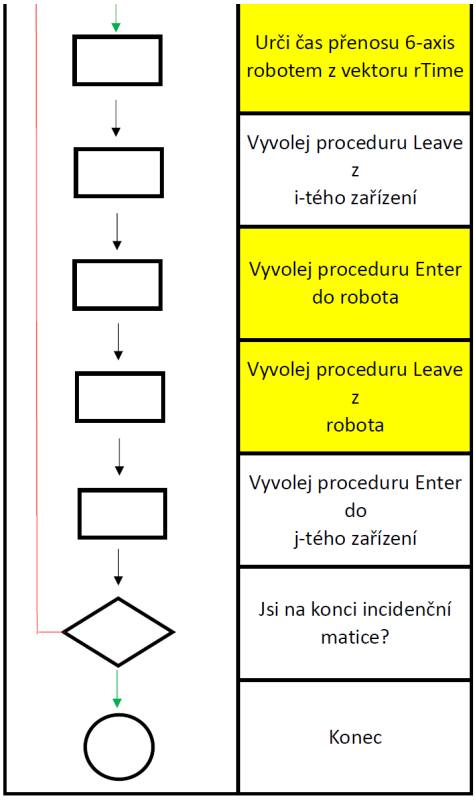
Jako poslední potřebujeme zavést vektor $rTime$ odpovídající vektoru ST. Jak již bylo řečeno čas na i-té pozici popisuje čas přenosu z i-té stanice vektoru ST.

$$rTime = (3, 4, 5, 6, 7, 8)$$

Dalším krokem bude úprava samotného vnitřního cyklu simulace. Potřebujeme přidat kroky na zjištění času přenosu pomocí traceability a vstup a výstup z robota. Tyto kroky jsou v novém Flow Chart zvýrazněny žlutě.



Obrázek 2.16: První část Flow

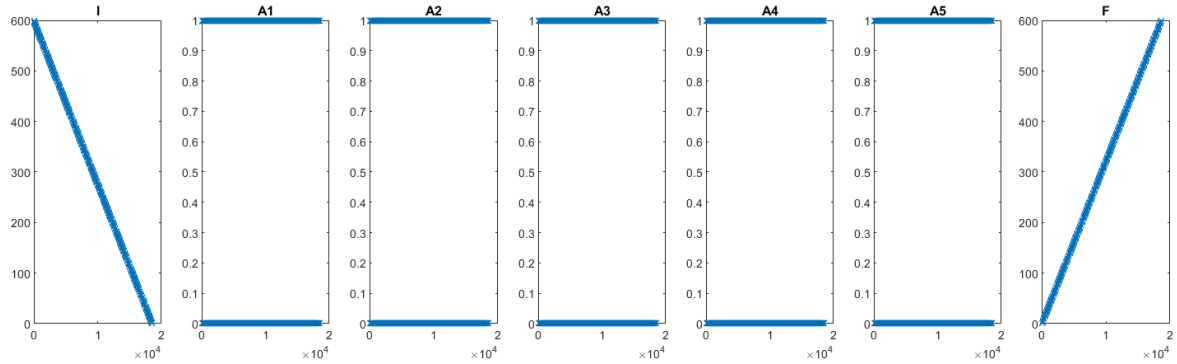


Obrázek 2.17: Druhá část Flow

Traceabilitu zapisujeme analogicky z popisu v kapitole 2.2.1. Výsledné zápisy si ukážeme v další podkapitole. Nyní zbývá spustit simulace a podívat se na výsledky. Opět použijeme výrobní dávku 600 kusů.

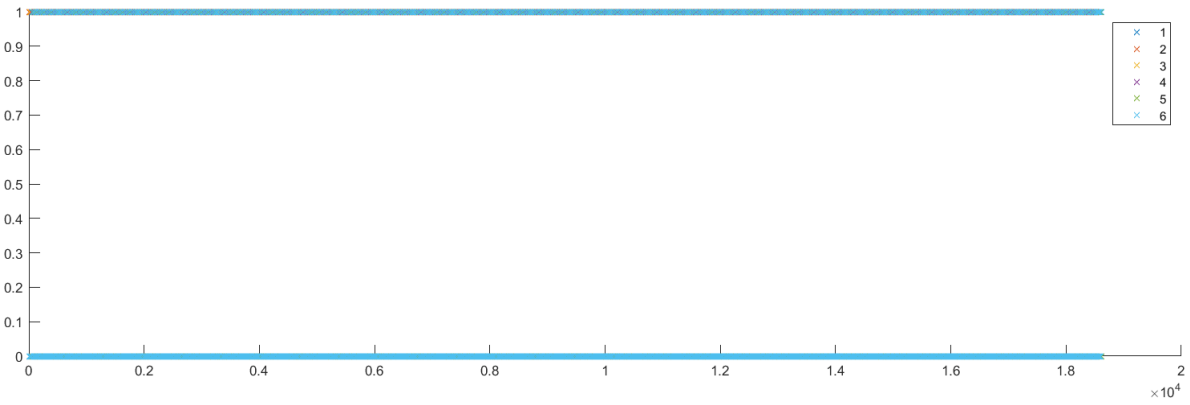
2.2.6. Výsledky simulace bez logiky

Nejprve se podíváme na obsazenost jednotlivých zařízení v čase. Výsledky pro každý časový okamžik jsou zaznamenány v tabulce. Z té nyní vytvoříme graf podobně jako v předchozí části.



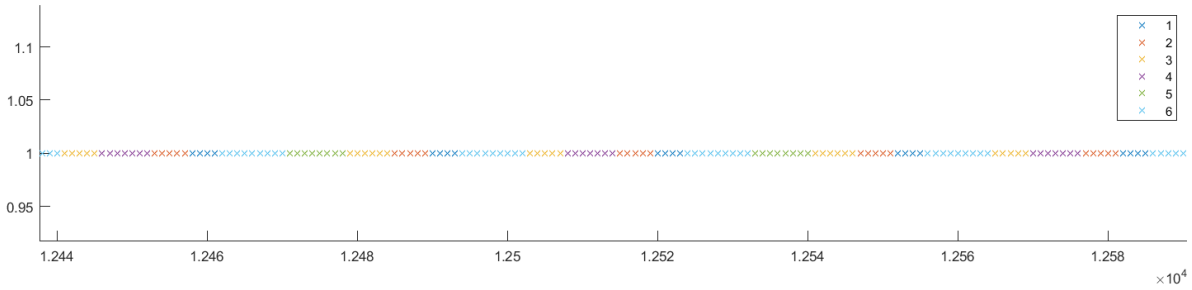
Obrázek 2.18: Závislost *Occupied* na T každé stanice

V tomto grafu nefiguruje šestiosý robot. Proto si vykreslíme graf ve kterém bude barevně rozlišeno u kterého zařízení robot v daný okamžik byl.



Obrázek 2.19: Závislost *Occupied* robota na T

V grafu není na první pohled patrný rozdíl v barvách. Je to zapříčiněno dlouhým trváním simulace a tedy malým měřítkem grafu. Nyní si ukážeme výřez grafu 2.19, kde již dokážeme rozlišit jednotky času.



Obrázek 2.20: Detail závislosti *Occupied* robota na T

Nyní si ukážeme zápisy z Traceability. V první tabulce se podíváme na historii prvních 20 kusů v zařízení A2. Můžeme si všimnou že simulace probíhala v pořádku, protože kusy

Tabulka 2.5: Traceabilita zařízení A2

SN	Tin	$Tout$	ID
001	13	22	2
002	32	41	2
003	57	66	2
004	89	98	2
005	119	128	2
006	151	160	2
007	181	190	2
008	213	222	2
009	243	252	2
010	275	284	2
011	305	314	2
012	337	346	2
013	367	376	2
014	399	408	2
015	429	438	2
016	461	470	2
017	491	500	2
018	523	532	2
019	553	562	2
020	585	594	2

procházely linkou chronologicky a vždy se správným ID označující míru rozpracovanosti. Nyní se podívejme na traceabilitu robota. Ta je již pestřejší, protože kusy vstupují do robota opakovaně při přenosu mezi zařízeními. Zde můžeme vidět postupně zpracování

Tabulka 2.6: Traceabilita Robota

SN	Tin	$Tout$	ID
001	1	4	1
001	9	13	2
002	14	17	1
001	22	27	3
002	28	32	2
003	33	36	1
002	41	46	3
001	46	52	4
003	53	57	2
004	58	61	1
001	62	70	6

produktu s $SN = 1$, kde se postupně mění ID, při přenosu z každého zařízení. Pokud se podíváme na vytíženost robota, je očividné, že ihned po ukončení jednoho procesu, začíná proces další. U zařízení $A2$ nastává čekání na obsluhu robotem. Vzniká tedy otázka, jestli

je nějaká možnost zrychlit obsluhu robotem. Touto otázkou se budeme zabývat v dalších kapitolách.

Nyní ještě zpět k Traceabilitě. Zbývá nám ukázat jak vypadá tabulka historie jednotlivého produktu. Ukažme si tedy opět zápisy v tabulce. Názvy zařízení odpovídají tabulce v úvodu kapitoly.

Tabulka 2.7: Traceabilita Produktu

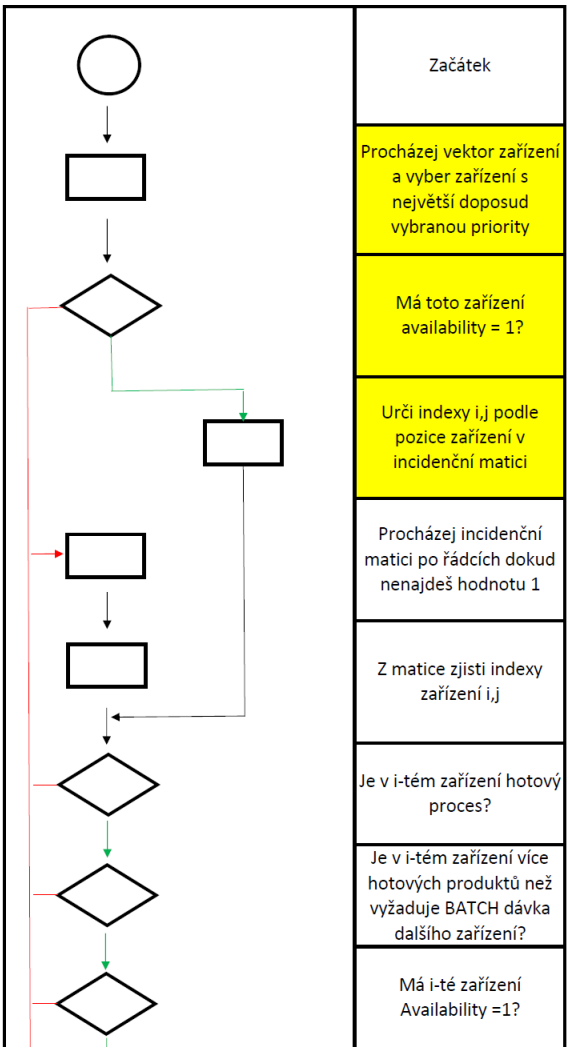
$Name$	101	501	001	501	002	501	003	501	005	501	999
Tin	1	1	4	9	13	22	27	46	52	62	70
$Tout$	1	4	9	13	22	27	45	52	59	70	70

Tímto jsme si popsali všechny základní struktury základního simulačního modelu. Nyní si ukážeme jakým způsobem lze optimalizovat výrobu na této lince.

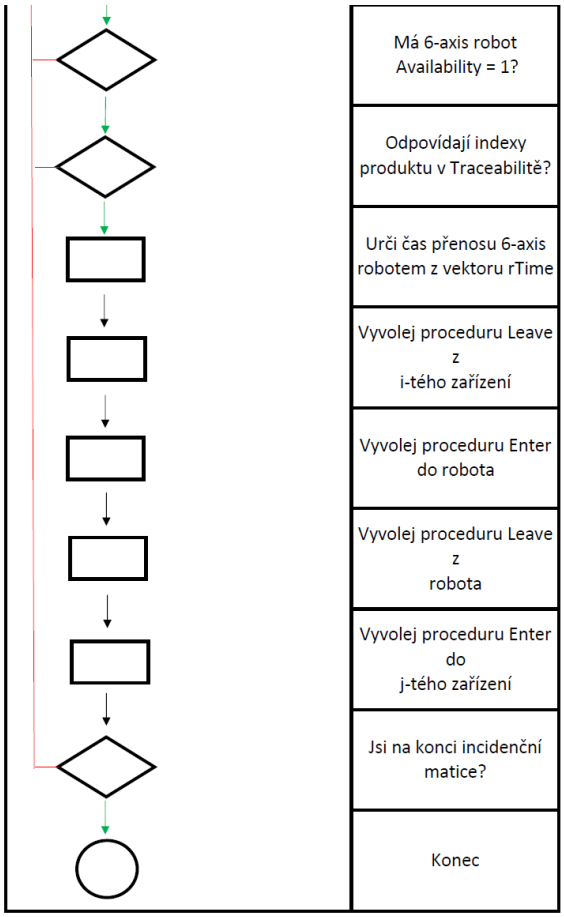
2.2.7. Simulace s logikou

Nevýhodu dosavadního modelu bylo, že při každém časovém okamžiku jsme procházeli zařízení postupně od začátku linky. Tedy pokud byla 2 zařízení volná k obsloužení robotem, bylo vybráno první v pořadí z nich. To vždy nemusí být cesta k optimálnímu výstupu. Zavedeme si tedy proměnnou *Priority*, kterou bude mít každé zařízení. Toto číslo bude udávat, jak důležité je dané zařízení obsloužit. Budou-li volná 2 zařízení k obsluze, vybere robot vždy to, které bude mít větší *Priority*.

Budeme tedy muset upravit Flow-Chart celé simulace, aby program dokázal na tuto proměnnou reagovat. Změna bude obnášet úpravu vnitřního cyklu, kde musíme přidat kroky na ověření největší *Priority*. Tyto kroky jsou opět označeny žlutě.



Obrázek 2.21: První část Flow



Obrázek 2.22: Druhá část Flow

Výsledky simulace pro různé logiky robota

Nyní máme funkční model výrobní linky s logikou priorit. Každá změna v logice by měla mít nějaký vliv na celkový výstup linky. Proto provedeme několik simulací pro různé priority. Porovnávací kritérium bude čas zpracování výrobní dávky čítající 600 kusů. Opět si ukážeme nastavení zařízení v tabulce, kde nyní bude navíc hodnota proměnné *Priority*.

Tabulka 2.8: Parametry zařízení s proměnnou *Priority*

Legenda	I	A1	A2	A3	A4	A5	F	R
Name	101	001	002	003	004	005	999	501
Capacity	1000	1	1	1	1	1	1000	1
ProcessTime	0	5	9	18	19	7	0	x
ID	1	2	3	4	5	6	7	x
Priority	1	1	1	1	1	1	1	1

Pro první simulaci, kterou budeme používat jako Benchmark použijeme základní nastavení kde žádné zařízení nemá prioritu. Pro takové nastavení linky dostáváme čas simulace:

$$T = 18730.$$

Vyzkoušíme několik nastavení priorit a pro každé nastavení a následně porovnáme časy simulací.

Tabulka 2.9: Různé nastavení systému priorit a jejich výsledky

Simulace č.	I	A1	A2	A3	A4	A5	F	R
1	1	1	5	1	1	4	1	1
2	1	5	2	2	2	4	1	1
3	1	4	3	2	2	2	1	1
4	1	1	5	1	1	3	1	1
5	1	1	4	4	4	3	1	1

Simulace č.	Čas simulace
1	18028
2	18924
3	52301
4	18028
5	18326

Můžeme vidět že simulace trvají okolo 18000 časových jednotek. V některých kombinacích priorit může nastat problém, kdy délka simulace je příliš dlouhá a může celý program zastavit. Proto si pro příští simulace nastavíme podmínku maximálního času, abychom zamezili ztrátě dat. Díky variabilitě časů simulací způsobených změnou priorit se budeme zabývat optimalizací parametrů v kapitole 3.

2.2.8. Digitální dvojče reálné linky

V této části se budeme zabývat tvorbou digitální kopie reálné výrobní linky. Data z reálné výroby poskytla mezinárodní společnost Alps Alpine, která má závod v Sebranicích u Boskovic. Tato firma vznikla v poválečném Japonsku a nyní se zabývá především tvorbou interierových dílů pro automotive.

ALPS ALPINE GROUP

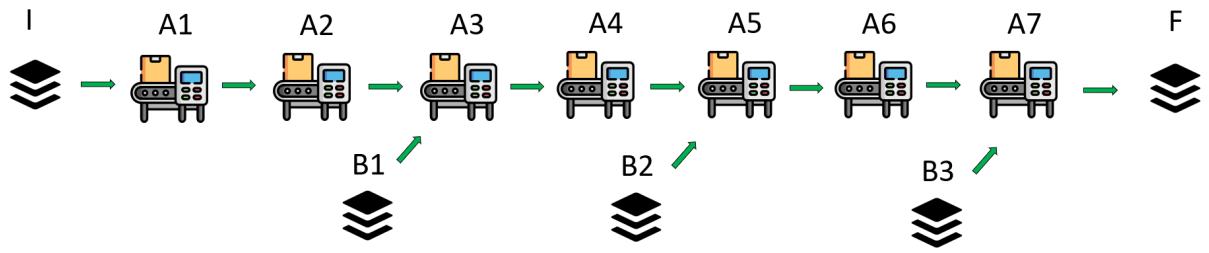
Obrázek 2.23: Logo společnosti

Obecně můžeme výrobu popsat jako tvorbu desky tištěného spoje na SMT a poté přesun na Assembly linku, kde se elektronika osadí pohledovými díly, se kterými pak interaguje uživatel vozidla.

Zaměříme se na Assembly linku, protože v této části výroby se nachází nejvíce procesů jako lisování, šroubování a jiné. Tedy je tu největší možnost zlepšení a úspor při správném návrhu linky. K dispozici dostaneme data linky, která je již v seriové produkci. Tudíž nebudeme navrhovat novou výrobu ale pouze se pokusíme co nejvíce přiblížit výstupem naší modelové linky k výstupu reálné výroby. Takto vytvořené digitální dvojče může sloužit pro experimentální implementaci změn a ověření jejich efektu, bez rizika zastavení výroby.

Schéma linky a data

Podívejme se nejprve na schéma linky. Linka se složena ze sedmi zařízení, kde každé zastává jiný proces. Komplikaci oproti modelovým linkám představují zásobníky B1, B2 a B3. Ty představují podsestavy přicházející z externích přípravných linek. Produkty z nich tedy nemusí být vždy k dispozici pro finální assembly linku, kterou modelujeme. Z pozorování provedených přímo na lince můžeme říct, že situace kdy přípravná linka nestihne vyrobit podsestavu pro finální linku nastane maximálně jednou za směnu. Tyto přípravné linky tedy aproximujeme zásobníkem, ve kterém budeme podsestava vždy k dispozici pro přenesení šestiosým robotem do odpovídajícího zařízení.



Obrázek 2.24: Výrobní linka v Alps Alpine

Incidenční matice této linky bude připomínat jednotkovou matici s posunutou diagonálou, protože všechny zařízení jsou v sérií a zásobníky B1, B2, B3 můžeme díky naši aproximaci vynechat.

$$FAC = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Dále se zaměříme na vlastnosti jednotlivých zařízení. Data jsou zapsány v tabulce, jako v předchozích kapitolách.

Tabulka 2.10: Parametry zařízení pro reálnou linku

Legenda	I	A1	A2	A3	A4	A5	A6	A7	F	R
Name	101	001	002	003	004	005	006	007	999	501
Capacity	1000	5	5	5	1	1	1	1	1000	1
ProcessTime	0	0	85	147	39.2	15	40.7	29	0	x
ID	1	2	3	4	5	6	7	8	9	x

$$rTime = \big(10,5 \quad 7,5 \quad 7,5 \quad 2,4 \quad 12,6 \quad 7,5 \quad 15,2 \quad 8,1\big).$$

Tady ovšem narážíme na problém s implementací zařízení do modelu. Model s šestiosým robotem v současnosti neumí manipulovat s více než jedním produktem. V tabulce vidíme, že zařízení A1,A2,A3 zpracovávají díly v dávkách 5 kusů. Protože se snažíme simulovat stejný výstup linky, můžeme si dovolit udělat další aproximaci. Jednotlivé časy podělíme velikostí dávky zařízení. Tím dostaneme průměrný čas na výrobu jednoho dílu, který ve výsledné simulaci můžeme použít. Stejný princip použijeme i na odpovídající hodnoty vektoru $rTime$. Tyto nové hodnoty neovlivní konečný výsledek simulace. Nové parametry opět zapíšeme do tabulky.

Tabulka 2.11: Parametry zařízení aproximované pro reálnou linku

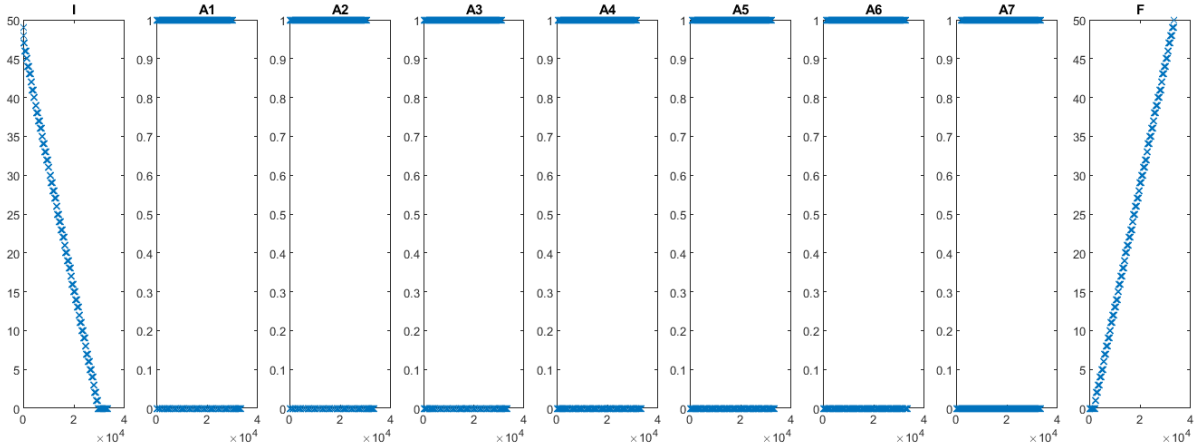
Legenda	I	A1	A2	A3	A4	A5	A6	A7	F	R
Name	101	001	002	003	004	005	006	007	999	501
Capacity	1000	1	1	1	1	1	1	1	1000	1
ProcessTime	0	0	17	29.4	39.2	15	40.7	29	0	x
ID	1	2	3	4	5	6	7	8	9	x

$$rTime = \big(2,1 \quad 1,5 \quad 1,5 \quad 2,4 \quad 12,6 \quad 7,5 \quad 15,2 \quad 8,1\big).$$

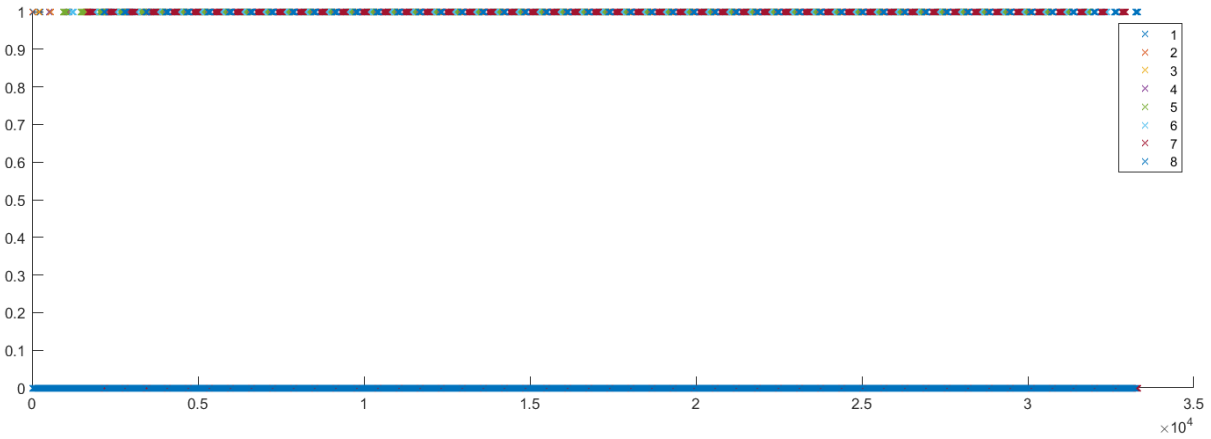
Simulace a validace modelu s reálnými daty

Nyní máme data vhodná pro náš simulační model. Nyní chceme ověřit zda výstup našeho modelu odpovídá výstupu skutečné linky. Hlavním porovnávacím kritériem bude Cycle Time linky. Termín označuje čas, za který z linky vystoupí hotový produkt. Cycle Time reálné linky je 63,5s.

Spustíme tedy simulaci s nastavením které jsme určili v předchozí podkapitole. Na ověření nám postačí menší dávka například 50 kusů. Cycle Time je stejný v průběhu celé výroby. Podívejme se na grafy ze simulace.



Obrázek 2.25: Závislost *Occupied* na čase



Obrázek 2.26: Závislost *Occupied* robota na čase

Z grafů je patrné, že simulace proběhla v pořádku pro celou dávku 50 kusů. Nyní potřebujeme zjistil Cycle Time linky. Ten můžeme ověřit několika způsoby. Nejjednodušší bude podívat se do Traceability zásobníku na konci linky. Několik zápisů z Traceability ukážeme v tabulce.

Tabulka 2.12: Část traceability zásobníku na konci linky

SN	Tin	$Tout$	ID
40	2697.8	2697.8	8
41	2761.3	2761.3	8
42	2824.8	2824.8	8
43	2888.3	2888.3	8
44	2951.8	2951.8	8
45	3015.3	3015.3	8
46	3078.8	3078.8	8
47	3142.3	3142.3	8
48	3205.8	3205.8	8
49	3269.3	3269.3	8
50	3332.8	3332.8	8

Pro výpočet Cycle time použijeme hodnoty Tin označující vstup daného produktu do zásobníku hotových produktů. Nyní stačí vzít nějakou hodnotu Tin libovolného produktu a od této hodnoty odečíst Tin produktu předchozího.

$$CycleTime(SN) = Tin(SN) - Tin(SN - 1)$$

Tímto výpočtem určíme jednotlivé Cycle Time pro každý produkt. Díky tomu že linka vyrábí v periodické ustálené sekvenci, jsou Cycle Time pro jednotlivé produkty stejné. Můžeme tedy říct že Cycle Time linky je:

$$CycleTime = 63,5s.$$

Tato hodnota odpovídá reálnému výstupu linky ve výrobní hale. Tedy dokázali jsme platnost a funkčnost našeho diskrétního simulačního modelu na reálných výrobních datech.

Na této lince nebudeme realizovat simulaci s logikou a její následnou optimalizací. Linka je již optimalizovaná pro sériovou výrobu a zavedením systému priorit bychom již nedosáhli významného zlepšení. Tyto zlepšení se nejlépe provádí při návrhu nových linek. K datům o nových výrobách bohužel nemáme přístup a proto se budeme věnovat optimalizaci na modelových linkách.

3. Optimalizace

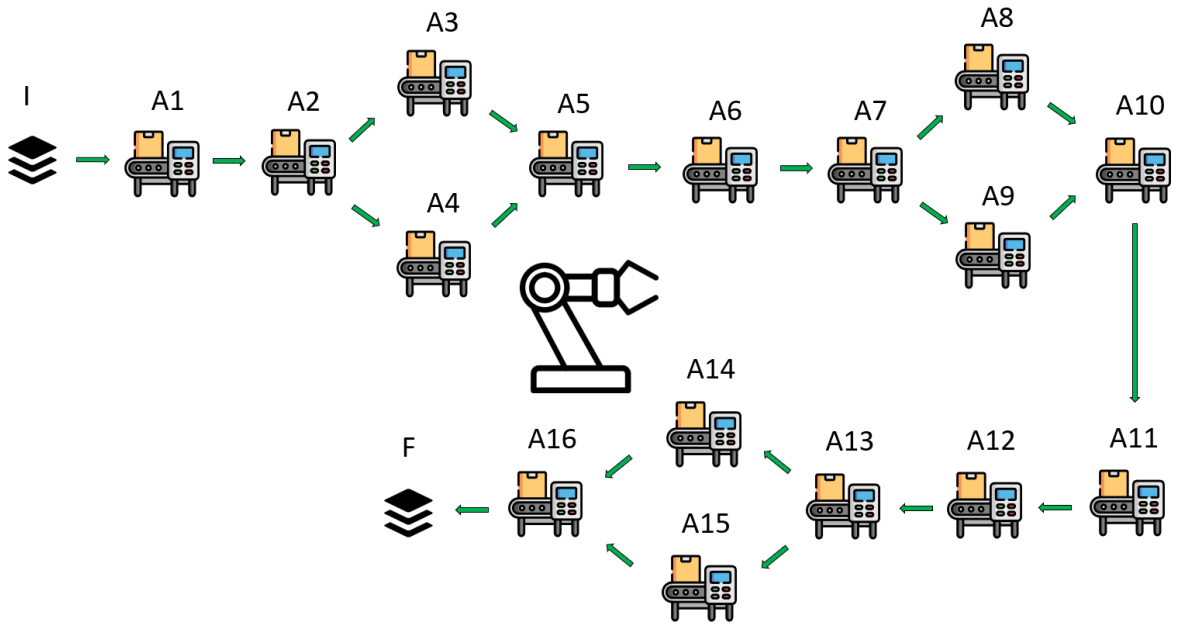
V této fázi máme funkční simulační model. V tomto modelu můžeme pro jednotlivá zařízení volit priority a touto volbou potencionálně změnit výstup celé výroby.

3.1. Cíl optimalizace

Jedním z cílů práce je optimalizovat výrobní proces z hlediska minimalizace výrobních časů. Tedy chceme za daný počet výrobků zpracovat v co možná nejkratším čase. Musíme si naformulovat optimalizační úlohu pro nějakou modelovou výrobní linku.

3.1.1. Modelová linka

Linka, na které se budeme určovat optimální systém priorit bude obsahovat více zařízení než předchozí, abychom mohli ukázat potenciál modelu. Ukažme si tedy schéma linky.



Obrázek 3.1: Modelová výrobní linka

Každý pohyb v lince je vykonáván jedním šestiosým robotem. Na začátku linky je zásobník s materiálem na zpracování. Na konci linky je zásobník na hotové produkty. Vektor zařízení bude vypadat následovně:

$$ST = (I, A1, A2, A3, A4, A5, A6, A7, A8, A9, A11, A12, A13, A14, A15, A16, F).$$

K vektoru zařízení zavedeme vektor časů přenosu šestiosým robotem analogicky s předchozími kapitolami.

$$rTime = (130, 240, 310, 410, 1630, 210, 1930, 410, 530, 1800, 310, 410, 540, 1650, 2320, 440, 560)$$

V modelu budeme chtít simulovat náhodnost pohybů robota a proto časy z vektoru $rTime$ nebudou deterministické, ale stochasticky závislé. Budeme jej pro každý pohyb určovat z náhodné proměnné rT s normálním rozdělením:

$$rT_i \sim \mathcal{N}(\mu, \sigma^2),$$

$$\mu = rTime_i,$$
$$\sigma^2 = \frac{rTime_i}{10}.$$

V lince se nachází 16 zařízení. Z pozorování reálné linky víme, že strojní časy jsou velice přesné, tedy pro naši simulaci je ponecháme deterministické. Parametry jednotlivých zařízení jsou zapsány v tabulce:

Tabulka 3.1: Parametry zařízení linky k optimalizaci

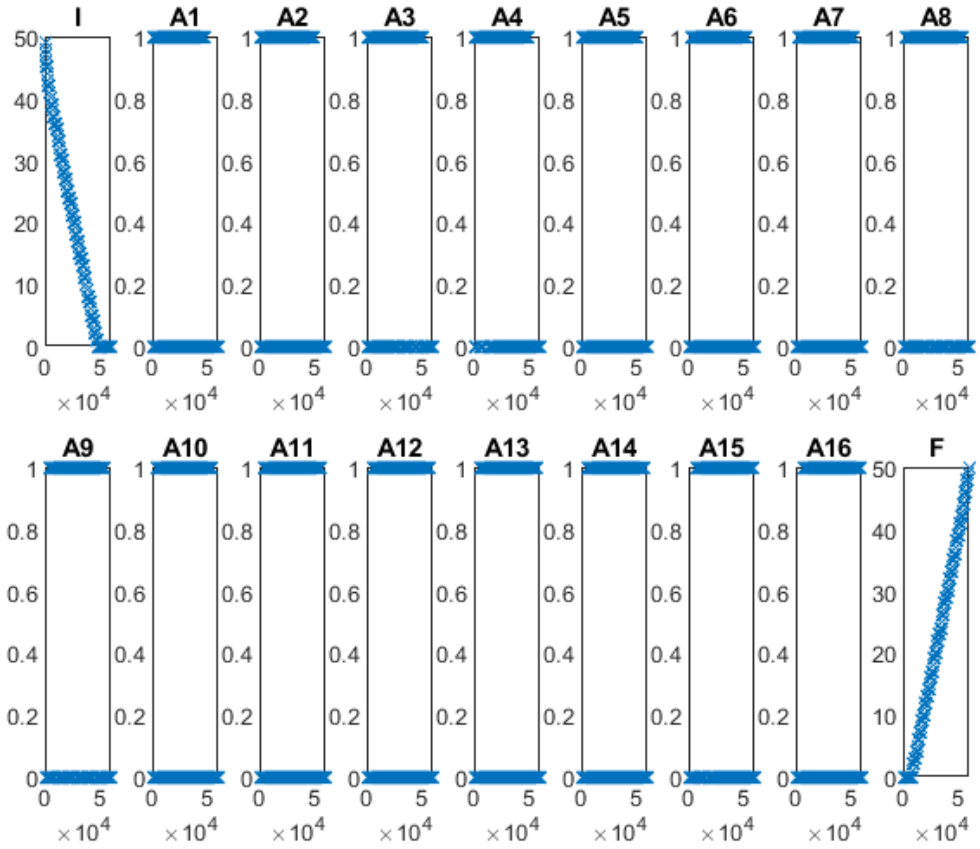
Zařízení	Name	Capacity	Processtime	ID
I	101	5000	0	1
A1	001	1	255	2
A2	002	1	254	3
A3	003	1	614	4
A4	004	1	605	5
A5	005	1	355	6
A6	006	1	354	7
A7	007	1	453	8
A8	008	1	867	9
A9	009	1	847	10
A10	010	1	455	11
A11	011	1	453	12
A12	012	1	452	13
A13	013	1	458	14
A14	014	1	758	15
A15	015	1	826	16
A16	016	1	674	17
F	999	5000	0	18

S takto připraveným modelem můžeme spustit simulaci. Nyní bez jakéhokoliv systému priorit, abychom získali jakýsi základní bod, ze kterého se budeme vycházet při optimalizaci. Našim úkolem je co nejkratší čas výroby, který v simulaci označujeme T. Čas pro simulaci bez logiky byl:

$$T = 528\ \ 743.$$

Oproti tomuto času budeme porovnávat řešení které budeme získávat optimalizačními algoritmy. Nyn9 zbývá ověřit, že simulace proběhla bez potíží.

Abychom ověřili, že simulace proběhla v pořádku, uvedeme graf, kde je vidět že zásobníky se vyprazdňovaly, respektive plnily souvisle po celou dobu výroby 50 kusů.



Obrázek 3.2: Závislost *Occupation* na T simulace bez logiky

3.1.2. Formulace úlohy

V této kapitole si odvodíme zadání optimalizační úlohy. Protože porovnáváme výrobní čas, zvolíme si za naši účelovou funkci čas simulace T. Hodnoty účelové funkce se budou lišit podle systému priorit který zadáme. Priority zadáme ve vektoru PR, kde i-tá pozice odpovídá hodnotě i-tého zařízení vektoru ST. Vektor tedy bude mít tvar:

$$PR = (pr_1, pr_2, pr_3, ..., pr_{16}, pr_{17}, pr_{18}),$$

Definujme, že priorita zásobníků I a F je jedna. Nemá důvod prioritně obsluhovat zásobník který má nulový $ProcessTime$, tedy:

$$pr_1 = 1,$$

$$pr_{18} = 1.$$

Nakonec si zavedeme podmínku že *Priority* zařízení bude v intervalu $\langle 1; 16\rangle$. Toto omezení vychází z faktu, že máme 16 zařízení v modelové lince. Omezíme se pouze na celá čísla z důvodu jednoduché implementace. Nyní můžeme zapsat celou úlohu do klasického tvaru.

$$min \quad T$$

$$pr_i \in Z$$

$$pr_i \in \langle 1; 16\rangle$$

$$pr_1 = pr_{18} = 1$$

Tuto úlohu nelze řešit klasickými optimalizačními algoritmy, protože funkce není spojitá a nemáme žádné informace o konvexnosti. V dalších kapitolách se budeme zabývat hledáním heuristického, tedy přibližného minima funkce T.

3.2. Heuristika a její implementace

Heuristika (z řečtiny heuriskó – nalézt, objevit) znamená zkusmé řešení problémů, pro něž neznáme algoritmus nebo přesnější metodu. Heuristické řešení je často jen přibližné, založené na poučeném odhadu, intuici, zkušenosti nebo prostě na zdravém rozumu. První odhad se může postupně zlepšovat, i když heuristika nikdy nezaručuje nejlepší řešení. Zato je univerzálně použitelná, jednoduchá a rychlá [9].

V naší úloze odhadneme řešení pomocí náhodně generovaných systémů priorit. Prvním alogoritmem, který zkusíme implementovat bude takzvaný "Random search", což je jedna z nejjednodušších heuristik. Na to navážeme sofistikovanější metodou zvaná "Diferenciální evoluce", která vyhází z evolučních algoritmů.

3.2.1. Velikost stavového prostoru

Pokud se podíváme na zadání naší optimalizační úlohy, zjistíme že můžeme určit velice jednoduše celkový počet řešení. Složky vektoru PR mohou nabývat pouze přirozených čísel z intervalu $\langle 1; 16\rangle$. Z toho plyne, že máme 16 možností na výběr na každou pozici vektoru. Samotný vektor PR má délku 16. Celkový počet možností volby systém priorit označme $SolCount$. Hodnotu určíme pomocí variací s opakováním.

$$SolCount = V'(16, 16) = 16^{16} = 18\ \ 446\ \ 744\ \ 073\ \ 709\ \ 551\ \ 616$$

Z důvodu nedostatečného výpočetního výkonu nejsme schopni projít všechny řešení a určit tak jednoznačné řešení. Proto se dále zaměříme na implementaci heuristik a nalezení přibližného řešení.

3.2.2. Random Search

Random search algoritmus by se dal označit jako jedna z nejjednodušších heuristik. Její princip spočívá ve vygenerování náhodného řešení a porovnání hodnoty účelové funkce s dosavadním nejlepším výsledkem. Pokud je řešení lepší než předchozí nejlepší výsledek, stává se toto řešení novým nejlepším a algoritmus pokračuje dokud nedosáhne stop kritéria.

Tato metoda má velice jednoduchou implementaci. Nevýhodou je nestrukturované procházení prostoru řešení. Z předchozího výpočtu nepřenášíme žádnou informaci do dalšího řešení. To může způsobit pomalou konvergenci protože "náhodně skáčeme"stavovým prostorem.

V našem případě použijeme jako stop kritérium maximální počet iterací, kde jedna iterace odpovídá jedné simulaci výrobní dávky. Dále potřebujeme zadat nějaké počáteční nejlepší řešení a výchozí hodnotu účelové funkce T.

Pseudokód Random Search

Nastav počáteční hodnotu účelové funkce $T^0 = \infty$
Inicializuj vektor PR_{best}.
Nastav počet simulací $SimCount$.

while $i < SimCount$ **do**

for j = 1 to length PR_{best} **do**
Na j-tou pozici vektoru PR vygeneruj náhodné číslo pr_i takové, že:

$$pr_i \in Z$$

$$pr_i \in \langle 1; 16 \rangle$$

end for
Proveď simulaci s prioritami danými vektorem PR
Ulož čas simulace T^{t+1}
Urči řešení iterace podle předpisů:

$$PR_{best}^{t+1} = \begin{cases} PR & \text{pokud } T^{t+1} < T^t \\ PR_{best}^t & \text{jinak} \end{cases}$$

$$T^{t+1} = \begin{cases} T^{t+1} & \text{pokud } T^{t+1} < T^t \\ T^t & \text{jinak} \end{cases}$$

end while

Tento kód je pouze ilustrativní. V reálném skriptu se musí inicializovat všechny stroje, včetně šestiosého robota a velikosti výrobní dávky, která bude čítat 50 kusů.

Připomeňme si výsledky proti kterým budeme model porovnávat. Jednalo se simulaci bez logiky. Důsledkem toho, je že hodnota *Priority* u všech zařízení bude rovna jedné.

$$PR = (1, 1, 1, ..., 1, 1, 1)$$

$$T = 528743$$

Tímto je algoritmus připraven a můžeme spustit skript pro nalezení heuristického řešení. Po provedení 200 iterací algoritmu jsme dostaly tyto výsledky.

$$PR = (1, 12, 16, 11, 3, 15, 2, 3, 14, 2, 9, 4, 8, 6, 15, 10, 16, 1)$$

$$T = 496069$$

Tento výsledek znamená úsporu asi 5%, což není zanedbatelná hodnota vzhledem k tomu, že náklady na změnu linky nebo storjních zařízení jsou nulové. Stačí pouze určit priority zařízení, což ve standartních výrobních linkách není problém.

Nyní ověříme, že výsledná optimalizovaná simulace proběhla v pořádku. Podíváme se do traceability posledního zásobníku F, abychom se ujistili, že výroba probíhala souvisle.

Tabulka 3.2: Traceabilita zásobníku na konci linky po Random Search optimalozaci

SN	Tin	ID		SN	Tin	ID
1	33378	17		26	281427	17
2	43029	17		27	291148	17
3	52480	17		28	300314	17
4	62780	17		29	310077	17
5	72857	17		30	319886	17
6	82591	17		31	329832	17
7	92366	17		32	340135	17
8	102253	17		33	349906	17
9	111998	17		34	360249	17
10	122285	17		35	369789	17
11	132393	17		36	379102	17
12	142298	17		37	388511	17
13	151811	17		38	397998	17
14	161707	17		39	407430	17
15	171271	17		40	417371	17
16	181583	17		41	427899	17
17	191011	17		42	438035	17
18	201033	17		43	447240	17
19	211548	17		44	456851	17
20	221688	17		45	466047	17
21	231157	17		46	475018	17
22	241372	17		47	481320	17
23	251910	17		48	487241	17
24	261723	17		49	492304	17
25	271626	17		50	496125	17

Simulace proběhla v pořádku, kde po najetí výroby byl cycletime $CT = 10582$. Random Search nám dal relativně dobré řešení. Pokusíme se využít sofistikovanější heuristiku, která najde buď lepší řešení nebo najde podobné řešení rychleji.

3.2.3. Diferenciální evoluce

Další heuristika, kterou implementujeme na náš model bude diferenciální evoluce. Díky využití sofistikovanější metody hledání optimálního řešení bychom měli snížit hodnotu účelové funkce, tedy času simulace. Nebo najít stejné řešení jako v případě Random Search, ale v menším počtu iterací.

Diferenciální evoluce byla vyvinuta v roce 1997 vědci R.Storn a K.Price. Jedná se o vektorový metaheuristický algoritmus, který má podobné vlastnosti s genetickými algoritmy. Používá se zde mutace a křížení jednotlivých řešení. Diferenciální evoluce je stochastický algortimus, který nevyužívá derivace [10,11,12].

Úvodní nastavení algoritmu, je takové že počáteční hodnota účelové funkce je nekonečně velká, tedy $T^0 = \infty$, abychom zajistili že v prvním kroku dojde ke zlepšení, jak bude popsáno dále.

Pro d-dimenzionální problém s d parametry, kde $d = 16$ generovat nějaký počáteční vektor dosavadního nejlepšího výsledku priorit x^t_{best}. Pro počáteční vektor je $t = 0$ a hodnoty generujeme náhodně s rovnoměrným rozdělením, tedy vektor můžeme zapsat ve tvaru:

$$x^0_{best} = (x^0_{best,1}, x^0_{best,2}, x^0_{best,3}, ..., x^0_{best,15}, x^0_{best,16}),$$
$$x^0_{best,i} \sim Ro(1, 16)\,.$$

Pro další výpočet, tedy pro $t \neq 0$, budeme využívat dosavadní nejlepší řešení x^t_{best} jako rodiče pro naši mutaci. Nyní vytvoříme dárcovský vektor v^t, který budeme křížit s vektorem počátečních hodnot. Vektor je popsán následujícím mutačním schématem.

$$v^{t+1} = x^t_{best} + F(x^t_q - x^t_r)$$

Označme část schématu $\delta = F(x^t_q - x^t_r)$ jako pertrubaci vektoru x^t_{best}, tedy část zajišťující odchylku od současného řešení. V tomto schématu se objevuje nová konstanta F, kterou nazveme diferenciální váha. Je to reálné číslo s vlastnosti $F \in [0, 1]$. Vektory x^t_q a x^t_r jsou náhodně generované z rovnoměrného rozdělení.

$$x^t_{q,i} \sim Ro(1, 16)$$

$$x^t_{r,i} \sim Ro(1, 16)$$

Dalším krokem bude nastavit nějaká pravidla pro křížení rodičovského a dárcovského vektoru. Zavedeme si tedy konstantu C_r, kterou nazveme parametr křížení. Popisuje pravděpodobnost s jakou přeskočí hodnota na i-té pozici z dárce do rodiče. Je to tedy reálné číslo a $C_r \in [0, 1]$. Dále zavedeme ke každému k vektoru v^t náhodný vektor r:

$$r = (r_1, r_2, r_3, ..., r_{15}, r_{16}),$$

$$r_i \sim Ro(0, 1),$$
$$r_i \in [0, 1].$$

Vektor r se počítá v každém časovém kroku znovu a informace nepřenášíme do další iterace. Tedy není potřeba ukládat předchozí hodnoty tohoto vektoru. Samotné křížení se provede podle následujícího předpisu:

$$u^{t+1}_i = \begin{cases} v^t_i & \text{pokud } r_i \leq C_r \\ x^t_{best} & \text{jinak.} \end{cases}$$

Tímto dostaneme vektor potomka u^{t+1}_i. Samotný postup diferenciální evoluce nezajistí, že potomek zlepší hodnotu účelové funkce. Je nutné provést simulaci se systémem priorit daným vektorem u^{t+1}_i. Ze simulace dostaneme novou hodnotu účelové funkce T^{t+1}. Tuto hodnotu porovnáme s dosavadní hodnotou T^t a určíme zda jsme provedli zlepšení nebo ne. Celý postup probíhá podle tohoto předpisu:

$$x^{t+1}_{best} = \begin{cases} u^{t+1} & \text{pokud } T^{t+1} < T^t \\ x^t_{best} & \text{jinak.} \end{cases}$$

S tímto vektorem x_{best}^{t+1} se vracíme zpět na začátek smyčky a zahájíme další iteraci.Stop kritérium, které algoritmu řekne, kdy má skončit vyřešíme velice jednoduše. Zavedeme proměnnou NG, která bude označovat počet po sobě jdoucích iterací bez zlepšení. Pokud v iteraci nastane zlepšení, tak proměnnou vynulujeme. Podívejme se nyní na pseudokód popisující heuristiku. Vše vychází z popisu, který byl uveden.

Pseudokód diferenciální evoluce

Nastav počáteční hodnotu účelové funkce $T^0 = \infty$
Inicializuj počáteční vektor x_{best}^0.
Nastav diferenciální váhu F.
Nastav parametr křížení C_r
Nastav proměnnou $NG = 0$ a její maximalní hodnotu NG_{max}

while $NG < NG_{max}$ **do**

Pro vektor x_{best}^t náhodně vyber vektory x_q^t a x_r^t
Vygeneruj nový vektor v^t podle předpisu $v^{t+1} = x_{best}^t + F(x_q^t - x_r^t)$
Vygeneruj náhodně rozdělený vektor $(r, r_i \in [0, 1])$
for i = 1 to d **do**
urči vektor u_i^{t+1} podle předpisu pro každou složku:

$$u_i^{t+1} = \begin{cases} v_i^t & \text{pokud } r_i \leq C_r \\ x_{best,i}^t & \text{pokud } r_i > C_r \end{cases}$$

end for
Proveď simulaci s prioritami danými vektorem u_i^{t+1}
Ulož čas simulace T^{t+1}
Urči řešení iterace podle předpisů:

$$x_{best}^{t+1} = \begin{cases} u^{t+1} & \text{pokud } T^{t+1} < T^t \\ x_{best}^t & \text{jinak} \end{cases}$$

$$T^{t+1} = \begin{cases} T^{t+1} & \text{pokud } T^{t+1} < T^t \\ T^t & \text{jinak} \end{cases}$$

$$NG = \begin{cases} 0 & \text{pokud } T^{t+1} < T^t \\ NG + 1 & \text{jinak} \end{cases}$$

end while

Pro naši optimalizační úlohu nastavíme stop kritérium $NG_{max} = 50$. Tedy po 50-ti iteracích, kdy nenastane zlepšení se algoritmus ukončí.

Před prezentací výsledků se podíváme na předchozí výsledky, které nám budou sloužit pro porovnání. Nejprve jsme provedli simulaci bez logiky:

$$PR = (1, 1, 1, ..., 1, 1, 1),$$

$$T = 528743.$$

Prvního zlepšení jsme dosáhli po implementaci Random Search algoritmu v kombinaci se systémem logiky.

$$PR = (1, 12, 16, 11, 3, 15, 2, 3, 14, 2, 9, 4, 8, 6, 15, 10, 16, 1)$$

$$T = 496126$$

Před spuštěním evoluční diference si musíme určit koeficient váhy F a parametr křížení C_r. Jelikož nyní nemáme žádné povědomí o ideálních hodnotách těchto parametrů, zvolíme je následovně:

$$F = 0,5,$$

$$C_r = 0,5.$$

S takto určenými parametry jsme po provedení simulace a heuristické optimalizace dostali tyto výsledky:

$$PR = (1, 5, 3, 6, 1, 1, 7, 15, 6, 4, 5, 6, 5, 8, 6, 5, 3, 1),$$

$$T = 497823.$$

Toto není zlepšení oproti předchozímu algoritmu. Výsledek byl sice nalezen v méně iteracích, ale chtěli bychom se posunout i ke zlepšení hodnoty účelové funkce. Po experimentálním ověření ideálního nastavení hodnot F a C_r jsme získali nastavení a k němu výsledky optimalizace:

$$F = 0,5,$$

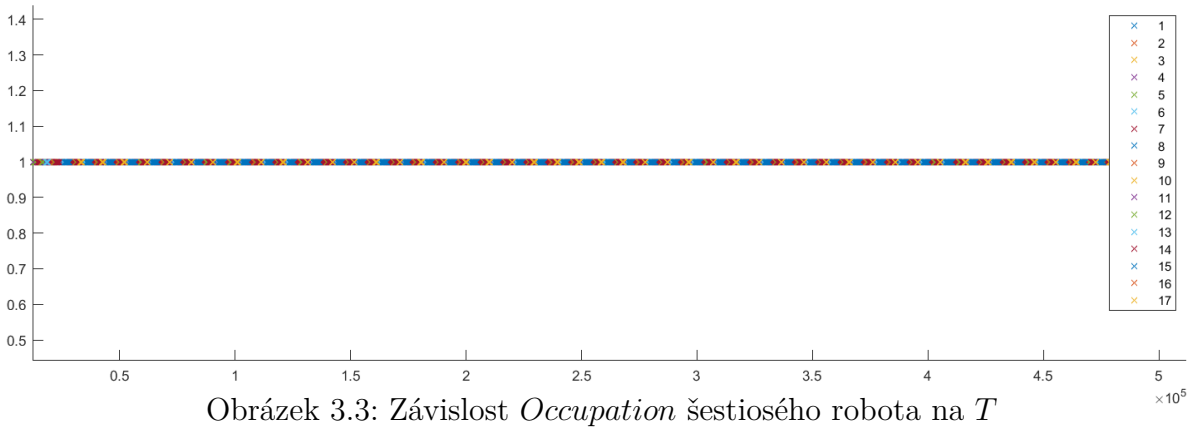
$$C_r = 0,9,$$

$$PR = (1, 7, 8, 7, 1, 5, 1, 1, 1, 7, 3, 10, 3, 12, 7, 8, 9, 1),$$

$$T = 495959.$$

S tímto nastavením jsme dosáhli zlepšení hodnoty účelové funkce i oproti Random Search algoritmu. Můžeme tedy říct že algoritmus diferenční evoluce najde lepší řešení než náhodné prohledávání stavového prostoru.

Zbývá ověřit zda simulace opět proběhly v pořádku. Podíváme se na vytíženost šestiosého robota. Ten by měl být úzkým místem, tedy bottleneckem celé linky. Tedy neměl by mít žádné viditelné prostoje. Druhým údajem pro ověření bude zápis traceability zásobníku F na konci výrobní linky. Data budou zapsána v tabulce jako u ověření Random Search simulace.



Obrázek 3.3: Závislost *Occupation* šestiosého robota na T

Tabulka 3.3: Traceabilita zásobníku na konci linky po optimalizaci

SN	Tin	ID		SN	Tin	ID
1	34076	17		26	281180	17
2	43694	17		27	291076	17
3	53668	17		28	301393	17
4	63751	17		29	311223	17
5	73765	17		30	320839	17
6	83737	17		31	330332	17
7	93606	17		32	340746	17
8	103775	17		33	350915	17
9	113728	17		34	360234	17
10	123652	17		35	370223	17
11	133642	17		36	379883	17
12	144028	17		37	389413	17
13	153968	17		38	399179	17
14	162975	17		39	408889	17
15	172660	17		40	418404	17
16	182371	17		41	428262	17
17	192333	17		42	438204	17
18	202245	17		43	448495	17
19	212241	17		44	457771	17
20	221978	17		45	466455	17
21	231871	17		46	475433	17
22	241741	17		47	481547	17
23	251485	17		48	487210	17
24	261228	17		49	492440	17
25	271402	17		50	495958	17

Z dat můžeme usoudit,že simulace s nejvhodnější logikou proběhla v pořádku a souvisle. Proto můžeme naše výsledky považovat za validní. Shrneme si tedy postupně výsledky našich heuristických optimalizaci.

Jednotlivé výsledky heuristických optimalizací si znázorníme v tabulce. Ve třetím sloupci bude procentuální zlepšení oproti simulaci bez logiky.

Tabulka 3.4: Porovnání výsledků heuristické optimalizace

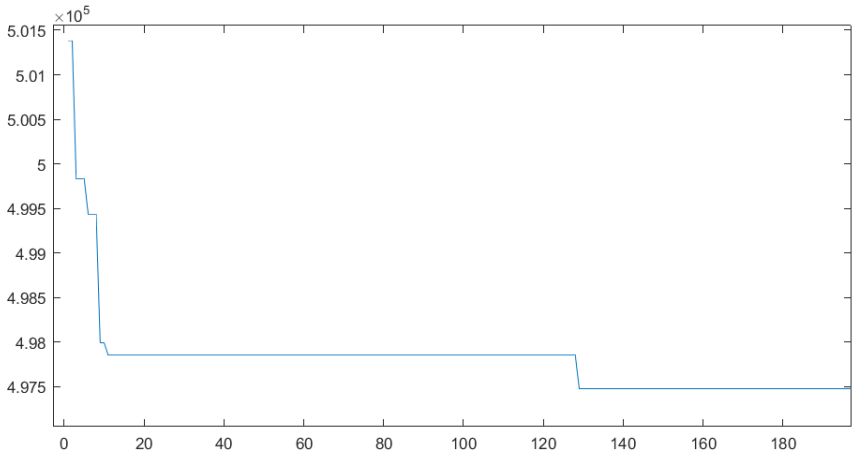
Metoda	Výsledná hodnota T	Zlepšení
Bez logiky	528743	0%
Random Search	496126	6,2%
Diferenciální evoluce	495959	6,4%

3.2.4. Porovnání rychlosti konvergence

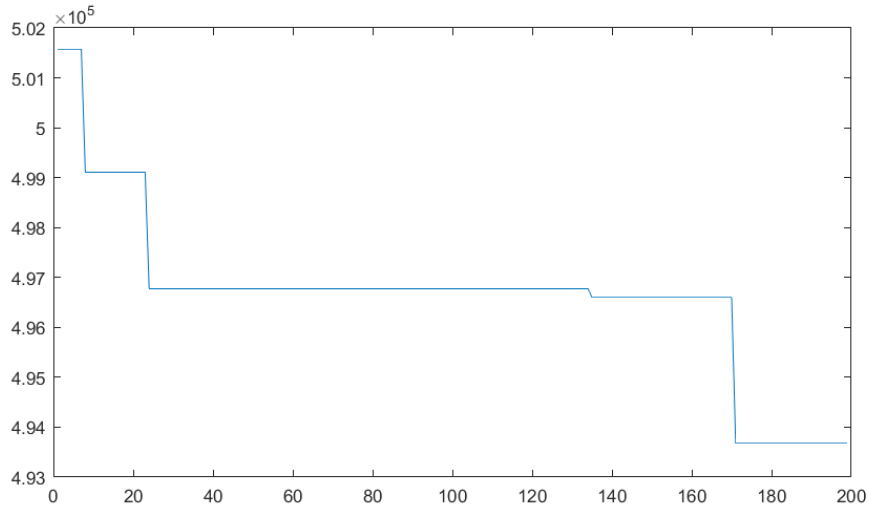
V této kapitole budeme chtít porovnat, jak rychle heuristiky konvergují k optimálnímu řešení. V první řadě si musíme určit nějaké stejné podmínky, kterých můžeme algoritmy porovnat. Nastavení výrobní linky necháme stejné z předchozí kapitoly.

Je třeba zajistit, že bude proveden stejný počet iterací v obou případech. V případě Random Search heuristiky je stop kritérium zadáno pouze počtem iterací. To se hodí, protože stačí pouze změnit jedno číslo u kritéria. U diferenční evoluce budeme muset zavést čítač iterací, který bude omezovat počet výpočtů, protože se současným omezením po sobě se nezlepšujících řešení není zajištěn počet kroků.

Počet simulací omezíme na 200, což je dostačující na nalezení zlepšení. Nejprve se podívejme na závislost hodnoty účelové funkce T na počtu iterací. V prvním grafu u Random Search algoritmu a v druhém u diferenciální evoluce.



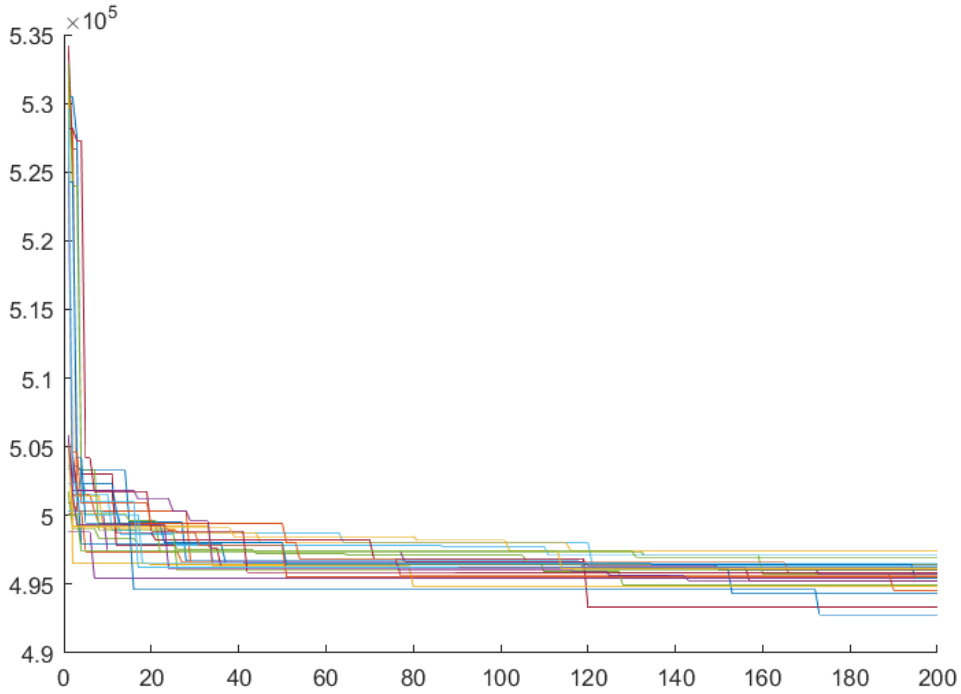
Obrázek 3.4: Závislost T na počtu iterací u Random Search



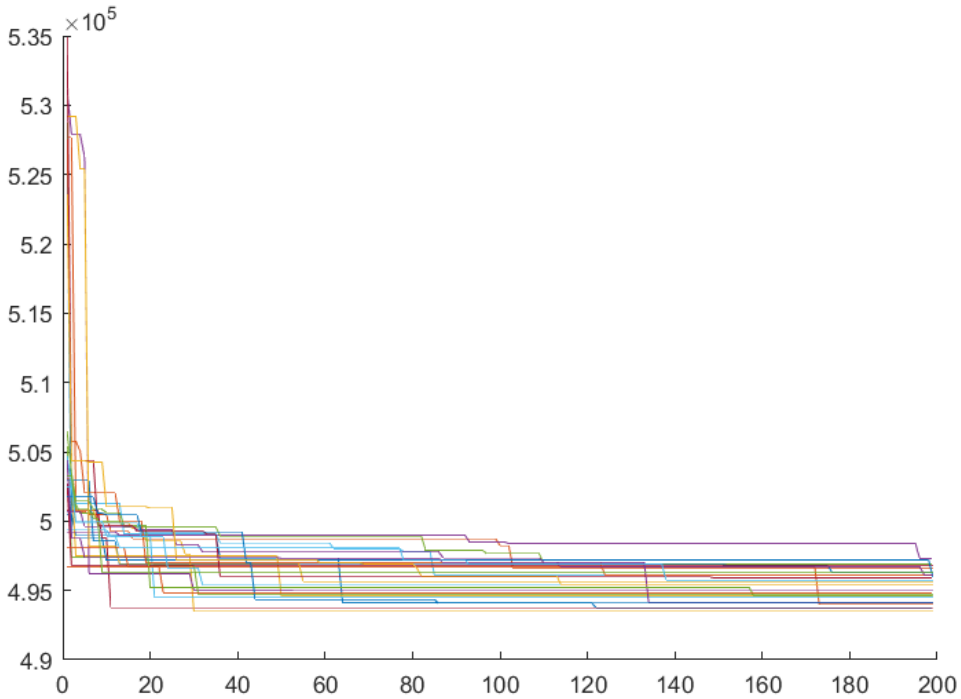
Obrázek 3.5: Závislost T na počtu iterací u diferenční evoluce

Za 200 iterací jsme se dostali s Random Search na výsledek $T = 497987$. Po dokončení 200 iterací evoluce jsme dosáhli výsledku $T = 495087$. Nejlepšího výsledku Random Search jsme přitom dosáhli po 20 iteracích.

Simulační model ovšem pracuje s náhodností a nebylo by vhodné spoléhat se pouze na jednu simulaci. Pro korektnost výsledků provedeme 30 simulací každého algoritmu a poté z těchto dat určíme, který algoritmus konverguje rychleji. Podívejme se na výsledky 30 simulací:

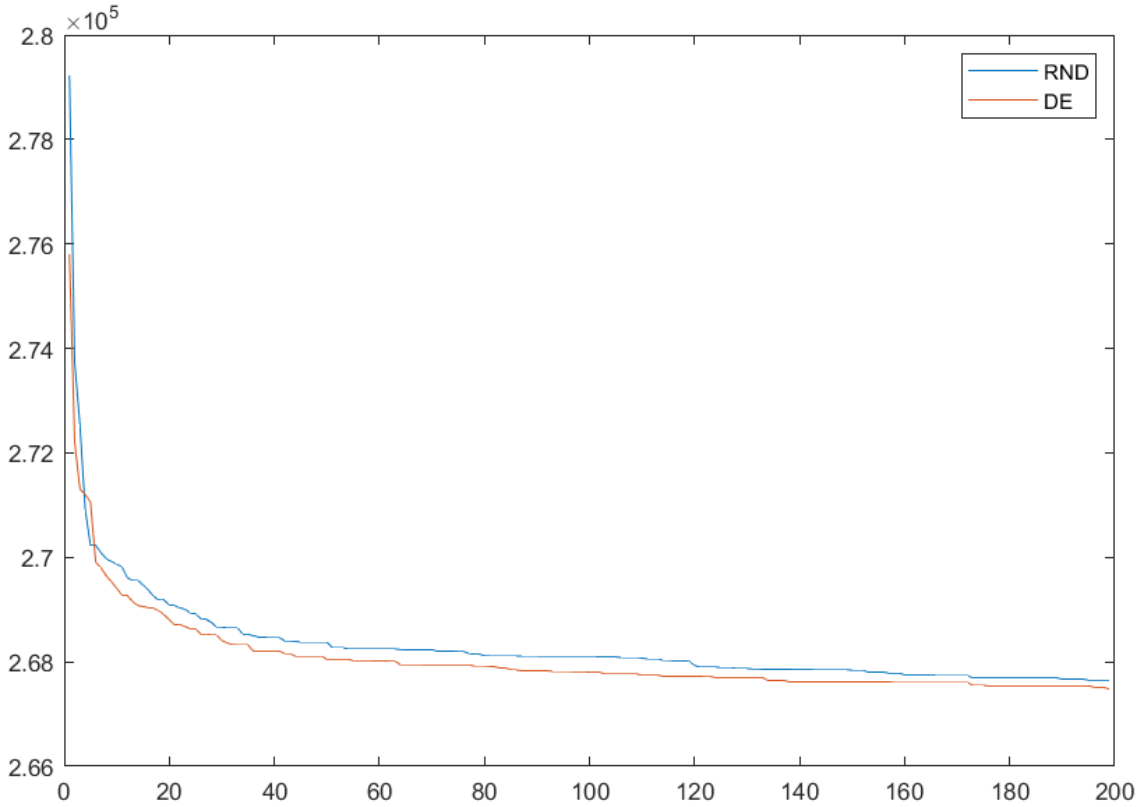


Obrázek 3.6: Závislost T na počtu iterací u Random Search



Obrázek 3.7: Závislost T na počtu iterací u diferenční evoluce

Tyto grafy sami o sobě nemají výpovědní hodnotu pro porovnání rychlosti. Proto určíme pro každý algoritmus střední hodnotu konvergence. Tu vypočteme jako aritmetický průměr všech hodnot účelové funkce v dané simulaci. Střední hodnoty pro oba heuristicky optimalizační algoritmy zobrazíme do jednoho grafu.



Obrázek 3.8: Závislost T na počtu iterací u středních hodnot heuristik

Z tohoto grafu je jasně vidět, že diferenciální evoluce konverguje k optimálnímu řešení rychleji než Random Search. Je to dáno tím, že diferenciální evoluce generuje následující generaci, tedy potenciálně lepší řešení na základě náhodné úpravy dosavadního nejlepšího řešení. Random Search pouze náhodně vybírá řešení ze stavového prostoru.

Závěr

Nyní shrneme výsledky této práce. V první řadě byly vysvětleny základní pojmy a souvislosti spojené s Industry 4.0 jako je výrobní linka, šestiosý robot, traceabilita atd.

Další část byla věnována popisu diskrétního simulačního modelu výrobní linky. Byl vysvětlen tok produktu ve výrobní lince za pomoci incidenční matice. V jednoduchosti byl popsán algoritmus simulace pomocí flowchartu. Byla vytvořena modelová linka složená z jednoduchým strojních zařízení. Ta se následně ověřila pomocí traceability. To znamená že produkty prošli výrobní linkou dle zadání

Dalším krokem bylo modelování linky obsluhované šestiosým robotem, kde všechny pohyby produktů byly vykonávány tímto automatickým ramenem. Opět byla provedena kontrola toku produktů v lince pomocí traceability. Dále byl model verifikován na základě dat z reálné výroby společnosti ALPS Electric Czech, který se zabývá automotive výrobou. Byl vytvořen model skutečné linky na kterém se porovnali výstupní modelová data s daty reálnými. Tímto byl náš diskrétní simulační model ověřen.

Do modelu s šestiosým robotem byl doplněn systém logiky obsluhy jednotlivých strojních zařízení. Pro různé systémy logiky jsme dostali různé výstupy linky. Cílem je linku co nejvíce zefektivnit. To znamená, že je nutné najít logiku pro modelovou výrobní linku, takovou aby výstup linky byl co nejvyšší.

Řešením této úlohy je implementace optimalizačního algoritmu. Protože je množina řešení nespojitá a nelineární, byly využity metod heuristické optimalizace. Při správné implementaci lze získat dobré optimální řešení. První algoritmus byl takzvaný random search, který náhodně prohledával stavový prostor. V další části byl implementován algoritmus diferenciální evoluce, který zajistil rychlejší konvergenci.

Díky implementaci systému logiky a následné heuristické optimalizaci se podařilo zvýšit výstup linky o 6,4%. To se na první pohled nemusí zdát jako velká úspora. Musíme vzít v potaz, že se jedná o zlepšení s nulovou investicí a velice jednoduchou implementací na mnoha moderních výrobních linkách.

Tato práce demonstruje praktické využití metod Industry 4.0 na reálném výrobním provozu. Lze očekávat, že v budoucnu se bez těchto a dalších metod spojených s pojmem Industry 4.0 neobejde žádný výrobní podnik, který bude chtít být konkurenceschopný.

Literatura

[1] Fourth Industrial Revolution. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2021-02-09]. Dostupné z: https://en.wikipedia.org/wiki/Fourth$_I ndustrial_R evolution$

[2] Wiley series in probability and statistics. ISBN: 9780471791270. RAO, Singiresu S. Engineering optimization: Theory and practice. Čtvrte vydání.

[3] PASCUAL, D. Galar, DEPONTE, Pasquale a KUMAR, Uday. Handbook of Industry 4.0 and SMART Systems. CRC Press, 2019. ISBN: 9781138316294.

[4] EUROPEAN PARLIAMENT, 2015. Industry 4.0 Digitalisation for productivity and growth [online]. 2015. Dostupné z: http://www.europarl.europa.eu/RegData/etudes/BRIE/2015/568337

[5] Umachandran, Krishnan Jurčić, Igor Corte, Valentina Ferdinand-James, Debra. (2018). Industry 4.0.: The New Industrial Revolution.

[6] GROSS, Donald, SHORTLE, John F., THOMPSON, James M. a HARRIS, Carl M. Fundamentals of queueing theory. 4th ed. Hoboken: John Wiley Sons, 2008.

[7] YAMAHA. Six-Axis robot. In: Https://www.yrginc.com/products/details/?product=6-axis [online]. [cit. 2021-02-25].

[8] Programovatelný logický automat. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2021-02-25]. Dostupné z: https://cs.wikipedia.org/wiki/ProgramovatelnC3BD$_l ogickC3BD_a utomat$

[9] Heuristika. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2021-03-28]. Dostupné z: https://cs.wikipedia.org/wiki/Heuristika

[10] YANG, Xin-She. Nature-inspired Optimization algorithms. Londýn: Elsevier, 2014. ISBN 978-0-12-416743-8.

[11] A Comparative Study of Differential Evolution Variants in Constrained Structural Optimization. Frontiers.org [online]. [cit. 2021-5-6]. Dostupné z: https://www.frontiersin.org/articles/10.3389/fbuil.2020.00102/full

[12] Differential Evolution with Population and Strategy Parameter Adaptation [online]. In: GONUGUNTLA, V., R. MALLIPEDDI a Kalyana VELUVOLU. [cit. 2021-5-6]. Dostupné z: https://www.hindawi.com/journals/mpe/2015/287607/

Přílohy

Příloha obsahující Skripty se simulací

Pro různé nastavení simulace jsou přiloze různé verze skriptů. Všechny mají stejnou srukturu, kdy inicializační skript volá ostatní.

- Main - Slouží pro spuštění programu. Nastavují se v něm parametry zařízení a incidenční matice. Dále zde probíhá optimalizační nadstavba. Main tedy volá všechny ostatní funkce z tohoto seznamu.
- Core - Tato funkce je volaná z Main. Probíhá zde samotná simulace s nastavením z nadřezeného souboru.
- GenerateGoods - Slouží k určení počtu produktů, které mají výrobní linkou projít.
- Draw - Vykresluje grafy po dokončení simulace.
- Station - Třída, jejíž instance jsou všechny zařízení v lince. Obsahuje metody na vstup a výstup produktu do zařízení.

Nyní uvedeme číslování souborů Main pro jednotlivé funkce, které lze spustit z přílohy:

- Main22 - Simulace výrobní linky bez šestiosého robota. Výrobní dávka je 600 kusů.
- Main40 - Simulace výrobní linky s šestiosým robotem bez logiky.
- Main52 - Optimalizační algoritmus Random Search. Pro ilustrační účely probíhá pouze 20 simulací pro dávku 50 kusů.
- Main60 - Optimalizační algoritmus diferenciální evoluce. Výrobní dávka je 50 kusů. Algoritmus se zastaví po 20 iteracích bez zlepšení hodnoty účelové funkce.