VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

ALGORITMY PRO OPERATIVNÍ PLÁNOVÁNÍ VÝROBY

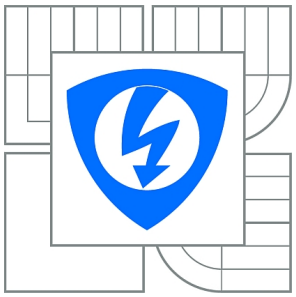
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE PAVOL KRŠÁK
AUTHOR

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF CONTROL AND INSTRUMENTATION

ALGORITMY PRO OPERATIVNÍ PLÁNOVÁNÍ VÝROBY

ALGORITHMS FOR ADVANCE PRODUCTION PLANNING AND SCHEDULING

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

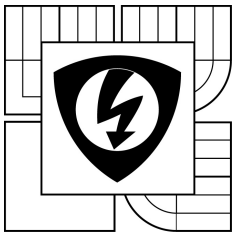
AUTOR PRÁCE — PAVOL KRŠÁK

AUTHOR

VEDOUCÍ PRÁCE — Ing. JAN PÁSEK, CSc.

SUPERVISOR

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav automatizace a měřicí techniky

Bakalářská práce

bakalářský studijní obor
Automatizační a měřicí technika

Student: Pavol Kršák ***ID:*** 146875
Ročník: 3 ***Akademický rok:*** 2013/14

NÁZEV TÉMATU:

Algoritmy pro operativní plánování výroby

POKYNY PRO VYPRACOVÁNÍ:

a. Prezentovat problematiku operativního plánování výroby pomocí systémů MES
b. Navrhnout vhodné algoritmy pro vytvoření plánu výroby podle zadaných kritérií
c. Realizovat algoritmy v jazyce C# jako funkce použitelné v systému COMES pro opakované využití při implementaci modulu COMES Modeller.

DOPORUČENÁ LITERATURA:

1. COMPAS automatizace - průmyslová automatizace a výrobní informační systémy MES [online], [2013
2. COMPAS. Podniková dokumentácia. 2013. verze 3.
3. BUBENÍK, P. Systémy pre plánovanie, riadenie a optimalizáciu výroby. In AT&P Journal, 2004, č. 6, s.18-20. ISSN 1335-2237

Termín zadání: 10.2.2014 ***Termín odevzdání:*** 26.5.2014

Vedoucí práce: Ing. Jan Pásek, CSc.
Konzultanti bakalářské práce: Ing. Brázda Roman

doc. Ing. Václav Jirsík, CSc.
předseda oborové rady

ABSTRAKT

Bakalárska práca sa zaoberá riešením problematiky operatívneho plánovania výroby. Účelom práce je prezentovať problematiku operatívneho plánovania výroby a následný návrh a realizácia plánovacích algoritmov v jazyku C# na konkrétny plánovací problém. Pri prezentácii problematiky je uvedená spoločnosť "'COMPAS automatizace s.r.o."', ktorá umožnila realizovanie projektu. V práci sú rozobrané plánovacie metódy a spôsoby ich realizácie. Konkrétnejšie sú rozpracované evolučné algoritmy, expertné systémy v plánovaní a deterministické plánovanie cez lineárne programovanie. V tretej kapitole je charakterizovaný plánovací problém výroby kávovej zmesi, ktorý mi bol poskytnutý spoločnosťou COMPAS. Následne je rozpracovaný samotný návrh riešenia založený na evolučnom princípe. Riešenie pozostáva z modelu technológie na výrobu kávovej zmesi a ActionListu. ActionList predstavuje súbor pravidiel, ktorými sa model riadi. Potom sú zadefinované pravidlá podrobené evolučným procesom so snahou nájsť lepšie existujúce riešenia. Realizované funkcie použiteľné v systéme COMES pre opakované využitie pri implementácii modulu COMES Modeller sú rozpracované spolu s podrobným popisom v štvrtej kapitole.

KĽÚČOVÉ SLOVÁ

operatívne plánovanie výroby, evolučné algoritmy, plánovanie, optimalizácia, plánovacie algoritmy

ABSTRACT

Bachelor's thesis analyzes solution of operational problems of production planning. Purpose of the thesis is presentation of operational problems of production planning and following proposal and implementation of planning algorithms in language C# on specific planning problem. By the presentation of the problematic is listed company „COMPAS automatizace s.r.o.", which enabled implementation of the project. The thesis analyzes planning methods and process of their implementation. More specifically the thesis analyzes evolutionary algorithms, expert systems in planning and deterministic planning via linear programming. In third chapter is described in more details planning program of production of coffee blends, which was provided by COMPAS company. Like a next, the thesis describes specifically proposal of solving based on evolutionary principle. The solving consists of the model of technology designed to production of coffee blends and ActionList. ActionList represents a set of rules that are governed by model. Then rules are defined undergone an evolutionary process in an effort to find better solution the existing. Applied features implemented in the system COMES to repurpose by the implementation of module COMES Modeller are developed together with detailed description in the fourth chapter.

KEYWORDS

advance production planning, evolution algorithms, scheduling, optimalization, scheduling algorithms

KRŠÁK, Pavol *Algoritmy pro operativní plánování výroby*: bakalárska práca. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2014. 77 s. Vedúci práce bol Ing. Ján Pásek, CSc.

PREHLÁSENIE

Prehlasujem, že som svoju bakalársku prácu na tému „Algoritmy pro operativní plánování výroby“ vypracoval samostatne pod vedením vedúceho bakalárskej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

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

Brno ....

(podpis autora)

POĎAKOVANIE

Dovoľujem si touto cestou poďakovať vedúcemu bakalárskej práce Ing. Janovi Pásekovi, CSc., ako aj konzultantom Ing. Alešovi Stehnovi a Ing. Romanovi Brázdovi za odborné vedenie, cenné rady a pomoc pri vypracovaní tejto práce.

Brno ....

(podpis autora)

OBSAH

Úvod **11**

1 Cieľ práce **13**

2 Prezentácia problematiky operatívneho plánovania výroby pomocou systému MES **14**

2.1 Spoločnosť COMPAS automatizace s.r.o. - průmyslová automatizace a výrobní informační systémy MES 14

2.2 Moduly a komponenty systému COMES 14

2.2.1 COMES CCI 15

2.2.2 COMES Logon 15

2.2.3 COMES Historian 15

2.2.4 COMES Modeller 16

2.2.5 COMES Tracebility 16

2.2.6 COMES Batch 16

2.2.7 COMES Komponent 17

2.3 Plánovacie systémy 18

2.3.1 História plánovacích systémov 18

2.3.2 Evolučné algoritmy 19

2.3.3 Expertné systémy 23

2.3.4 Deterministické plánovanie 26

3 Výber plánovacieho algoritmu pre optimalizáciu v PKZ **29**

3.1 Popis zadaných kritérií pri tvorbe plánu výroby pražených kávových zmesí 29

3.2 Návrh plánovacieho algoritmu 30

3.3 Vytváranie modelu technológie a špecifikácie operácii 31

3.4 Popis fungovania plánovania na modele technológie 32

3.5 Testovanie dostupnosti výrobnej cesty 33

3.6 Zápis výrobnej cesty do rozvrhov modelu 35

3.7 Použitý evolučný algoritmus 36

3.8 Testovanie aplikácie plánovania 38

3.9 Popis aplikácie 41

3.10 Možné vylepšenia 43

4 Realizácia algoritmov v jazyku C# ako funkcie použiteľnej v systéme COMES pre opakované využitie pri implementácii modulu COMES Modeller **44**

5 Záver **48**

Literatúra **50**

Zoznam symbolov, veličín a skratiek **52**

Zoznam príloh **53**

A Tabuľky meraných hodnôt **54**

B Plánovacie kraft data **56**

C Zdrojové kódy **57**

ZOZNAM OBRÁZKOV

2.1 Architektúra systému COMES medzi vrstvami ERP a MCS 15
2.2 Hierarchia pojmov používaných v systému COMES Batch a jej členenie 17
2.3 Základné delenie plánovania výroby. 18
2.4 Všeobecná štruktúra evolučného algoritmu [5] 21
2.5 Bloková schéma plánovacieho expertného systému [9]. 24
3.1 Príklad možného usporiadania technológie 29
3.2 Vývojový diagram fungovania modelu technológie 32
3.3 Vývojový diagram testovacej fázy modelu. TD-testovanie dostupnosti, testovanie PnD-pripojených na dostupné 34
3.4 Vývojový diagram zápisu rozvrhu výroby vybranej kávy 36
3.5 Vývojový diagram evolučného algoritmu 37
3.6 Príklad symetrického, nezdieľajúceho zapojenia technológie 38
3.7 Výrez z aplikácie. Závislosť ohodnotenia najlepšieho jedinca v populácii od evolučného cyklu. x-ová os číslo evolučného cyklu, y-ová ohodnotenie najlepšieho jedinca v populácii. Použitie Roulette Wheel Selection. 40
3.8 Hlavné okno plánovacej aplikácie 41

ZOZNAM TABULIEK

A.1 Merané hodnoty na modeli s evolučnou trasou 54
A.2 Merané hodnoty na modeli s deterministickou trasou 55
B.1 Plánovacie kraft data 56

ÚVOD

Výrobný proces predstavuje zložitý, tvorivý systém, ktorého funkciou je tvorba úžitkových hodnôt. Závisí od povahy výroby, výrobného programu, použitých technológií, zložitosti a skladby výrobkov, spôsobu a miery opakovateľnosti výroby, využiteľnosti strojov a strojných zariadení a pod. Nástrojom pre zefektívnenie výrobného procesu, s dôrazom na maximalizovanie zisku firmy, je zavádzanie automatizovaného plánovania výroby. Myšlienka automatizovať a optimalizovať je stará ako ľudstvo samo. V dnešných podnikoch to platí dvojnásobne, najmä od roku 2009, keď novodobá ekonomická situácia donútila hľadať rezervy v podnikovom systéme na miestach, ktoré sa doteraz prehliadali. Jednou z možností ako zoptimalizovať systém je počítačové plánovanie výroby, ktoré v súčasnej dobe zaznamenáva neustály rozvoj. Jeho hlavnou výhodou je, že pomáha uľahčovať prácu plánovačom, ktorí už nie sú odkázaní na tabuľkové prepočty. Tým sa eliminuje riziko vyskytnutia možných chýb v plánovacom systéme zapríčinených ľudským faktorom, ktoré by mohli spôsobiť finančnú stratu podniku. V svojej práci sa zaoberám problematikou plánovania výroby.

Pri prezentácii plánovacej problematiky charakterizujem okrem iného aj spoločnosť COMPAS a integrovaný podnikový systém MES, umožňujúci úplne elektronické organizovanie výroby. Tento systém je zastúpený systémom COMES, ktorý v tomto bode podrobnejšie opisujem, ako aj jeho moduly a komponenty.

Následne sa zaoberám problematikou plánovacích systémov, ich vývojom od prvotných, počiatočných metód až po v súčasnosti využívané sofistikované modely. Bližšie tu uvádzam dôvody neustáleho zavádzania nových modelov používaných v procese plánovania výroby.

Podrobnejšie popisujem používané spôsoby plánovania výroby v podnikoch. Tieto spôsoby využívajú evolučné algoritmy, expertné systémy a deterministické plánovanie. Evolučné algoritmy sú bližšie rozobrané bode 2.3.2, ako aj ich aplikovateľnosť a využitie pre efektívne riešenie problémov. Detailne tu rozpracuvávam všeobecnú štruktúru evolučných algoritmov. Expertné systémy obšírnejšie charakterizujem v časti 2.3.3, ktorá je zameraná najmä na ich základnú architektúru. Lineárne programovanie ako súčasť deterministického plánovania načrtávam v bode 2.3.4, tu je tiež opísaná štandardná klasifikácia tohto typu plánovania.

Od kapitoly 3 sa venujem popisu a riešeniu problému, ktorý mi bol poskytnutý spoločnosťou COMPAS. Ide o problematiku návrhu výroby v prevádzke na výrobu kávových zmesí s danými kritériami. Medzi kritéria patrí:

- nutnosť dodržiavania výrobnej následnosti pri absolvovaní piatich výrobných operácii (praženie, zrenie, zomletie, odplynenie a plnenie).
- možnosť plánovania na variabilnom počte jednotlivých strojných zariadení/síl

v jednotlivých operáciách.

- možnosť definovania ne/existencie spojení medzi jednotlivými strojnými zariadeniami a silami.
- a iné.

V bakalárskej práci navrhujem algoritmy riešenia problematiky evolučnými algoritmami. Toto riešenie pozostáva z modelu technológie na výrobu kávových zmesi a ActionListu. ActionList predstavuje súbor pravidiel (evolučné pravidlá 3.1), podľa ktorých sa model správa. Tieto pravidlá sú následne podrobené evolučným procesom ako sú ohodnotenie, kríženie, mutácia, selekcia so snahou nájsť lepšie riešenie.

V ďalšej časti práce sa venujem testovaniu navrhnutej aplikácie ako aj návrhmi na jej vylepšenie a ďalšie rozpracovanie.

V kapitole 3.9 je z dôvodu overenia funkčnosti algoritmov a pre zobrazenie výsledkov vypracované WPF uživateľské rozhranie s možnosťou nastavenia parametrov ako v modely technológie výroby, tak aj v evolučnej populácii jedincov.

V poslednej kapitole 4 je poskytnutý náhľad na realizované funkcie použiteľné v systéme COMES pre opakované využitie pri implementácii modulu COMES Modeller s podrobným popisom. Pomocou týchto funkcií je možné zaručiť plné ovládanie navrhovaných algoritmov a zobrazenie výsledkov.

1 CIEĽ PRÁCE

Hlavným cieľom bakalárskej práce je navrhnutie, testovanie a realizácia algoritmov pre optimalizovanie výrobného plánu v podniku pre výrobu kávových zmesí. V teoretickej časti bola práca orientovaná na prezentáciu problematiky operatívneho plánovania výroby pomocou systému MES od spoločnosti COMPAS automatizace, spol. s r.o. a na plánovacie systémy obecne. Ďalšia časť práce bola venovaná výberu vhodného plánovacieho algoritmu pre vytvorenie plánu výroby kávových zmesí podľa zadaných kritérií. Skladala sa z čiastkových cieľov, ktoré spočívali v identifikácii plánovacieho problému v prevádzke, výbere vhodnej plánovacej metódy pre optimalizáciu výroby v zmysle minimalizovania dĺžky časových prestojov a maximalizácie vyťaženia poskytnutej technológie. Tretia časť práce bola zameraná na realizáciu algoritmov v jazyku C# ako funkcie použiteľnej v systéme COMES pre opakované využitie pri implementácii modulu COMES Modeller.

2 PREZENTÁCIA PROBLEMATIKY OPERATÍVNEHO PLÁNOVANIA VÝROBY POMOCOU SYSTÉMU MES

2.1 Spoločnosť COMPAS automatizace s.r.o. - průmyslová automatizace a výrobní informační systémy MES

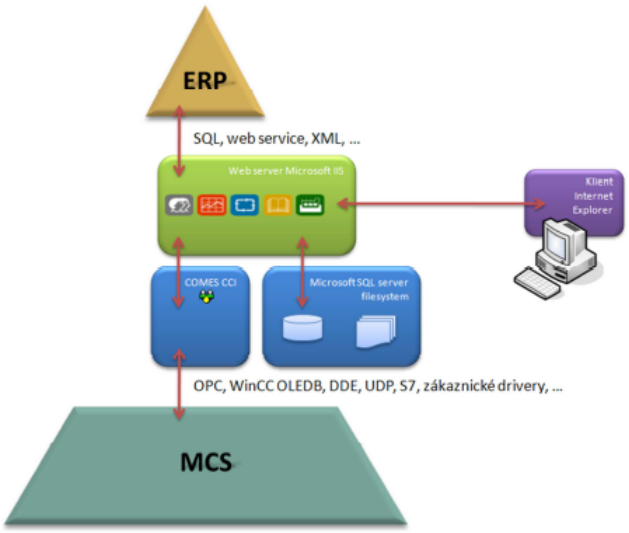
Česká spoločnosť COMPAS so sídlom v Žďáru nad Sázavou ponúka inteligentné riešenia priemyselnej automatizácie a MES (Manufacturing Execution System) systémov. Spoločnosť dlhodobo spolupracuje s firmou SIEMENS a s tým spojenou ponukou kompletného spektra produktov SIEMENS TIA ako aj programovým vybavením pre riadiace systémy Simatic S7 a PCS7, ktoré majú HMI zobrazenie na platforme vizualizácie WinCC. Spoločnosť COMPAS úspešne realizovala stovky projektov, získala množstvo certifikátov. Spoločnosť sa zameriava hlavne na pružné receptúrové automatizácie šaržových výrob potravín a liečiv, Batch systémy, a ďalej automatizáciu sériových výrob predovšetkým v odvetví strojárenstva a výroby automobilov.

Princípom je integrácia podnikových systémov MES (zastúpených systémom COMES), umožňujúca úplné elektronické organizovanie výroby so spojenými výhodami ako je prehľadnosť, zníženie nákladov vyplývajúcich z optimalizácie riadenia výroby, pri vysokej rovnomernosti kvality (COMES funkčnosťou podporuje štíhlu výrobu - Lean Manufacturing a jeho systém charakterizuje štíhlosť - Lean MES). V oblasti plánovania sa osvedčili aplikované funkcie, kapacitné plánovanie výroby, či krátkodobé plánovanie výroby ASP (Active Server Pages).

Produkt COMES v podnikovej pyramíde výrobných informačných systémov riadenia predstavuje medzivrstvu medzi ERP a MCS (Manufacturing Control System). Účelom je poskytovať čo možno najviac informácií pre okamžité riadenie a optimalizáciu vo výrobe [1]. Pri opisovaní produktu COMES som vychádzal z podnikovej dokumentácie spoločnosti COMPAS [2].

2.2 Moduly a komponenty systému COMES

Každý z modulov systému COMES, ktoré zaisťujú fundamentálnu funkčnosť obsahuje doplnkové komponenty s prídavnými možnosťami, samozrejmosťou je vzájomná kompatibilita.



Obr. 2.1: Architektúra systému COMES medzi vrstvami ERP a MCS

2.2.1 COMES CCI

Súčasťou systému COMES je modul COMES CCI (COMES Communication interface), klient-server aplikácia, zaisťujúca komunikáciu s hierarchicky nižšie položenou úrovňou MCS. Každému CCI serveru sa priraďuje tzv. modul, a ten určuje typ komunikácie (napr.OPC). Klientska časť COMES CCI je inštalovaná do jednotlivých modulov COMES systému.

2.2.2 COMES Logon

Ako centrálne rozhranie, ktoré umožňuje prístup k dátam z ostatných modulov a pre správu užívateľov a užívateľských skupín je určený COMES Logon. Taktiež zabezpečuje prihlasovanie, správu číselníkov a záloh dát v databázach spoločnosti.

2.2.3 COMES Historian

Pre zber, ukladanie, analýzy každého druhu, archiváciu dát a pre ich rôznu interpretáciu prípadne manipuláciu vo forme výpočtov je vyvinutý COMES Historian. Časy všetkých udalostí sú ukladané do databázy v UTC (Universal Time Coordinated) kvôli univerzálnosti pre prípadné nahliadanie do histórie z viacerých časových pásiem.

2.2.4 COMES Modeller

Modul COMES Modeller je súbor rôznych editorov a možností ako transformovať veľké množstvo dát z procesu (od užívateľov, či z databáz) na užitočné informácie. Je možné vytvoriť náhľady na model technologického procesu, tabuľky pre vstup a zobrazenie informácii.

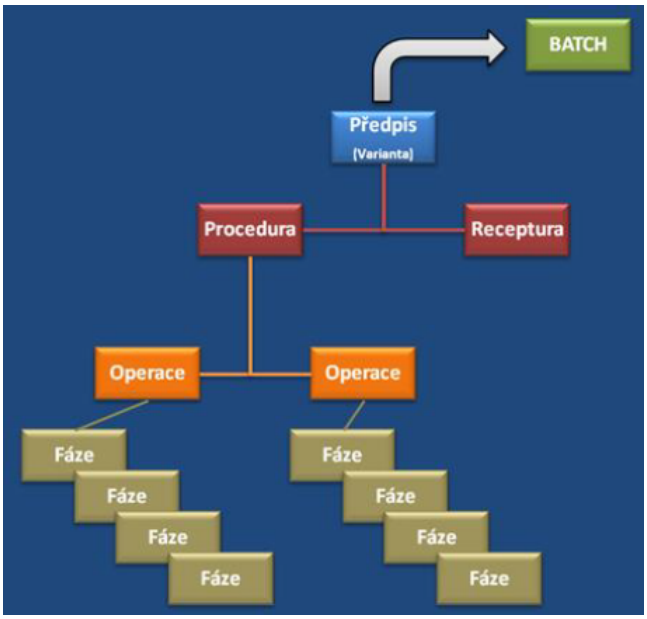
2.2.5 COMES Tracebility

Aby bolo možné vypátrať pôvod chyby vo výrobnom procese, COMES Tracebility si ukladá informácie o všetkých surovinách a ich medziproduktoch (identifikuje šarže/série), ktoré boli do procesu zahrnuté. Pomocou takto uložených väzieb je možné spätne prechádzať výrobnú históriu a nájsť výrobný krok, či operáciu, kde chyba nastala. Po identifikácii chyby sú všetky výrobky a medziprodukty vzniknuté po tomto kroku, či operácii zaznamenané ako potenciálne chybné a je potrebné preveriť ich kvalitatívne parametre.

2.2.6 COMES Batch

Modul je určený pre nespojité riadenie výrobných procesov, umožňuje prenášať programovanie výrobných krokov a parametrov z riadiaceho systému (DSC či PLC) do COMES Batch. Cez správu a vytváranie predpisov je užívateľ - technológ schopný spravovať výrobné postupy cez graficky alebo tradičný editor kódu bez zásahu do riadiaceho systému. Modul generuje elektronický záznam výroby (EBR) a zdieľa ich s ostatnými COMES komponentmi.

- BATCH - je proces s definovaným vstupom (množstvo surovín) a definovaným výstupom (produkt). Počas tohto procesu prechádza surovina sústavou činnosti na rôznych zariadeniach po definovaný čas. Batch alebo tiež Dávka, je objem produktu vytvoreného za Batch proces. Batch je zadefinovaný v predpise a nový Batch sa vytvára v čase zaplánovania predpisu do procesu výroby.
- Predpis - určuje výrobné požiadavky pre konkrétny produkt. Norma definuje predpis obecný, miestny, hlavný, a vykonávací (zo vzorového predpisu pre výrobu jednej konkrétnej dávky). S poslednými dvoma pracuje COMES Batch.
- Procedúra - súbor operácií prepojených logickými operátormi, pri vykonávaní dohliada na zachovanie postupností operácii a alokáciu zariadení.
- Receptúra - súbor informácií o predpise (vstupy, výstupy a iné parametre).
- Operácia - je menšie zoskupenie fáz (aspoň jednej), opäť prepojených logickými operátormi.



Obr. 2.2: Hierarchia pojmov používaných v systému COMES Batch a jej členenie

- Fáza - tiež fyzická fáza - je elementárny prvok procedúry, ktorý definuje istý stav v procese, nie je možné ich meniť bez zásahu do programovej časti PLC. Naopak užívateľská fáza je editovateľná v COMES Batch.

2.2.7 COMES Komponent

- COMES CLIENT - preberá funkcie prehliadača Internet Explorer a zamedzuje prístup do operačného systému z prostredia MSIE. Tým zvyšuje bezpečnosť OS.
- COMES WATCHDOG - sleduje správnosť vykonávania programu v cykloch.
- COMES LOGON WinCC autorizačný klient - rozšírenie umožňujúce prihlásiť užívateľa prihláseného do SCADA systému WinCC do komponentu COMES CLIENT
- COMES LOGON klient - rozšírenie zabezpečujúce autorizáciu do COMES systému prostredníctvom Microsoft domény.

2.3 Plánovacie systémy

Dnes už nikto nepochybuje o potrebe plánovania a diskusia sa vyvíja skôr smerom, ako zlepšiť tento proces. V súčasnosti každý podnik rieši situácie súvisiace s kolísavým dopytom, kratším životným cyklom výrobkov a globálnou konkurenciou. Uvedené problémy nútia výrobcov prispôsobiť sa a pružne reagovať na požiadavky okolia. Pre väčšiu efektivitu riešenia dennej problematiky, firmy prijímajú progresívne techniky plánovania a rozvrhnutia výroby, ktoré generujú optimalizačné realizačné plány, zamerané najmä na :

- zabezpečenie vysokej výťažnosti výrobného procesu, predstavuje odhad termínov a ceny výroby,
- flexibilitu výrobného sortimentu v závislosti od stanovených kritérií zákazníka,
- schopnosť inovácií u jednotlivých výrobných programoch,
- rýchle riešenie neočakávaných problémov,
- znižovanie množstva zásob materiálu,
- zvyšovanie kvality výrobkov, časového a výkonového využitia strojov.

Aplikácie určenú k plánovaniu výroby sú orientované k príprave optimalizovaných plánov v diskrétnom, prebiehajúcom i opakovanom výrobnom systéme. Za pomoci využitia moderných informačných technológii ako aj optimalizačných algoritmov, prebieha disponibility/použiteľnosť v skutočnom čase. Plánovanie výroby možno rozdeliť do troch častí, ktoré sú vyobrazené na obrázku 2.3.

Obr. 2.3: Základné delenie plánovania výroby.

2.3.1 História plánovacích systémov

Prvá metóda, ktorá sa uplatnila v priemysle, bola MRP metóda (Material Requirements Planning). MRP slúži na plánovanie výroby, riadenie zásob, obsahuje zapracovanie štruktúrovaných kusovníkov a základné plánovanie nákupu. Ako prvá metóda

zavádza počítačové spracovávanie dát, čo urýchlilo ich transformáciu na informácie. Hlavným nedostatkom je prílišné zjednodušenie modelu, ktorý nezohľadňuje reálne obmedzenia podniku [3].
Odpoveďou boli metódy, ktoré tieto obmedzenia už brali do úvahy.

- MRPII - rozširuje MRP, pričom započítava kapacitné a materiálové obmedzenia. Obsahuje MPS (Master Production Schedule alebo Plán finálnej výroby), do ktorého vstup predstavuje predpokladaný dopyt, náklady na výrobu a pod. a výstup predstavuje odhad termínov a ceny výroby.
- CRP - kapacitné plánovanie, ktoré rozhoduje o uskutočnení plánu spoločnosti na základe dostupnej produkčnej kapacity [3, 4].

V deväťdesiatych rokoch prebehlo vetvenie plánovacích systémov do špecializovaných oblastí jednotne nazývaných ERP napr. DPR-plánovanie distribúcie [4].
Príchodom nového milénia sa výpočtový výkon IT techniky dostal na takú úroveň, ktorá dovoľuje pracovať so sofistikovanejšími modelmi. Príkladom môže byť:

- SCM - prepája jednotlivé články dodávateľského reťazca zdieľanými informáciami, a tým sflexibilňuje možnosti podniku. Cieľom je minimalizovať skladové zásoby a cenu vnútri reťazca.
- APS - úlohou je nielen určenie realizovateľnosti, ale aj zoptimalizovanie procesu napr. minimalizovanie prestojov pri prenastavovaní komponentov výroby

[3, 4].

2.3.2 Evolučné algoritmy

Evolučné algoritmy sú algoritmy, ako už názov napovedá, hľadajúce najlepšie riešenie skoro ľubovoľného problému. Podľa typu riešených úloh je prehľadávanie vo všeobecnosti nenáročné, vyžadované sú iba dve podmienky:

- rozložiteľnosť
- vyhodnotiteľnosť [5]

Metodika spočíva v hľadaní extrémov tzv. plochy vhodnosti, ktorej každý bod predstavuje jedno ohodnotené riešenie. Táto plocha má $(n+1)$ rozmerov pričom n predstavuje počet parametrov popisujúci problém a $(n+1)$-vá súradnicová os reprezentuje výsledok vyhodnocovacej funkcie s n parametrami. Parametre môžu byť rôznych typov, kvantitatívne ale aj kvalitatívne, pre praktickosť a lepšiu spracovateľnosť sa však kvalitatívne parametre prevádzajú na číselné hodnoty [5].

Postup hľadania sa začína určením a testovaním množiny bodov (kandidátov) na ploche vhodnosti. Následne sa určia noví kandidáti. Algoritmus sa opakuje pokiaľ nenasleduje ukončovacia podmienka, ktorou môže byť obmedzený počet cyklov, nenájdenie lepších kandidátov alebo dosiahnutie žiadanej hodnoty vyhodnocovacej funkcie [6].

Hlavným rozdielom evolučných algoritmov je mohutnosť množiny testovaných kandidátov (tiež jedincov), s ktorou algoritmus pracuje a metóda vytvárania nových kandidátov. Podľa mohutnosti populácie sa algoritmy rozdeľujú na individuálne (v každom cykle sa vytvára iba jeden jedinec) a populačné algoritmy. Populačné algoritmy môžu prehľadávať paralelne rôzne miesta na ploche vhodnosti alebo jednu oblasť s viacerými jedincami. Podobne vytváranie nových kandidátov sa delí na nezávislé prehľadávanie (bez pamäti) a na závislé (s pamäťou). Pri závislom prehľadávaní sa vychádza z predpisu z predchádzajúcich "generácii"kandidátov. Snahou je, v čo najväčšej miere, využiť informácie, ktoré sme obdržali na prehľadávanie oblastí s pre nás zaujímavou charakteristikou. Podrobnejšie sa budeme zaoberať populačnými algoritmami s pamäťou [5].

Výber nových jedincov je rovnováhou medzi náhodnosťou a cieleným usmerňovaním. Ak by sme prehľadávali priestor príliš definitívne, populácia jedincov by strácala diverzibilitu, a tým pádom schopnosť nájsť globálny extrém. Naopak ak sa spoliehame iba na náhodu, strácame možnosť zamerať sa na oblasti s lepšími výsledkami. Mechanizmus je založený na Darwinovom procese evolučného výberu a Mendelejevovom procese génovej dedičnosti. Cieľom je využiť prírodné mechanizmy pre efektívne riešenie problémov [7].

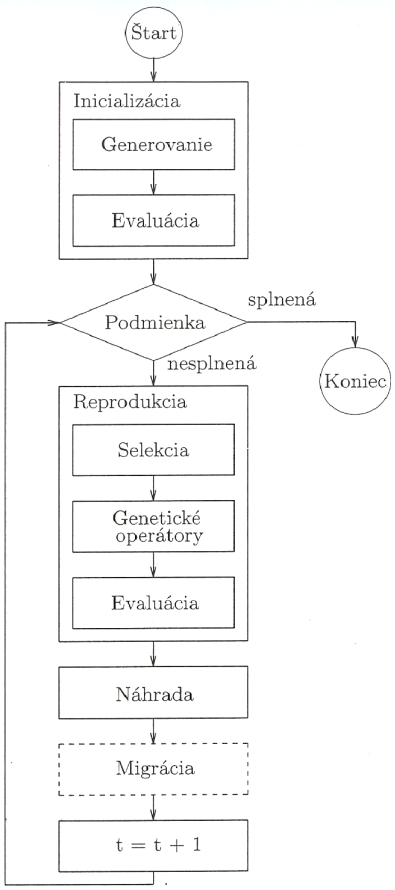
Ako to už býva, neexistuje univerzálny genetický algoritmus pre riešenie každého problému s vysokou výkonnosťou. V praxi máme metódy slabé so širokým spektrom použiteľnosti a silné tzv. špecializované. Obecne slabé metódy volíme, ak neexistuje na daný problém silná metóda. Použitie silnej metódy je priveľmi náročné, a výsledky silnej metódy sú približne rovnaké ako pri slabej metóde [7].

Všeobecná štruktúra. Evolučný algoritmus ako prehľadávací nástroj pracuje so súborom riešení problému (populáciou). Každé riešenie sa označuje ako jedinec s charakteristickými parametrami (vlastnosťami), ktoré nejakým spôsobom riešia problém. Tieto parametre lokalizujú daného jedinca na ploche vhodnosti [7].

Prvý krok algoritmu predstavuje vytvorenie prvotnej populácie v bloku *Generovanie*. Tá je následne podrobená evolučnému cyklu, a každý prechod týmto cyklom predstavuje jednu generáciu (prvotná generácia sa označuje 0-tá generácia). Ak prvotnú populáciu označíme P(t=0), i-tého jedinca $a_i(t=0)$ v prvotnej generácií a počet jedincov v populácii bude $\mu(t=0)$, potom

$$P(0) = \{a_1(0), a_2(0), ..., a_\mu(0)\} \tag{2.1}$$

Títo jedinci predstavujú μ riešení daného problému. Generovať ich je možné na základe náhody, rovnomerne rozložiť po ploche vhodnosti (hodnotiacej funkcii).



Obr. 2.4: Všeobecná štruktúra evolučného algoritmu [5]

Ak máme predpoklad oblasti s najlepším výsledkom, môžeme tam umiestniť zopár inicializačných jedincov.

Následne je potrebné jedincov ohodnotiť v bloku *Evaluácia*. Výstup tohto bloku tvorí množina (distribúcia) hodnôt vhodnosti, ktorá poskytuje počiatočný obraz tvaru hodnotiacej funkcie

$$\{\Phi(a_1(0)), \Phi(a_2(0)), ..., \Phi(a_{\mu}(0))\} \tag{2.2}$$

Evolučný cyklus začína blokom *Podmienka* rozhodujúcim o ukončení alebo pokračovaní v prehľadávaní hodnotiacej funkcie. Táto podmienka zväčša signalizuje, že ďalšie hľadanie nemá význam. Ak podmienka nie je splnená pokračuje sa do sekcie *Reprodukcie* a evolučný cyklus sa opakuje.

Jedinci sú v bloku *Selekcia* podrobení výberu, na základe ktorého im bude umožnená reprodukcia. Selekcia uprednostňuje jedince s najlepšími výsledkami po evaluácii. Takéto jedince potom nazývame tiež rodičia. Pre označenie počtu vyselektovaných rodičov v generácii t zavedieme $\rho(t)$ a pre j-tého rodiča $b_j(t{=}0)$ v t-ej generácii.

Výstupom teda bude

$$P'(0) = \{b_1(t), b_2(t), ..., b_\rho(t)\} \tag{2.3}$$

Zároveň platí $b_j(t) = a_i(t), i \in < 1, \mu(t) >$ pre $\forall\ j$, pričom $a_i(t)$ môže byť vybraný za rodiča raz, viackrát alebo vôbec. Rodičia sa ďalej podieľajú na procese generovania nových jedincov (potomkov). Tento proces sa udeje pomocou genetických operátorov v rovnomennom bloku *Genetické operátory*. Výsledkom je množina potomkov

$$P''(0) = \{a_{\mu(t)+1}(t), a_{\mu(t)+2}(t), ..., a_{\mu(t)+\lambda(t)}(t)\} \tag{2.4}$$

,

kde $\lambda(t)$ pomenúva počet potomkov generovaných v t-tom prechode evolučným cyklom. Reprodukcia je ukončená blokom *Evaluácia*, ktorý plní rovnakú funkciu ako pri inicializácii.

Po reprodukčnom procese máme k dispozícii $\mu(t)$ jedincov z "rodičovskej"populácie $P(t)$ a $\lambda(t)$ potomkov populácie $P''(t)$, pričom populácie nemusia byť disjunktívne. Výber, vytvorenie novej generácie a odstránenie nepotrebných jedincov sa prevádza v bloku *Náhrada*, ktorej výsledkom je $\mu(t+1)$ jedincov novej populácie $P(t+1)$.

V prípade, že evolučný algoritmus pracuje s viacerými paralelne sa vyvíjajúcimi populáciami v bloku *Migrácia* si jednotlivé populácie môžu vymieňať informácie a jedincov. Na záver sa z novovzniknutej populácie $P(t+1)$ stáva aktuálna $P(t)$ navýšením čísla generácie $t = t + 1$. Hoci je opísaný postup realizovateľný, často sa stáva, že parametre algoritmu $\mu, \rho\ a\ \lambda$ sú pre všetky generácie konštantné [5, 8].

2.3.3 Expertné systémy

Expertné systémy (ES) majú začiatok na konci sedemdesiatych rokov. Sú to zautomatizované rozhodovacie systémy zakladajúce sa na databáze znalostí od experta. Podľa Feigenbauem (Feigenbauem a kol.,1988): *"expertné systémy sú počítačové programy, simulujúce rozhodovaciu činnosť experta pri riešení zložitých úloh a využívajúce vhodne zakódovaných, explicitne vyjadrených špeciálnych znalostí, prevzatých od experta, s cieľom dosiahnuť vo zvolenej problémovej oblasti kvality rozhodovania sa na úrovni experta"*. Pri objavení sa povedomie o systéme rýchlo rozšírilo aj mimo odborné kruhy, čo spôsobilo až nereálne očakávania. Pôvodne sa uvažovalo o vyvinutí široko využiteľných, problémovo nezávislých expertných systémov, skrátka systém s odpoveďou na všetko. V súčasnosti sa expertné systémy skôr zameriavajú na špecifické oblasti a sú súčasťou programovo rozsiahlejších celkov (tzv. embeded aplikácie) [9].

Charakteristika ES. Každý expertný systém má tzv. bázu znalostí so striktne oddelenou, vopred danou, stratégiou nakladania so znalosťami. Toto oddelenie zabezpečuje opakovateľnosť výsledkov. Báza znalostí je naplnená znalosťami experta, zachycuje ako obecné, tak aj úzko špecializované znalosti, dokonca až osobné heuristicky neisté, ktoré nie sú exaktne dokázané, ale praxou získané od daného experta (neisté znalosti). Aj takéto poznatky odlišujú experta od radového pracovníka. Báza obsahujúca takéto znalosti má charakter obecného rozhodovacieho pravidla [10].

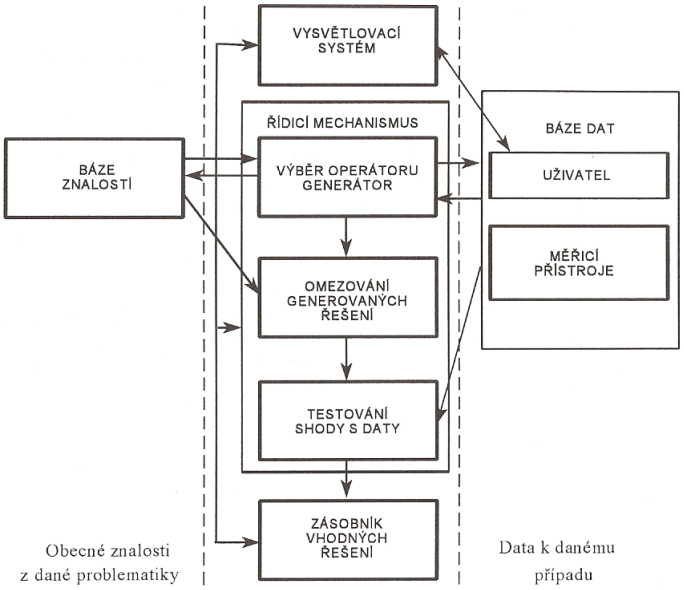
Expertný systém sa pri zisťovaní najlepšieho riešenia pýta na údaje, a na ich základe dynamicky vyberá nové otázky, ktoré najrýchlejšie vedú k najvhodnejšiemu výsledku. Tieto údaje získava ES z tvz. *bázy dát*, ktorú môže predstavovať buď ľudský užívateľ alebo systémové dáta z procesu. Dáta môžu byť neistého charakteru (odpoveď typu "asi áno","netuším"...). ES by mal byť vo všeobecnosti schopný podať riešenie aj keď nedostane úplné dáta, čo v praxi znamená, že jeden možný výsledok bude podporovaný/vyvracaný viacerými hypotézami alebo tézami. Každý krok ES a ohodnotenie riešenia by mal systém vedieť zdôvodniť [11].

Treba však povedať, že uvedený popis ES sa nie vždy dodržuje, kedže nie sú definované presné hranice zaradenia. Existujú pozmenené alternatívy s rovnomenným názvom, ktoré napríklad nepracujú s neurčitosťou [11].

Základná architektúra ES. Podľa riešených úloh je možné ES rozdeliť na:

- diagnostické - úlohou je efektívne transformovať dáta s cieľom určiť, ktorá z konečného množstva expertom zadefinovaných hypotéz najlepšie rieši daný problém. Počas procesu rozhodovania prebieha ohodnocovanie jednotlivých hypotéz a podhypotéz podľa vopred stanoveného modelu.

- plánovacie - riešený problém má známy začiatok aj koniec. Úlohou ES je nájsť optimálnu postupnosť krokov (operátorov), ktorými je možné stanovený cieľ dosiahnuť.
- hybridné - ide o kombináciu plánovacích a diagnostických ES v zmysle špecifických úloh ako sú napr, monitorovacie systémy (pri procese sa neustále spúšťa diagnostika poruchy a po jej zistení sa spustí plánovanie jej nápravy) [11].



Obr. 2.5: Bloková schéma plánovacieho expertného systému [9].

Plánovací expertný systém pozostáva z generátora možných riešení kombinujúceho postupnosti operátorov. Keďže počet variácií s počtom operátorov neúmerne rastie riadiaci mechanizmus pomocou bázy znalostí a bázy dát sa snaží tento počet čo najviac orezať (obmedzením výberu operátorov, vylúčením nereálnych alebo inak nevhodných postupností). V ideálnom prípade sú výstupom navrhované riešenia zoradené podľa ohodnotenia ES. Ďalšie detaily konštrukcie sú výrazne závislé od aplikácie [11].

Plánovací ES TEPRO. V literatúre [9] sa uvádza nasledujúci príklad:
Ide o podporu technologickej prípravy výroby v pružnom výrobnom úseku kusovej výroby, konkrétne na úseku výroby súčiastok z plochých materiálov. TEPRO má v báze znalostí širokú databázu materiálov, databázu popisujúcu schopnosti jednotlivých pracovísk, databázu typických operácií obsahujúce okrem iného indikáciu pracovísk, na ktorých je možné tieto operácie realizovať. Na základe kontextu týchto

dát sa technológovi následne ponúkajú najvhodnejšie možnosti po sebe nasledujúcich krokov. K opisu systému sa zavedie označenie:

$S = \{s_i\}$ ... množina fyzických stavov výrobkov v priebehu výrobného procesu, pričom počiatočný stav označujeme s_0, koncový s_n

$O = \{o_i\}$ ... množina technologických operácií, ktoré môžu byť vykonávané na danom úseku výroby

$W = \{w_i\}$ ... množina pracovísk

$M = \{m_i\}$ ... množina surovín alebo polotovarov na vstupe výrobného úseku

$C = \{c_i\}$ ... množina technologických a transportných obmedzení

Technologické operácie o_i v sebe nesú informáciu o transformácii fyzického stavu s_i ako aj o presune výrobku medzi medzioperačnými skladmi (zastúpené symbolom *con*). S technologickou operáciou je spojené operačné pracovisko w_i s parametrami par_i.

$$o_i : (w_i, par_i) < s_{i-1}, con_{i-1} > \longrightarrow < s_i, con_i > \tag{2.5}$$

Systém pre nájdenie najlepšieho výsledku prehľadáva dva stavové priestory F_1 a F_2. F_1 je definovaný ako

$$F_1 =< S_1, O > \tag{2.6}$$

$$S_1 = \{< s_j, con_j >\} \tag{2.7}$$

Tento priestor je možné chápať ako súbor fyzických stavov a medzistavov, ktorých premenu sprostredkúvajú operácie. Model je ale neúplný, pretože jeho úplné vypracovanie je neadekvátne zložité. Aj preto je zavedený model F_2.

$$F_2 =< S_2, T > \tag{2.8}$$

$$S_2 = \{< w_j, o_j >\} \tag{2.9}$$

$$T = \{t_j\}, t_j(con_j) :< w_j, o_j > \longrightarrow < w_{j+1}, o_{j+1} > \tag{2.10}$$

Model F_2 predstavuje výrobné zariadenia s pridruženými operáciami. V F_2 je zohľadnený medzioperačný systém s vyrovnávacími skladmi. V množine T sú definované topologické prechody medzi jednotlivými pracoviskami.

Je zrejmá výrazná previazanosť oboch priestorov. Napríklad fyzická operácia v priestore F_1 je vykonávaná na zariadení v priestore F_2. Každá akcia t je vykonaná prostredníctvom medziskladu *con*, ktorý je súčasťou popisu aj F_1. Voľba dvoch priestorov bola zvolená pre jednoduchšie definovanie obmedzení, a s tým spojené programovanie. Takto poprepájané modely vytvárajú súbor obmedzení vedúcich k zostrojeniu vhodnej postupnosti operácii P.

$$P = o_1, o_2, ...o_m, o_i \in O \tag{2.11}$$

2.3.4 Deterministické plánovanie

Problém strojového plánovania je možné charakterizovať nasledujúcim popisom dát. Máme k dispozícii n *prác* alebo *jobs* $V = \{1, 2, ..., n\}$. Každá práca j sa definuje procesným časom $p_j \geq 0$, tento čas je potrebný na dokončenie job-u. Práce musia byť spracované na m *paralelných pracoviskách*. Jednotlivé pracoviská môžu vykonávať vždy jeden job v rovnakom čase. Obdobne každý job môže byť v jednej chvíli naplánovaný iba na jednom pracovisku. Pracoviská sú neustále k dispozícii od času 0. Job $j \in V$ môže mať *release date* $r_j \geq 0$, čo je najskorší možný čas spustenia job. Nakoniec sú tu obmedzenia medzi prácami, zakódované v acyklicky orientovanom grafe $G = (V, A)$ kde V charakterizuje jobs a $(i, k) \in A, A \subset V \times V$ predstavujú usporiadanie. Zamýšľaný význam $(i, j) \in A$ je, že job J_i musí byť dokončený pred štartom job J_j. *Rozvrh* $S = (S_1, S_2, ..., S_n)$ je priradenie štartovacieho času S_j k práci J_j. Pre rozvrh platí [12]:

- rešpektuje release date $S_j \geq r_j, \forall\ j \in V$
- riadi sa grafom $G, S_j \geq S_i + p_i, \forall\ (i, j) \in A$
- v každom čase $t \geq 0$ sa v procese nevykonáva viac ako m jobs

$$|\{j \in V | S_j \leq t < S_j + p_j\}| \leq m \tag{2.12}$$

Čas ukončenia job j v rozvrhu sa označí $C_j (= S_j + p_j)$. O *čiastočnom rozvrhu* hovoríme, ak sú priradené nezáporné štartovacie časy iba pre jobs z podmnožiny $W, W \subseteq V$ a iba z tejto podmnožiny je rozvrh utváraný [12].

Definícia 2.1 (availability-dostupnosť) *Pre daný rozvrh, môžeme job j nazvať dostupným v čase t, ak boli vykonané predpoklady podľa grafu G a ak $t \geq r_j$[13].*

Úlohou plánovania je nájsť rozvrh, ktorý minimalizuje danú cieľovú funkciu. Hlavným cieľom je tzv. *makespan*, ktorý charakterizuje ukončujúci čas posledného job-u($C_{max} = max_{j \in V} C_j$) a *total weighted completion time* $\sum_{j=1}^{n} w_j C_j$, kde w_j je nezáporná váha, ktorá zohľadňuje dôležitosť daného job-u[13].

Štandardná klasifikácia. Z dôvodu širokej škály problémov, s ktorými sa v plánovaní môžeme stretnúť, bola zavedená štandardná klasifikácia od Graham, Lawler, Lenstra a Rinnooy Kan (1979). Klasifikované boli tri kategórie, označené $\alpha|\beta|\gamma$ s nasledujúcim významom:

- označením α sa špecifikujú parametre pracoviska. Napríklad $\alpha = P$ označuje model s identickými paralelnými pracoviskami. $\alpha = R$ označuje model s pracoviskami, ktorých rýchlosť s_{ij}, a s tým spojený procesný čas dokončenia job-u p_{ij} závisí od konkrétnej práce aj konkrétneho pracoviska. Závislostne $p_{ij} = p_i / s_{ij}$ pre všetky job-závislé rýchlosti s_{ij} pracovísk M_j.

- pod označením β chápeme charakteristiku job-u. Ak je prázdna, znamená to nezávislosť job-u na ostatných parametroch. Je možných mnoho alternatív: r_j ak sú definované release dates, *pmtn* znamená, že vykonávanie každého job-u môže byť prerušené v ktoromkoľvek čase a znovuobnovené na ktoromkoľvek pracovisku. Označenie d_i oznamuje, že je definovaný ukončovací čas pre každý job, v ktorom sa daná práca J_i musí ukončiť.
- označením γ sa rozumie cieľová funkcia. Ak vo funkcii uvažujeme pri každej práci o váhe w_j zapíšeme $\gamma = \sum_{j=1}^{n} w_\mathrm{j} C_\mathrm{j}$. Pre makespan $\gamma = C_\mathrm{max}$

Lineárne programovanie. Pri lineárnom programovaní hľadáme minimum účelovej funkcie v tvare:

$$minimize\ z(\mathbf{x}) = c_1 x_1 + ... + c_\mathrm{n} x_\mathrm{n} \tag{2.13}$$

Za podmienky splnenia m nerovníc:

$$\begin{aligned} a_{11}x_1 + \cdots + a_{1n}x_n &\geq b_1 \\ &\vdots \\ a_{m1}x_1 + \cdots + a_{mn}x_n &\geq b_m \end{aligned} \tag{2.14}$$

Vektor $\mathbf{x} = (x_1, ..., x_n)$ splňujúci (2.14) sa nazýva *feasible solution* (uskutočniteľné riešenie). Lineárny program je tzv. *neohraničený* ak pre každé reálne K existuje uskutočniteľné riešenie $\mathbf{x}$ splňujúce $z(x) < K$. V prípade, že lineárny program má uskutočniteľné riešenie a nie je neohraničený, má vždy optimálne riešenie [14].

Špeciálnou variantou lineárneho programovania je tzv. *transshipment problem* (transportný problém). Nech $G = (V, A)$ je orientovaný graf s vrcholmi $V = \{1, ..., n\}$ a prepojeniami A[13]. Transportný problém je daný:

$$minimize\ z(\mathbf{x}) = \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{2.15}$$

$$\sum_{j,(i,j)\in A} x_{ji} \sum_{j,(i,j)\in A} x_{ij} = b_i\ pre\ \forall i \in V \tag{2.16}$$

$$l_{ij} \leq x_{ij} \leq u_{ij}\ pre\ \forall (i,j) \in A \tag{2.17}$$

Graf G je možné v tomto prípade chápať ako transportnú sieť. Hodnota b_i predstavuje požiadavku $b_i > 0$ alebo dodávku $b_i < 0$ produktu na vrchole i. Jednotlivé prepojenia (i, j) vrcholov môžu byť rôzne zaťažené cenou c_{ij}. Množstvo produktu transportované môže byť zhora u_{ij} aj zdola l_{ij} ohraničené [13].

Úlohou algoritmu je nájsť množstvo produktu x_{ij}, ktoré sa bude transportovať po cenovo ohodnotených prepojeniach z vrcholov s dodávkou k vrcholom s požiadavkou. Z fyzikálneho hľadiska musí platiť $\sum_{i=1}^{n} b_i = 0$. Ak $b_i = 0\ pre\ \forall\ i \in V$ tak

problém nazývame *circulation problem*. Štandardné algoritmy pre riešenie transportného problému sú network simplex method a out-of-kilter algorithm[13].

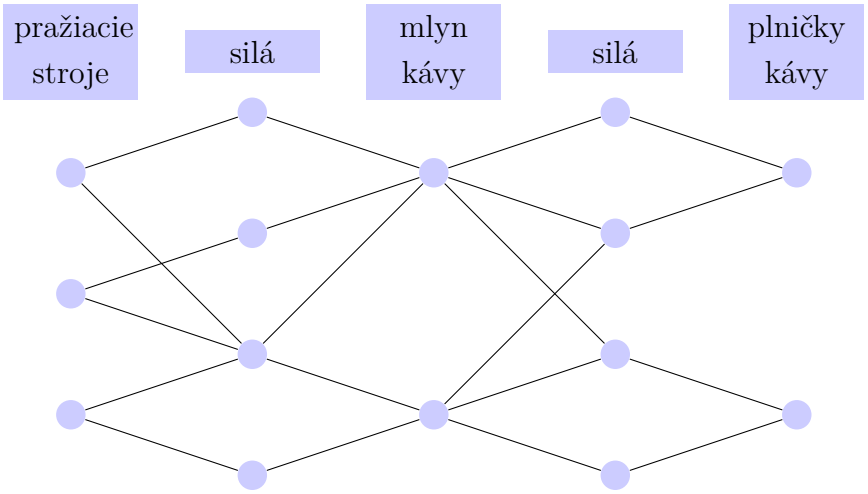
3 VÝBER PLÁNOVACIEHO ALGORITMU PRE OPTIMALIZÁCIU V PKZ

3.1 Popis zadaných kritérií pri tvorbe plánu výroby pražených kávových zmesí

Predtým, ako bude stanovená plánovacia úloha s cieľom optimalizovať výrobu pražených kávových zmesí, je nutné, pre lepšie pochopenie danej problematiky, zozbierať všetky dostupné informácie o výrobnom procese. Systém na výrobu kávových zmesí má tieto hlavné časti :

- pražiace stroje,
- zrecie silá,
- mlyn,
- odplyňovacie silá,
- plničky.

Zmes je prepravovaná prostredníctvom dopravníkov do jednotlivých strojných zariadení. Výrobu je schematicky zobrazená na obr. (obr.3.1/s29).



Obr. 3.1: Príklad možného usporiadania technológie

Výroba kávových zmesí sa začína vložením vysušených kávových bobúľ do pražiacich strojoch, kde sa uskutočňuje samotné praženie kávy. Systém je tvorený tromi pražiacimi strojmi, ktoré sú určené na spracovávanie rôznych kávových zmesí. Tieto zmesi sú do strojov vkladané v konštantných dávkach (cca 100 kg). Možné je aj praženie rovnakej kávovej zmesi na viacerých strojoch súčasne. Pri pražení je nevyhnutnou podmienkou dodržanie stanovenej doby praženia kávy ako aj teploty

praženia. Ak by praženie neprebiehalo dostatočne dlho, alebo by prebiehalo pri nízkej teplote, na povrch kávových zŕn by nevystúpili aromatické oleje, a káva by mala v dôsledku toho mdlú a nevýraznú chuť. Naopak, ak by kávové zrná boli pražené príliš dlho, alebo pri vysokej teplote, káva by mala slabú chuť. Výsledkom praženia je medziprodukt, ktorý je následne automaticky transportovaný do zrecieho sila tak, aby nedošlo k vzájomnému premiešaniu rôznorodých kávových zmesí. V silách produkt vyčkáva presne definovaný čas. Tento čas je ohraničený časovými hranicami zhora aj zdola (napr. minimálny zrecí čas je 16 h a maximálny 20 h).

Zo zrecích síl upražená káva putuje do mlynov, kde sa káva zomelie na vopred stanovenú veľkosť zŕn. Zomletím sa reakčný povrch kávového zrna mnohonásobne zväčší, čoho dôsledkom je jednoduchšie uvoľňovanie kávovej arómy. Zomletá kávová zmes putuje do odplyňovacieho sila. Rovnako aj tu medziprodukt zotrváva určitý čas, definovaný horným a dolným ohraničením. Po uplynutí definovaného času odplynenia kávy nasleduje jej plnenie, ktoré umožňujú plničky nachádzajúce sa za príslušným silom.

Počas celej doby spracovania kávových zmesí musí byť dodržaná podmienka, že jednotlivé zmesi vyskytujúce sa v silách, v prípade prípravy viacerých druhov zmesí súčasne, nesmú prísť do vzájomného kontaktu. Preto má každá zmes vyhradené samostatné silo, respektíve silá, v závislosti od množstva pripravovanej zmesi. Jednotlivé (technologické uzly TU) počty strojov, počty síl, dopravné cesty, druhy kávových zmesí budú, pre potreby tejto práce, variabilné a budú vopred zadefinované na začiatku každého plánovacieho procesu. Pojem dopravné cesty charakterizuje prepojenia medzi jednotlivými fázami prípravy kávy. Na (obr.3.1/s29) sú vyobrazené dopravné cesty reprezentované spojnicami modrých blokov.

3.2 Návrh plánovacieho algoritmu

Popísaná problematika sa mi ťažko identifikovala s existujúcimi modelmi z literatúry. V skutočnosti obsahuje množstvo obmedzení, ktoré znemožňujú jednoduché testovanie operačných kombinácii, pretože množstvo z týchto variantov by nebolo ani len realizovateľných. Napr. kombinácia, v ktorej by sa naplnilo priveľa zrejúcich síl a ignorovala by sa skutočnosť, že sa nestihnú vyprázdniť v definovanom čase je nepoužiteľná.

Jednotlivé začiatky operácii sú závislé nielen od seba navzájom, ale aj od zvolenej kombinácie použitých dopravných ciest a nutnosti dodržať definované doby odpočinku produktu.

Preto som vytvoril model technológie. Model v ňom simuluje plán výroby každého technologického uzlu. Pred začiatkom operácie praženia sa najprv overí do-

stupná kapacita v nadväzujúcich fázach (snaha vyhnúť sa kolíziám). Ak sa taká možnosť nájde, model rezervuje danej dávke príslušné vybavenie a operácia sa spustí. Rozhodnutie, ktorý pražiaci stroj (v prípade, že je ich voľných viac) sa spustí ako prvý, bude riadené pomocou action listu. Action list predstavuje súbor parametrov alebo pravidiel, podľa ktorých sa model správa. Voľba trasy, akou sa dávka kávy spracuje, môže byť taktiež riadená action listom.

Súbor pravidiel v Action liste predstavuje jedinca v evolučnom algoritme. Následne som jedinca hodnotil v modeli technológie podľa kritéria najkratších prestojov. Evolučný algoritmus na základe selekčného mechanizmu vyberá riešenia (rodičov), ktoré sa určia ako východiskové. Nasleduje proces kríženia kombinujúci parametre rodičov, inak povedané pravidlá v Action liste. Pre zachovanie diverzibility pri prehľadávaní nechýba mutačný proces pozmeňujúci náhodné pravidlá v rámci definovaných kritérií.

3.3 Vytváranie modelu technológie a špecifikácie operácii

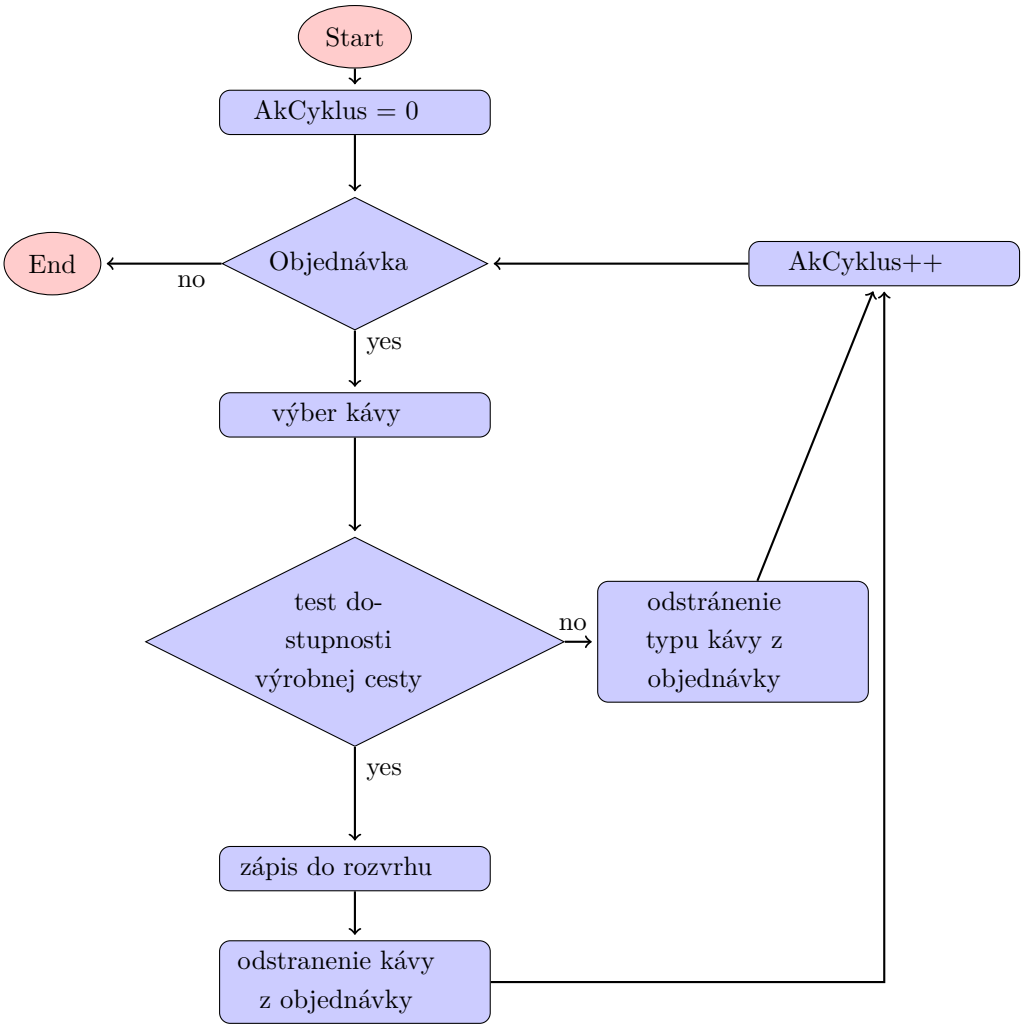
Môj model technológie sa skladá z piatich tzv. operácii a pomocného objektu zastupujúceho úlohu počiatku/skladu. Jeho vytváranie prebieha zľava doprava, čiže najprv sa vytvorí inštancia objektu sklad, ďalej objekty zastupujúce funkciu pražičiek kávy, zrecie silá, mlyny, odplyňovacie silá a nakoniec plničky na kávových zmesí. Ich počet je variabilný. Jednotlivé inštancie pražičiek, mlynov atď. sa tiež nazývajú technologickými uzlami (TU).

Pri vytváraní jednotlivých operácii si užívateľ podľa číselníka z užívateľského rozhrania určuje prepojenia jednotlivých uzlov. V konečnom dôsledku má každý technologický uzol uložené referencie na všetky bezprostredne pripojené inštancie zozadu aj spredu.

Jednotlivé špecifikácie operácii, ktoré som pri vytváraní modelu bral do úvahy:

- praženie prebieha v dávkach v krátkych intervaloch, takže v okamihu začatia praženia je vhodné mať voľné silo. Zrecia doba sa pre zjednodušenie počíta už od okamihu začatia praženia prvej dávky,
- jednotlivé typy káv sa nedajú pražiť na všetkých pražičkách v prílohe B.1,
- vyprázdnenie zrecieho sila a jeho zomletie po trase sa vykoná v cca 10tich minútach, a preto je ho možné v plánovaní na celé hodiny zanedbať,
- jedno zrecie silo sa vyprázdni do 4-och odplyňovacích síl,
- po dovŕšení nevyhnutného času na odplynenie v odplyňovacom sile je rozvrh tohto sila obsadený až po úplné vyprázdnenie na príslušnú plničku.

3.4 Popis fungovania plánovania na modele technológie



Obr. 3.2: Vývojový diagram fungovania modelu technológie

Mnou navrhnuté plánovanie na modele technológie popisuje vývojový diagram na (obr.3.2/s32).

Celé plánovanie výroby kávových zmesí sa odohráva v cykloch. Na začiatku je nastavený aktuálny cyklus (AkCyklus) na nule. Následne je ukončovacou podmienkou testovaný zoznam objednávok kávy. Plánovanie je ukončené po vyprázdnení zoznamu objednávok. V najlepšom prípade sa naplánujú všetky objednávky a AkCyklus dosiahne MaxCyklus.

Po splnení podmienky o existencii objednávky prichádza na rad výber typu kávovej zmesi. Ten sa riadi evolučným pravidlom závislým od aktuálneho cyklu, ktorý je

vysvetlený v kapitole 3.1. Postupne sa testuje dostupnosť výrobnej cesty pre každú jednotlivú objednávku kávovej zmesi na modele technológie. Ak sa pre konkrétnu objednávku výrobná cesta nenájde, daný typ kávy je nevyrobiteľný, a tým pádom sa vymažú všetky požiadavky na tento daný typ. Aktuálny cyklus sa inkrementuje, podľa nového evolučného pravidla sa vyberie objednávka iného typu kávovj zmesi zo zoznamu a znova sa testuje dostupnosť jej výrobnej cesty.

Naopak pri úspešnom nájdení výrobnej cesty/ciest sa pristúpi k zápisu jednej z možných trás do rozvrhu všetkých dotknutých technologických uzlov. Výber konkrétnej trasy je opäť sprevádzaný sériou evolučných pravidiel alebo je trasa deterministicky vybraná podľa pravidla najskoršie dostupného technologického uzlu. Po úspešnom zápise do rozvrhu sa odstráni požiadavka na výrobu danej dávky kávovj zmesi konkrétneho typu zo zoznamu objednávok, aktuálny cyklus sa zväčší o jedna a proces sa opakuje.

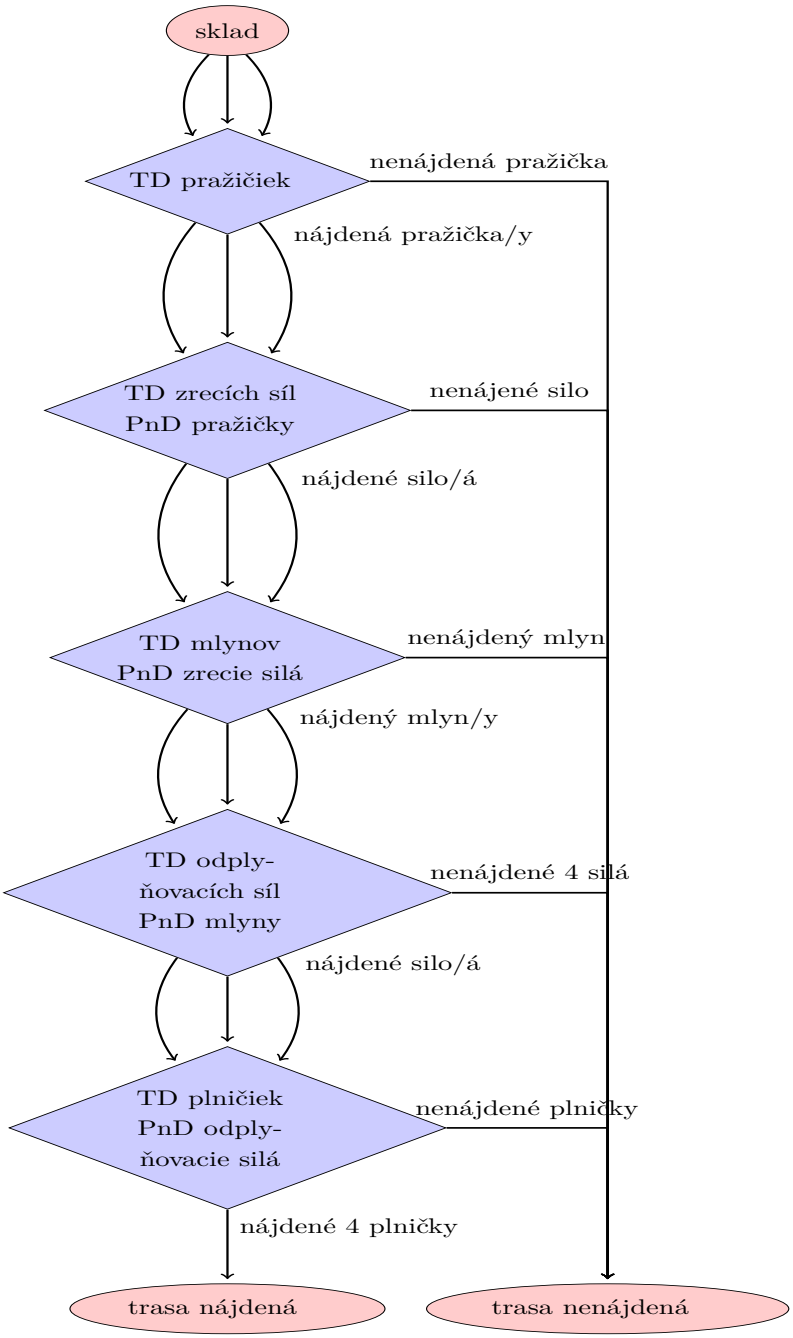
3.5 Testovanie dostupnosti výrobnej cesty

Po vybraní kávovej dávky, mnou navrhovaný algoritmus na plánovanie, si najprv otestuje výrobné kapacity. Táto dávka predstavuje objem jedného zrecieho sila. Každý TU má vlastný rozvrh výroby, a plánovaním sa testuje využitie voľných výrobných kapacít. Testovacia funkcia každého technologického uzla spočíva v dvoch krokoch.

- testovanie vlastného rozvrhu technologického uzla na voľné výrobné kapacity (testovanie dostupnosti TD).
- testovanie funkcií všetkých pripojených technologických uzlov z nasledujúcej operácie, ak je predchádzajúci krok úspešný (pripojených na dostupné PnD). Ilustrované na (obr.3.3/s34)

Pri testovaní vlastného rozvrhu je vždy kladená otázka na konkrétny časový úsek, v ktorom je hľadaný kratší alebo rovnako dlhý časový úsek na vykonanie operácie. Napríklad je hľadaný 8 hodinový voľný časový úsek od pondelka 6:00 hod do utorka 23:00 hod. Prvý krok vracia celý zoznam intervalov (ďalej Zoznam), ktoré vyhovujú podmienke. Ako prvé sú uvedené intervaly, pred ktorými je naplánovaná už nejaká činnosť, a až potom intervaly vytvárajúce v testovacom uzle postupne narastajúcu medzeru.

V druhom kroku je vygenerovaný nový časový interval, v závislosti od nasledujúcej operácie, ktorý následne kladie otázku už spomínaným pripojeným uzlom. Pre názornosť uvádzam príklad: zrecie silo má voľné kapacity od pondelka 7:00 hod, operácia potrvá do 15:00 hod toho istého dňa, pričom sa zrecie silo potrebuje vyprázdniť do odplyňovacieho sila do stredy 03:00 hod (dané receptom alebo v danom



Obr. 3.3: Vývojový diagram testovacej fázy modelu. TD-testovanie dostupnosti, testovanie PnD-pripojených na dostupné

okamihu je už naplánovaná iná činnosť). Následne má dávka v odplyňovacom sile vyčkať 6 hodín. Vygenerovaný interval má hranice od Po 15:00 hod do St 09:00 hod, hľadajúcim je 6 hodinový voľný úsek. Ak sa v druhom kroku nevráti dostatočný počet kladných odpovedí z pripojených uzlov na interval vygenerovaný na základe prvého intervalu zo Zoznamu z prvého kroku, vygeneruje sa nový testovací interval

na základe druhého intervalu zo Zoznamu z prvého kroku. Celý proces testovania sa končí pri operácii plničiek, ktoré testujú už iba vlastný rozvrh.

Každá úspešná odpoveď od nasledujúcich technologických uzlov sa spolu s informáciou odkiaľ prišla požiadavka na voľnú kapacitu ukladá. Rozlišuje sa, či prišla otázka od prvej pražičky, druhého zrecieho sila a prvého mlynu, alebo od druhej pražičky, druhého zrecieho sila a druhého mlynu. Účelom je, aby pri zostavovaní zápisu do rozvrhu, bol možný výber pri rozhodovaní už z predpripravených možností. Tieto možnosti sú tak závisle od tzv. cesty zápisu do rozvrhov.

3.6 Zápis výrobnej cesty do rozvrhov modelu

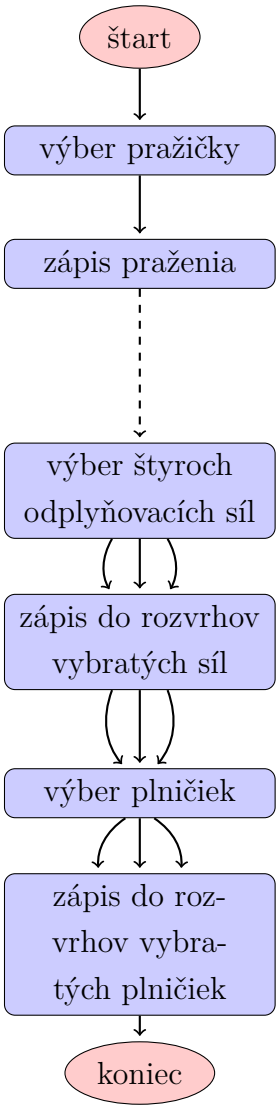
V tomto štádiu sa môj model technológie nachádza vo fáze zápisu do rozvrhu výroby. Existuje teda výrobná cesta pre daný typ a dávku kávovej zmesi od pražičiek až po plničky na kávové zmesí. Plánovanie sa začína rozhodnutím, na ktorej pražičke sa bude dávka kávovej zmesi pražiť. Overené možnosti má inštancia objektu sklad uložené z fázy testovania, čiže si nemôže vybrať neuskutočniteľnú variantu. O tom, ktorú si vyberie, rozhoduje evolučné pravidlo. Pri každom rozhodovaní o type kávovej zmesi na praženie rozhoduje evolučné pravidlo fungujúce na rovnakom princípe. Výber trasy môže, ale aj nemusí byť riadený evolučne (záleží od zvoleného nastavenia).

Definícia 3.1 (Evolučné pravidlo) *Pravidlo predstavuje evolučné číslo (*EČ) z intervalu $< 0; 1 >$. Počet možností, z ktorých sa vyberá, si rozdelí tento interval na subintervaly o rovnakej veľkosti. Evolučné číslo pripadne len do jedného takého subintervalu, a tým rozhodne o vybranej možnosti.

Teda ak sa vyberá z desiatich možností a evolučné pravidlo predstavuje číslo 0,4 vyberie sa štvrtá možnosť. Výber trasy neevolučným spôsobom je riadený deterministickým pravidlom.

Definícia 3.2 (Deterministické pravidlo) *Pravidlo vyberá z poskytnutých možnosti ten technologický uzol, ktorý ponúka najskoršiu dobu vykonania.*

Zápis do rozvrhov modelu technológie pomocou evolučného pravidla som naprogramoval nasledovne: inštancia objektu sklad si na základe evolučného čísla vyberie pražiaci stroj, ten si na základe vlastného EČ vyberie zrecie silo atď. až po plničky. Výnimku tvorí inštancia mlyn, ktorá si vyberá 4 odplyňovacie silá na vyprázdnenie zrecieho sila.



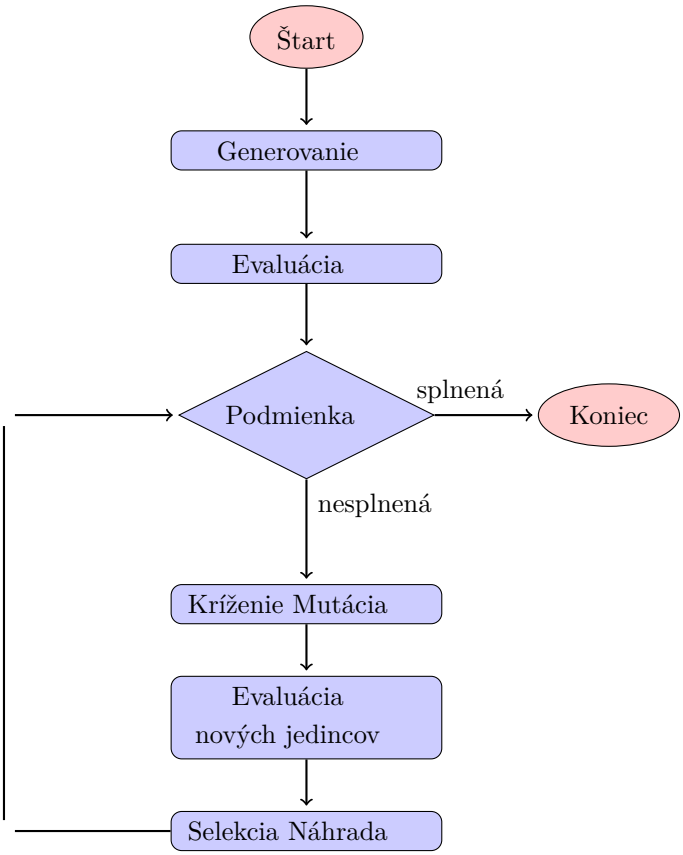
Obr. 3.4: Vývojový diagram zápisu rozvrhu výroby vybranej kávy

3.7 Použitý evolučný algoritmus

Evolučný algoritmus opísaný na obrázku (obr.3.5/s37) začína vygenerovaním a ohodnotením jedincov populácie. Jedinca reprezentuje pole čísel typu double na intervale <0;1>. Pri evaluácií je každý technologický uzol odkázaný na určitú časť tohto poľa v závislosti od typu vykonávanej operácie (vždy ide o dĺžku v násobkoch MaxCyklu). Hodnotiaca funkcia/fitness funkcia je vo forme:

$$fitness = 120 * C - 2 * M - 5 * X \tag{3.1}$$

C- počet upražených dávok kávových zmesí
M- súčet dĺžok voľných intervalov



Obr. 3.5: Vývojový diagram evolučného algoritmu

X- počet voľných intervalov
Inštancia objektu sklad sa v každom cykle rozhoduje 2 krát. Raz pri určovaní typu praženej kávovej zmesi a druhý raz pri výbere pražičky na praženie. Z tohto dôvodu si tento objekt rezervuje prvých 2*MaxCyklus čísel z poľa reprezentujúceho jedného jedinca. Inštancia mlyn si vyberá štyri odplyňovacie silá na vyprázdnenie zrecieho sila. Jeho "chromozómödkazuje na pole dĺžky 4*MaxCyklus. Ostatné technologické uzly majú "chromozómö veľkosti MaxCyklus.

Podmienkou na ukončenie evolučného deja je dosiahnutie požadovaného počtu cyklov.

Kríženie jedincov v populácii nastáva:

- zámenou dvoch reťazcov čísel na náhodnom mieste a náhodnej dĺžke
- pričítavaním alebo odčítavaním rozdielu dvoch čísel z reťazca na rovnakom mieste s dôrazom na neprekročenie hraníc intervalu <0;1>. Kríženie cez celú dĺžku poľa.

Oboma spôsobmi vzniknú nové dva jedince, ktoré sa pričlenia do populácie.

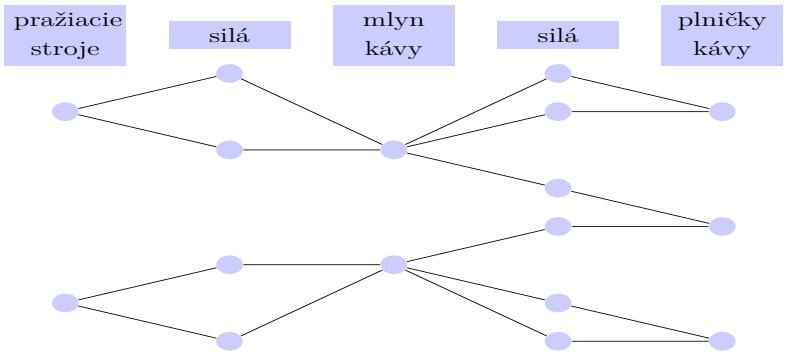
Pri mutácií sa vyberajú náhodné čísla z postupnosti čísel a podobne ako pri

krížení druhého typu sa pričítava alebo odčítava tentoraz náhodné číslo z intervalu <-0,5; 0.5>. Pričítava sa v prípade, ak takýto výsledok neprekročí interval <0;1>. V prípade neúspechu sa otestuje odčítanie za rovnakých podmienok. Je možná aj situácia nesplnenia oboch testov, ktorá spôsobí preskočenie mutovania daného čísla. Selekčné mechanizmy fungujú na všeobecne známych princípoch:

- Elite Selection- zoradenie jedincov podľa fitness funkcie a odstránenie horšej polovice. Pre účel aplikácie tento mechanizmus tiež vyraďuje rovnakých jedincov a jedného z troch s rovnakou fitness funkciou.
- Rank Selection- zoradenie jedincov podľa fitness funkcie s následným náhodným výberom, pričom pravdepodobnosť výberu je úmerná poradiu.
- Roulette Wheel Selection - náhodný výber pričom pravdepodobnosť výberu je úmerná ohodnoteniu fitness funkciou.

3.8 Testovanie aplikácie plánovania

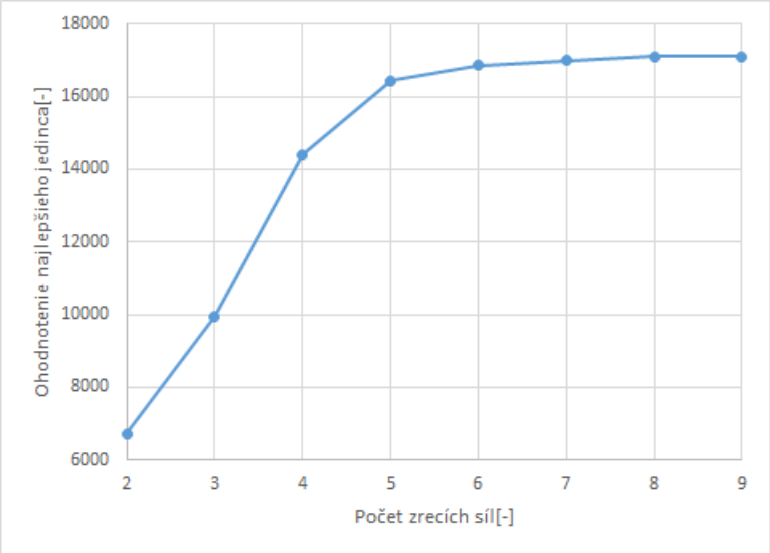
Testovanie som vykonával na notebooku Lenovo ThinkPad Edge E540(64-bitový operačný systém, 8GB RAM, intel(R) Core(TM) i7-4702MQ CPU@ 2,20GHz). Samotný výpočet plánovacieho algoritmu prebiehal na jednom vlákne pri nastavenom maximálnom výkone notebooku. Z dôvodu lepšej čitateľnosti výsledkov som testy



Obr. 3.6: Príklad symetrického, nezdieľajúceho zapojenia technológie

vykonával prevažne na konštantnom počte pražiacich strojov(3), plničiek(10) mlecích strojov(1). Pri testoch, ktoré neskúmajú výsledky v závislosti od počtu síl bol ich počet konštantný (6 zrecích síl, 20 odplyňovacých síl). Požiadavka bola definovaná na 8 druhov kávových zmesí, z každej 19 dávok (jedna dávka predstavuje veľkosť zrecieho sila). Parametre daných kávových zmesí boli zadané, a sú uvedené v (prílohaB.1/s56)(prvých 8). Ak nie je zadané inak prepojenia medzi jednotlivými operáciami sú symetrické a rozdelené rovnakým dielom (obr.3.6/s38). Populáciu som

Graf 3.1 ( ) *Závislosť fitness hodnoty najlepšieho jedinca v populácii od počtu zrecích síl*

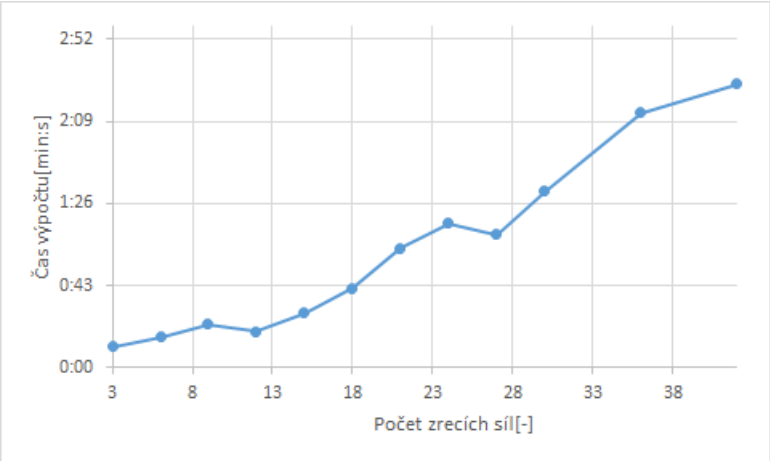


nastavil na 20 jedincov, rovnako tak počet evolučných cyklusov, preferovaný selekčný mechanizmus:Elite Selection. Opísané usporiadanie možno nazvať základné.

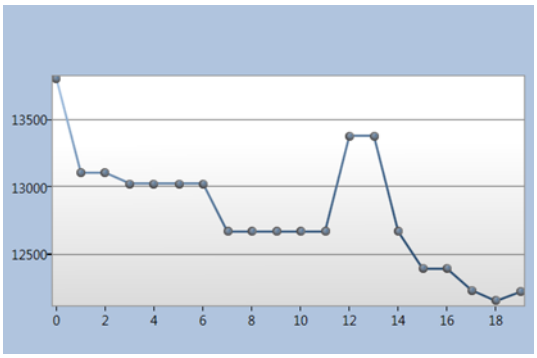
Z testovania (prílohaA/s54)) je možné konštatovať nasledovné závery:

- s rastúcim počtom sil jednej operácie (zrenie/odplynenie) sa zlepšuje výsledok plánovania, a to až do chvíle vytvorenia "bottleneck"(jav, pri ktorom výkon alebo schopnosť celého systému je obmedzená jedným alebo obmedzeným počtom dielcov alebo zdrojov) na inej operácii. Toto tvrdenie ilustruje graf 3.1 vytvorený z výsledkov 43-50 z tabuľky (prílohaA.2/s55).
- pri prekrývaní prepojení susedných technologických uzlov v jednej operácii (napr. plnička 2 napojená na 3 silá zdieľa 2 silá so susedmi) dochádza k prvotným zhoršeniam výsledkov. S ďalším nárasom zdieľaní sa trend obracia a vylepšuje základné usporiadanie. K prvotnému zhoršeniu dochádza pretože algoritmus nedokáže identifikovať nezdieľané silá a napĺňa všetky silá rovnomerne. Z tohto dôvodu dochádza k situáciám, v ktorých je naplnené zdieľané silo, jedna plnička z neho plní a druhá nemá pripravené nezdieľané silo.
- pri navyšovaní TU v jednej operácií sa predlžuje výpočtový čas. Charakter tejto závislosti je lineárny, keďže aj počet nutných výpočtov rastie priamoúmerne. Závislosť je ilustrovaná na výsledkoch 7 až 18 z tabuľky (prílohaA.2/s55) v grafe 3.2.
- je badať zámennosť počtu evolučných cyklov s veľkosťou populácie v evolučnom mechanizme. V oboch prípadoch je potrebné vykonať rovnaký počet výpočtov (buď 20 krát ohodnotím 40 pozmenených jedincov alebo 40 krát 20-tich). Ich časová náročnosť aj výsledný plán výroby sú totožné.

Graf 3.2 ( ) *Závislosť výpočtového času od počtu zrecích síl*



- neskoršie dotvárané deterministické vyberanie trasy vykazuje lepšie výsledky vo všetkých skúmaných prípadoch. Dokonca aj pri testovaní zdieľania technologických uzlov, kde existuje predpoklad odhalenia vhodnejšej trasy evolučným spôsobom. Hlavný problém sa zrejme nachádza v spôsobe kríženia a mutácie jedincov v populácii.
- najlepšie výsledky vykazoval elitistický selekčný mechanizmus. Selekcia pomocou Roulette Wheel Selection a Rank Selection počas evolučných cyklov zabúda najlepšie riešenia a už sa k nim nedokáže vrátiť. Toto tvrdenie podporuje obrázok (obr.3.7/s40). Zo selekcii Rank a Roulette Wheel Selection má lepšie výsledky Rank selekcia, keďže model technológie generuje podobne ohodnotené plány výroby.

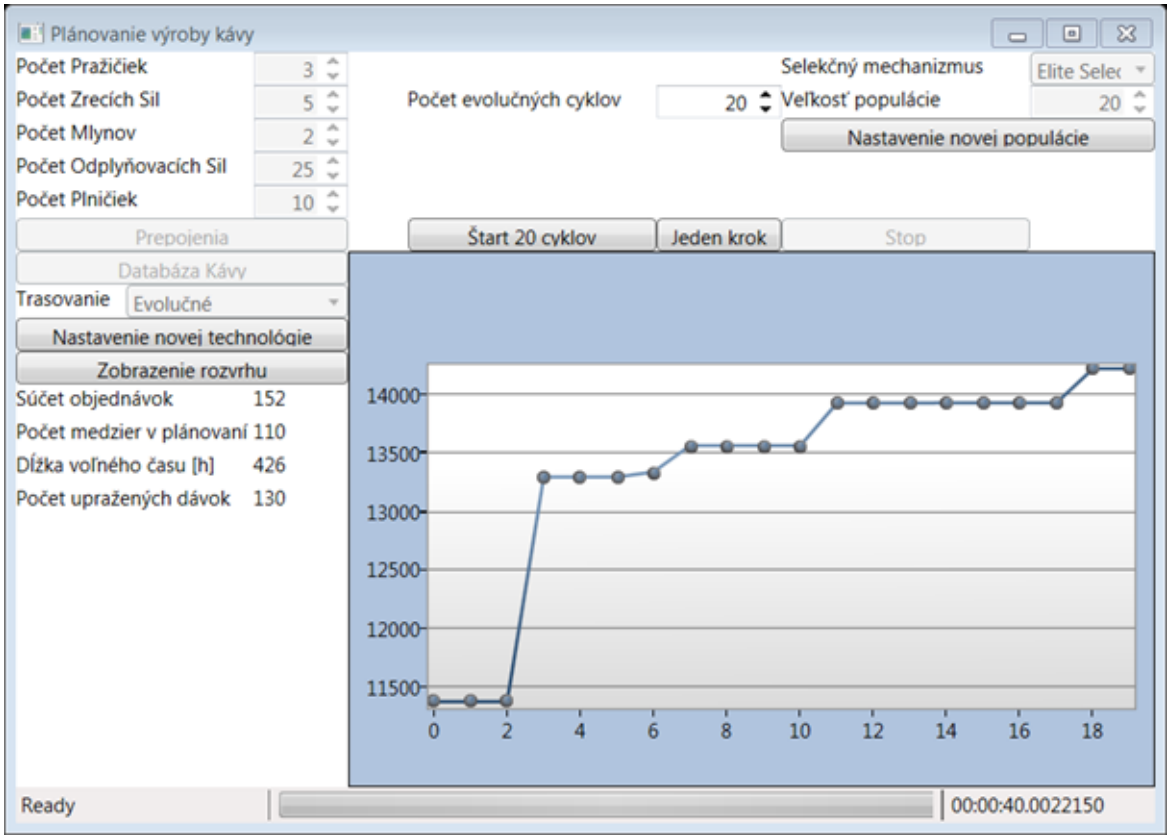


Obr. 3.7: Výrez z aplikácie. Závislosť ohodnotenia najlepšieho jedinca v populácii od evolučného cyklu. x-ová os číslo evolučného cyklu, y-ová ohodnotenie najlepšieho jedinca v populácii. Použitie Roulette Wheel Selection.

Reálna technológia obsahuje prídavné obmedzenia zahrňujúce výrobné prestávky a

odstávky, ktoré môj model nezohľadnuje. Preto som výsledné plány výroby neporovnával so skutočnosťou.

3.9 Popis aplikácie



Obr. 3.8: Hlavné okno plánovacej aplikácie

Aplikačné grafické rozhranie som programoval v xaml jazyku. Pri popise aplikácie som postupoval z ľavého horného rohu po stĺpcoch doprava. WPF grafické rozhranie bolo naprogramované technikou MVVM.

Číselník jednotlivých operácii slúžiaci na opis technológie, na ktorej som plánoval.

Tlačidlo *Prepojenia* vygeneruje päticu okien s tabuľkami prepojení medzi operáciami.

Tlačidlo *Databáza Kávy* poskytne náhľad na uloženú databázu káv aj s ich parametrami. Možná je editácia: mazanie, pridávanie, úprava káv v databáze. Rovnako sa v tomto okne zadávajú objednávky jednotlivých typov káv/kávových zmesí.

V rozbaľovacom boxe s popisom *Trasovanie* je možné rozhodnúť o spôsobe výberu plánovacej trasy.

Tlačidlo *Vytvor Model Technológie/Nastavenie novej technológie* na základe nastaveného počtu technologických uzlov a ich vzájomného prepájania vytvorí model technológie na testovanie. Po odkliknutí zablokuje možnosti editácie technológie až do doby opätovného stlačenia tlačidla, ktoré umožní vytvoriť nový model technológie.

Tlačidlo *Zobrazenie rozvrhu* slúži na zobrazenie aktuálne vygenerovaného rozvrhu.

Ďalej nasleduje pár informačných riadkov zobrazujúcich špecifikácie aktuálneho riešenia plánu výroby. Menovite ide o:

- *Súčet objednávok* - množstvo dávok všetkých kávových zmesí určených na výrobu
- *Počet medzier v plánovaní* - číselne zobrazený počet miest v rozvrhoch plničiek s nedefinovanou činnosťou (medzery); dĺžka medzery nezohráva úlohu
- *Dĺžka voľného času [h]* - sčítané všetky dĺžky medzier v rozvrhoch plničiek
- *Počet upražených dávok* - zobrazuje počet dávok kávovej zmesi, ktoré sa podarilo naplánovať

V pravej časti okna sú umiestnené nastavenia a ovládacie prvky evolučného algoritmu zaisťujúceho nájdenie riešenia. Ako parameter je meniteľný *Počet evolučných cyklov*, počas ktorých budeme hľadať riešenie.

Selekčný mechanizmus sa vyberá z rozbaľovacieho boxu vpravo hore. Na výber sú 3 typy: elitistický; RankSelection (s prihliadaním na poradie); RouletteWheelSelection (s prihliadaním na ohodnotenie modelom).

Tlačidlo *Nová populácia/Nastavenie novej populácie* inicializuje prvotnú náhodnú populáciu jedincov. Podobne ako pri tlačidle pre vytvorenie modelu technológie aj tu dôjde k zablokovaniu možností editácie populácie do doby opätovného stlačenia.

Tlačidlo *Štart XX cyklov* odštartuje nastavený počet cyklov z číselníka *Počet evolučných cyklov*.

Tlačidlo *Jeden krok* spustí jeden evolučný cyklus, slúži na doladenie výsledku.

Tlačidlo *Stop* ukončí aktuálne prebiehajúci výpočet, treba však počkať na ukončenie aktuálneho evolučného cyklu

Graf umiestnený vpravo dole zobrazuje ohodnoteného najlepšieho jedinca z populácie, aktualizuje sa každý evolučný cyklus.

V spodnej časti aplikácie je lišta s troma komponentami

- úplne vľavo sa zobrazuje aktuálny stav (Pripravený, Počítam, Čakám na ukončenie evol. cyklu).
- nasleduje progress bar - odzrkadľujúci progres zadanej úlohy
- posledné sú stopky, spočítavajúce trvanie aktuálne zadanej úlohy

Obsluha aplikácie je nasledovná:

- výber počtu pražičiek, sil, mlynov a plničiek. Ak sú počty navzájom deliteľné bez zvyšku, aplikácia vygeneruje prepojenia. Menší počet technologických uzlov jednej operácie si rovnomerne, prerozdelí pripojenia s druhou operáciou ako je uvedené napríklad na (obr.3.6/s38).
- korekcia alebo doplnenie prepojení TU
- vytvorenie modelu technológie
- voľba selekčného mechanizmu, veľkosť populácie, spôsob trasovania
- vytvorenie novej populácie jedincov
- spustenie výpočtu tlačidlom *Štart XX cyklov* alebo *Jeden krok*.
- možné zastavenie tlačidlom *Stop*
- prezeranie výsledkov po stlačení *Zobrazenie rozvrhu* prípadne pridanie ďalších evolučných cyklov.

3.10 Možné vylepšenia

Algoritmus je možné ďalej vylepšovať a optimalizovať:

- rozdelenie prepočtov v modele do viacerých vlákien, ktoré sa budú prepočítavať paralélne na procesorových jadrách PC.
- zavedenie váhového systému v jednotlivých TU, ktorý by zohľadňoval ich možnosti vzhľadom na množstvo prepojení s ostatnými TU a zohľadňoval zdielané TU.
- zavedenie prvkov elitizmu aj do ostatných selekčných mechanizmov.
- jemnejšie zachádzanie s jedincami populácie pri krížení a mutácii (zmena EČ maximálne o 20%). Bolo by vhodné rozlišovať medzi evolučným pravidlom, ktoré rozhoduje o výbere z troch možností (napr. pri výbere pražičiek) a dvadsiatich možností (pri výbere odplyňovacích síl).
- umožnenie vyprázdnenia jednej dávky zrecieho sila naraz aj cez dva mlyny, poprípade do jedného odplyňovacieho sila s čakaním na jeho vyprázdnenie.
- keďže model technológie v konečnej fáze (cca v dvadsiatom evolučnom cykle) generuje veľmi podobné rozvrhy pre plán výroby, bolo by dobré umožniť náhľad na tieto rozvrhy.

4 REALIZÁCIA ALGORITMOV V JAZYKU C# AKO FUNKCIE POUŽITEĽNEJ V SYSTÉME COMES PRE OPAKOVANÉ VYUŽITIE PRI IMPLEMENTÁCII MODULU COMES MODELLER

COMES Modeller umožňuje dva spôsoby tvorby a použitia funkcií.

- 1. Funkcie sa napíšu ako C# kód (jednoduché funkcie alebo triedy) priamo v textovom editore COMES.
- 2. Naprogramovaný C# kód (jednoduché funkcie alebo triedy) sa skompiluje do dll knižnice, ktorá sa následne pridá do konfiguračného súboru COMES Modeller a potom sú v COMES Modeller dostupné všetky public triedy a funkcie z pridanej knižnice.

[2]

Rozhodol som sa pre variantu druhú, a preto samotné riešenie pozostáva z dvoch dll knižníc a xml súboru s databázou kávových zmesí v jednom adresári. Na vytvorenie modelu technológie slúži knižnica TechnologySimulation.dll a v nej obsiahnutá trieda TechSim. Táto trieda predstavuje fitness funkciu, ktorou sa ohodnotia jedinci z populácie v genetickom algoritme. Trieda Population z knižnice GeneticAlgorithm.dll zapuzdruje správu nad týmito jedincami. Táto knižnica vznikla modifikovaním verejne dostupných zdrojových kódov zo stránok AForge.NET [17].

Konštruktor triedy TechSim

```
public TechSim(List<ushort> pOrder,int pNumOfRoast, int
    pNumOfSiloRest,int pNumOfMill,int pNumOfSiloDegass,int
    pNumOfFillMach, bool[] pSelect,ushort pTracingIndex)
```

požaduje nasledujúce parametre:

- pOrder predstavuje zoznam objednávok jednotlivých typov kávy určených pre plánovanie (jedno naplnené zrecie silo sa rovná jednej dávke). Zoradené sú rovnako ako v xml databáze.
- pNumOf... nasleduje päť parametrov určujúcich počty technologických uzlov v jednotlivých operáciách. Menovite: počet pražičiek, zrecích síl, mlynov, odplyňovacích síl, plničiek.
- pSelect je pole binárnych hodnôt určujúce prepojenia TU. Prvá hodnota označuje prepojenie prvej pražičky a prvého zrecieho sila, druhá hodnota označuje prepojenie prvej pražičky s druhým zrecím silom atd. V prípade hodnoty "true"prepojenie existuje, inak neexistuje.

- pTracingIndex rozhoduje o voľbe plánovacej trasy. V prípade zadania 0 sa vyberie Evolučné trasovanie, inak sa zvolí Deterministické.

Trieda TechSim má ďalej nasledujúce verejné funkcie:

```
public int Evaluate(IChromosome chromosome)
```

Parametrom je chromozóm jedinca, v našom prípade pole double hodnôt. Zo zadaných parametrov ohodnocuje daného jedinca. Táto metóda je volaná z triedy Population vo fáze evaluácie.

```
public List<ViewModelForSchedule> GetView(Operation
    op,ushort pIndex);
```

Ako parameter vstupuje druh operácie (pražička, mlyn...) a jej index (napr. pražička 5). Výstupom je naplánovaný rozvrh v danom technologickom uzle na práve ohodnotenom jedincovi z populácie vo forme zoznamu intervalov (prílohaC.3/s62). Takýto interval nesie informáciu o začiatku a konci úkonu, type vyhotovujúcej kávy, type úkonu (spracúvanie kávy, vyčkávanie na vyprázdnenie, voľná kapacita) a technologický uzol, na ktorom sa vykonávala predchádzajúca operácia.

```
public CScheduleView GetDownTime()
```

Funkcia má informačný charakter o plánovacích rozvrhoch plničiek. Navracia triedu (prílohaC.4/s63), ktorá zapúzdruje 4 informácie o aktuálne ohodnotenom jedincovi. Okrem ohodnotenia daného jedinca je to aj počet medzier v plánovacích rozpisoch, počet napražených kávových dávok vynásobený štyrmi, keďže jedna dávka sa praží na štyroch plničkách, a súčet voľných časových intervalov. Kompletný súbor funkcií a metód triedy TechSim je uvedený aj s popisom v (prílohaC.1/s57).

Inštancia triedy TechSim si počas vytvárania modelu načíta dáta o jednotlivých kávových zmesiach zo xml súboru “MyData“.

Pre správu xml súboru som vytvoril triedu CCoffeeData (prílohaC.2/s60). Trieda umožňuje načítanie a ukladanie dát z xml súboru. Pre získanie uloženej objednávky jednotlivých kávových zmesí slúži funkcia:

```
public List<ushort> GetOrder
```

Funkcia vráti zoznam objednávok, ktorý je možné vložiť priamo do konštruktoru triedy TechSim.

Trieda Population umožnuje okrem iného nastavovať parametre evolučného algoritmu. Na tento účel slúžia “public property“:

- CrossoverRate - hodnota definuje percentuálne množstvo jedincov, ktorí sa budú podieľať na krížení (0,4 predstavuje 40 percent).
- MutationRate - hodnota definuje percentuálne množstvo jedincov, ktorí sa budú podieľať na mutácií.

- RandomSelectionPortion - hodnota definuje percentuálne množstvo jedincov nanovo vygenerovaných v každom evolučnom cykle.
- a iné.

Pomocou nasledujúcich "public property" je možné nahliadať na aktuálny stav populácie:

- FitnessMax - funkcia navracia aktuálne uložený najlepší fitness
- BestChromosome - funkcia navracia aktuálne uloženého najlepšieho jedinca-/chromozóm
- a iné.

Konštruktor triedy Population s popisom jeho parametrov:

```
public Population(int size, IChromosome ancestor,
   IFitnessFunction fitnessFunction, ISelectionMethod
   selectionMethod)
```

- size - veľkosť vytváranej populácie
- ancestor - táto trieda reprezentuje jedinca v populácii, objekt dedí od interface IChromosome vlastnosti a metódy:
  - int Fitness - schopnosť vracať ohodnotenie daného jedinca
  - bool ChromosomeEqual(IChromosome pair) - schopnosť porovnať dva chromozómy
  - void Generate() - schopnosť vygenerovať náhodné hodnoty chromozómu
  - IChromosome CreateNew() - schopnosť vytvoriť nový chromozóm s rovnakými parametrami
  - IChromosome Clone() - schopnosť vytvoriť kópiu seba samého
  - void Mutate() - schopnosť definovaným spôsobom mutovať chromozóm
  - void Crossover(IChromosome pair) - schopnosť skrížiť dva chromozómy
  - void Evaluate(IFitnessFunction function) - schopnosť ohodnotiť seba samého pomocou fitness funkcie odovzdávanej ako parameter

  Tieto vlastnosti dedí aj použitá trieda DoubleArrayChromosome, ktorá bola použitá v plánovacom algoritme(prílohaC.6/s72). Jej metódy Mutate() a Crossover(IChromosome pair) som upravoval pre účel môjho plánovacieho algoritmu.
- fitnessFunkcion - objekt dedí od interface "IFitnessFunction" funkciu "int Evaluate(IChromosome chromosome)", o ktorej sa píše vyššie. Na túto pozíciu sa umiestni inštancia triedy TechSim.
- selectionMethod - objekt dedí od interface "ISelectionMethod" metódu "void ApplySelection(List<IChromosome> chromosomes, int size)", ktorá je schopná vykonať nad populáciou jedincov reprezentujúcich "chromosomes" veľkosti "size" selekciu.

Metóda pre spúšťanie evolučných cyklov v populácii:

```
public void RunEpoch()
```

Táto metóda spúšťa každý evolučný cyklus v postupnosti, ktorá je popísaná v podkapitole 3.7. Jednotlivé evolučné úkony je možné volať aj samostatne a viac krát za sebou. Túto možnosť poskytujú metódy:

```
public virtual void Crossover()
public virtual void Mutate()
public virtual void Selection()
```

Ďalšou z vybraných je metóda slúžiaca na pridávanie aj užívateľsky vytvorených chromozómov:

```
public void AddChromosome(IChromosome chromosome);
```

Kompletný súbor funkcií a metód triedy Population je uvedený aj s popisom v (prílohaC.5/s64). Každá verejná metóda/funkcia má zabuduvanú kontrolu vstupných parametrov a definované chovanie pri jej nesplnení. Popísaný súbor funkcii a metód plne zabezpečuje funkcionalitu plánovacej aplikácie a zobrazenie jej výsledkov.

5 ZÁVER

V mojej bakalárskej práci je rozpracovaná problematika výroby rôznych druhov kávových zmesí s cieľom optimalizovať plán ich výroby. V súčasnosti je každé percento ušetrených nákladov podstatné a optimalizácia je nekonečný a potrebný proces v každej výrobnej prevádzke. Kritériom optimalizácie bolo minimalizovanie dĺžky a počtu časových prestojov vyskytujúcich sa v plánovacích rozvrhoch plniacich zariadení na kávové zmesi. Sprievodným, ale tiež sledovaným efektom tohto plánovacieho procesu bolo naplánovanie, čo najväčšieho vyprodukovaného množstva kávových zmesí počas dvojtýždňového časového obdobia.

Zadanie mojej bakalárskej práce bolo splnené vo všetkých bodoch zadania:

V prvej časti bola odprezentovaná problematika plánovacích systémov, ich história, ako aj možnosti ich využívania. Na základe ich špecifických vlastnosti som vybral tri základné spôsoby plánovania. Konkrétne som opísal štruktúru a vlastnosti evolučných algoritmov 2.3.2, expertné systémy a ich základnú architektúru uvedených v podkapitile 2.3.3. V ďalšej časti uvádzam čisto deterministické plánovanie použiteľné pri malom počte možných plánovacích kombinácií.

Návrh vhodných algoritmov podľa zadaných kritérií opisujem od kapitoly 3. Ako prvú som rozpracoval problematiku plánovania výroby kávovej zmesi s následným podrobným popisom navrhovaného plánovacieho riešenia, ktoré pozostáva z modelu výrobnej technológie kávovej zmesi a Action listu (evolučné pravidlá) podľa, ktorých sa model technológií správa. Evolučné pravidlá som ďalej upravoval evolučným algoritmom štandardným postupom (kríženie, mutácia, selekcia).

Vypracoval som dvoj krokové testovanie dostupných výrobných kapacít na modele technológie, ktoré zamedzuje generovanie nerealizovateľných plánovacích rozvrhov.

Hlavne z dôvodu overovania funkčnosti týchto algoritmov a zobrazenia výsledkov som zostrojil WPF grafické rozhranie. Jeho podrobný popis aj s odporúčaným použitím uvádzam v podkapitole 3.9.

V kapitole 4 poskytujem náhľad aj s popisom na realizované verejné metódy a funkcie určené na opakovanie použitie v modele COMES Modeller. Ich úplný výpis uvádzam v (prílohaC.1/s57) a (prílohaC.5/s64). Tieto funkcie/metódy poskytujú plnú správu nad plánovacím algoritmom a zobrazením výsledkov plánovania.

Navrhujem dva spôsoby chovania sa modelu technológie pri hodnotení jedincov z evolučnej populácie.

- v prvom prípade je na základe evolučných pravidiel (definícia:3.1) rozhodované iba o následnosti plánovaných typov kávových zmesí a výrobná trasa ktorú daná dávka kávy absolvuje je vyberaná deterministickým pravidlom (definícia:3.2).

- v druhom prípade je následnosť plánovaných typov kávových zmesí aj výrobná trasa určovaná evolučnými pravidlami.

Výber z možných variánt sa uskutočnuje pri vytváraní modelu technológie. Z testovania algoritmov v podkapitole 3.8 vychádza ako lepšie prvé riešenie. Je to spôsobené diskrétnym chovaním fitness funkcie vo forme modelu technológie. K jej "vyhladeniu" by pomohlo napríklad zavedenie váhového systému v jednotlivých TU, ktorý by zohľadňoval ich možnosti vzhľadom na množstvo prepojení s ostatnými TU a zohľadňoval zdielané TU. Ďalšie odporúčania sa uvádzam v podkapitole 3.10.

Hlavnou výhodou variabilnosti algoritmov je možnosť optimalizovať nielen plánovacie rozvrhy výroby kávových zmesí, ale aj technológiu samotnú, pridávaním alebo odoberaním TU. Táto vlastnosť mnou navrhnutého riešenia výpočtového algoritmu umožňuje nachádzať aj iné riešenia ako bola modelovaná technológia.

LITERATÚRA

[1] *COMPAS automatizace - průmyslová automatizace a výrobní informační systémy MES* [online]. [cit. 23. 11. 2013]. Dostupné z URL: <http://www.compas.cz/>.

[2] COMPAS. *Podniková dokumentácia.* 2013. verze 3.

[3] HANÁČEK, Tomáš. FIKAR, Jaroslav. *Nové metody výrobního plánování* [online]. 2005, poslední aktualizácia 31. 5. 2005 [cit. 23. 11. 2013]. Dostupné z URL: <http://si.vse.cz/archive/proceedings/2005/nove-metody-vyrobniho-planovani.pdf>.

[4] KLOBASA, František. *Systémy plánování a řízení výroby.* [online]. [cit. 23. 11. 2013]. Dostupné z URL: <http://si.vse.cz/archive/proceedings/2005/nove-metody-vyrobniho-planovani.pdf>.

[5] MACH, Marián. *Evolučné algoritmy: prvky a princípy.* Letná 9, Košice, Slovensko: elfa s.r.o., 2009. ISBN 978-80-8086-123-0.

[6] HYNEK, Jozef. *Genetické algoritmy a genetické programování.* Husova ulice 1881,Havlíčkův Brod: Tiskárny Havlíčkův brod, a.s., 2008. ISBN 978-80-247-2695-3.

[7] *Hybrid evolutionary algorithms.* Editor Crina Groşan, Ajith Abraham, Hisao Ishibuchi. Berlin: Springer, c2007, xv, 403 s. ISBN 978-3-540-73296-9.

[8] ZELINKA, Ivan. *Umělá inteligence v problémech globální optimalizace.* 1. vyd. Praha: BEN - technická literatura, 2002, 189 s. ISBN 80-730-0069-5.

[9] MAŘIK, V. et al. *Umělá inteligence 2.* Štefánikova 2, 737 36 Český Těšín: Těšínská tiskárna, a.s., dotisk 2003. 5476. ISBN 80-200-0504-8.

[10] POPPER, Mikuláš. *Expertné systémy* 1. vyd. Bratislava: Alfa, 1989, 358 s. ISBN 80-050-0051-0.

[11] DVOŘÁK, Jiří. *Expertní Systémy.* 2004 [online]. [cit. 23. 11. 2013]. Dostupné z URL: <http://www.uai.fme.vutbr.cz/~jdvorak/Opory/ExpertniSystemy.pdf>.

[12] MÖHRING, Rolf H. a Jan Karel LENSTRA *Algorithms for deterministic and stochastic scheduling.* Berlin, 2001. 145 s. Dizertačná práca. der Technischen Universität Berlin.

[13] BRUCKER, Peter. *Scheduling algorithms.* 5th ed. Berlin: Springer, c2007, xii, 371 s. ISBN 978-3-540-69515-8.

[14] BEREŽNÝ, Š. a D. KRAVECOVÁ. *Lineárne programovanie.* Košice: Katedra matiky a toeretickej informatiky Fakulta elektrotechniky a informatiky Technická univerzita v Košiciach, 2012. ISBN 978-80-553-0910-1.

[15] MICHALEWICZ, Zbigniew. *Genetic algorithms data structures=evolution programs.* 3rd rev. extended ed. Berlin: Springer, 1996, viii, ix, 387 s. ISBN 35-406-0676-9.

[16] MAŘIK, V. et al. *Umělá inteligence 3.* Praha: SARIFA, s.r.o., dotisk 2004, 328 s. ISBN 80-200-0472-6.

[17] *AForge.NET* [online]. [cit. 12. 5. 2014]. Dostupné z URL: <`http://www.aforgenet.com/framework/features/genetic_algorithms.html`>.

ZOZNAM SYMBOLOV, VELIČÍN A SKRATIEK

COMES CCI COMES Communication interface

MSIE Microsoft Internet Explorer

MES Manufacturing Execution System

TIA Totálne Integrovaná Automatizácia

HMI Human Machine Interface

ERP Enterprise Resource Planning

ASP Active Server Pages

MCS Manufacturing Control System

OPC Object linking and embedding for Process Control

UTC Universal Time Coordinated

MRP Material Requirements Planning

MPS Master Production Schedule

CRP Capacity Requirements Planning

DRP Distribution Requirements Planning

SCM Supply Chain Management

APS Advanced Planning Scheduling

GA Genetic Algorithms

ES Expert system

EČ Evolučné číslo

TU technologický uzol

MVVM Model-View-ViewModel

ZOZNAM PRÍLOH

A Tabuľky meraných hodnôt **54**

B Plánovacie kraft data **56**

C Zdrojové kódy **57**

A TABUĽKY MERANÝCH HODNÔT

Tab. A.1: Merané hodnoty na modeli s evolučnou trasou

Základné nastavenie: Selekčný mechanizmus-Elite Selection, 20evolučných cyklov, veľkosť populácie:20jedincov,

Doplňujúce informácie	Čas výpočtu [min:s]	Usporiadenie technológie	Počet upražených dávok kávy	Dĺžka voľného času	Počet medzier v pánovaní	Vysledok fitness funkcie
	0:11	3/3/1/10/10	71	1748	284	3604
+20 evolučných cyklov	0:12	3/3/1/10/10	72	1744	288	3712
	0:16	3/6/1/10/10	73	1769	292	3762
+20 evolučných cyklov	0:15	3/6/1/10/10	76	1738	304	4124
	0:36	3/6/1/20/10	127	287	80	14266
+20 evolučných cyklov	0:36	3/6/1/20/10	127	280	69	14335
	1:28	3/6/1/30/10	126	334	68	14112
+20 evolučných cyklov	1:28	3/6/1/30/10	127	300	66	14310
	1:20	3/9/1/20/10	129	430	90	14170
+20 evolučných cyklov	0:59	3/9/1/20/10	130	398	86	14374
	1:52	3/9/1/30/10	143	125	10	16860
	2:50	3/12/1/20/10	143	134	16	16812
+20 evolučných cyklov	2:43	3/12/1/20/10	132	374	94	14622
Každá plnička má pripojené/ých X odplyňovacích síl , A zdieľa s ľavým susedom, B zdieľa s pravým susedom						
Každá pražička má pripojené/ých Y zrecích síl , B zdieľa s ľavým susedom, D zdieľa s pravým susedom						
X=3,A=1,B=1	0:38	3/6/1/20/10	132	374	94	14622
+20 evolučných cyklov	0:46	3/6/1/20/10	133	379	90	14752
X=4,A=2,B=2	0:51	3/6/1/20/10	132	415	107	14475
+20 evolučných cyklov	0:40	3/6/1/20/10	134	361	85	14933
Y=3,C=1,D=1	0:50	3/6/1/20/10	140	211	54	16108
Y=4,C=2,D=2	0:44	3/6/1/20/10	139	207	44	16046
Y=6,C=6,D=6	1:07	3/6/1/20/10	141	181	29	16413
+20 evolučných cyklov	1:08	3/6/1/20/10	141	176	26	16438
Y=3,C=1,D=1,X=4,A=2,B=2	1:00	3/6/1/20/10	133	377	74	14836
Y=10,C=5,D=5,X=2,A=0,B=0	2:00	3/20/2/80/40	144	567	41	15941
1evolučný cyklus, 640objednávok	0:23	3/20/2/80/40	569	581	48	66878

Tab. A.2: Merané hodnoty na modeli s deterministickou trasou

Číslo merania	Doplňujúce informácie	Čas výpočtu [min:s]	Usporiadenie technológie	Počet upražených dávok kávy	Dĺžka voľného času	Počet medzier v pánovaní	Vysledok fitness funkcie
1		0:43	3/6/1/20/10	138	252	74	15686
2		0:35	3/6/1/20/10	139	220	60	15940
3		0:36	3/6/1/20/10	137	231	52	15718
4		0:36	3/9/1/20/10	144	139	18	16912
5		0:44	3/6/1/30/10	145	132	16	17056
6		0:47	3/12/1/20/10	144	123	10	16984
7	10evolučných cyklov	0:11	3/3/1/20/10	107	880	222	9970
8	10evolučných cyklov	0:16	3/6/1/20/10	120	250	30	13750
9	10evolučných cyklov	0:23	3/9/1/20/10	143	200	20	16660
10	10evolučných cyklov	0:19	3/12/1/20/10	145	122	10	17106
11	10evolučných cyklov	0:29	3/15/1/20/10	145	122	10	17106
12	10evolučných cyklov	0:42	3/18/1/20/10	145	122	10	17106
13	10evolučných cyklov	1:03	3/21/1/20/10	145	122	10	17106
14	10evolučných cyklov	1:16	3/24/1/20/10	145	122	10	17106
15	10evolučných cyklov	1:10	3/27/1/20/10	145	122	10	17106
16	10evolučných cyklov	1:33	3/30/1/20/10	145	122	10	17106
17	10evolučných cyklov	2:14	3/36/1/20/10	145	122	10	17106
18	10evolučných cyklov	2:29	3/42/1/20/10	145	122	10	17106
19	**Každá plnička má pripojené/ých X odplyňovacích síl , A zdieľa s ľavým susedom, B zdieľa s pravým susedom**						
20	**Každá pražička má pripojené/ých Y zrecích síl , B zdieľa s ľavým susedom, D zdieľa s pravým susedom**						
21	X=3,A=1,B=1	0:51	3/6/1/20/10	134	366	91	14893
22	X=3,A=1,B=1	0:46	3/6/1/20/10	133	364	91	14777
23	X=3,A=1,B=1	0:47	3/6/1/20/10	137	322	87	15361
24	X=4,A=2,B=2	0:47	3/6/1/20/10	135	333	71	15179
25	X=4,A=2,B=2	0:50	3/6/1/20/10	135	276	61	15343
26	X=4,A=2,B=2	0:51	3/6/1/20/10	134	355	80	14970
27	X=5,A=3,B=3	1:03	3/6/1/20/10	136	313	76	15314
28	X=5,A=3,B=3	1:04	3/6/1/20/10	135	307	80	15186
29	X=5,A=3,B=3	1:02	3/6/1/20/10	133	372	106	14686
30	X=6,A=4,B=4	1:05	3/6/1/20/10	134	315	87	15015
31	X=6,A=4,B=4	0:59	3/6/1/20/10	136	291	75	15363
32	X=6,A=4,B=4	0:59	3/6/1/20/10	133	317	91	14871
33	X=7,A=5,B=5	0:53	3/6/1/20/10	134	342	78	15006
34	X=7,A=5,B=5	0:53	3/6/1/20/10	136	292	75	15361
35		0:55	3/6/1/20/10	134	314	71	15097
36	Y=3,C=1,D=1	0:48	3/6/1/20/10	141	180	38	16370
37	Y=4,C=2,D=2	0:58	3/6/1/20/10	143	146	26	16738
38		0:23	3/3/1/20/10	107	934	236	9792
39	Y=3,C=3,D=3	0:46	3/3/1/20/10	118	703	195	11779
40	selection= rank	0:35	3/6/1/20/10	136	319	84	15262
41	selection= rouletteWheel	0:35	3/6/1/20/10	130	386	105	14303
42	populácia= 40jedincov	1:21	3/6/1/20/10	137	271	61	15593
43	Y=9,C=9,D=9	1:25	3/9/1/20/10	145	122	10	17106
44	Y=8,C=8,D=8	1:20	3/8/1/20/10	145	122	10	17106
45	Y=7,C=7,D=7	0:56	3/7/1/20/10	144	122	10	16986
46	Y=6,C=6,D=6	1:08	3/6/1/20/10	143	124	10	16862
47	Y=5,C=5,D=5	0:58	3/5/1/20/10	141	162	30	16446
48	Y=4,C=4,D=4	0:57	3/4/1/20/10	131	358	118	14414
49	Y=3,C=3,D=3	0:26	3/3/1/20/10	107	904	219	9937
50	Y=2,C=2,D=2	0:29	3/2/1/20/10	84	1106	231	6713
51	X=21,A=21,B=21	2:00	3/3/1/21/10	140	238	49	16079
52	X=21,A=21,B=21	2:02	3/3/1/21/10	140	222	41	16151
53	X=21,A=21,B=21	2:22	3/3/1/21/10	138	235	85	15665
54	X=19,A=19,B=19	1:56	3/3/1/19/10	133	316	76	14948
55	X=17,A=17,B=17	1:57	3/3/1/17/10	129	468	153	13779
56	X=15,A=15,B=15	1:33	3/6/1/15/10	119	643	23	12879
57	X=13,A=13,B=13	1:18	3/6/1/13/10	107	950	289	9495
58	X=11,A=11,B=11	1:07	3/6/1/11/10	93	1159	306	7312

B PLÁNOVACIE KRAFT DATA

Tab. B.1: Plánovacie kraft data

ID číslo	Popis	Min. čas odplynenia ROV [h]	Max. čas odplynenia ROV [h]	Min. čas zrenia [h]	Max. čas zrenia [h]	Na Pražičke 1	Na Pražičke 2	Na Pražičke 3
12606	Krönung HY	4	1200	8	30	áno	áno	nie
12608	Royal Diplomat	4	1200	8	30	áno	áno	nie
12610	Wundermild	4	1200	8	30	áno	nie	ano
12614	HAG Coffeemat HY	4	1200	8	30	áno	áno	áno
12616	Milea	4	1200	8	30	áno	áno	áno
12622	Kenco Continental	4	1200	8	30	áno	nie	nie
12634	Kenco decaf.	4	1200	8	30	áno	nie	nie
12644	Bankett airline hrubší mletí	4	1200	8	30	nie	áno	áno
12650	Jacobs Optigrind	4	1200	8	30	nie	nie	áno
12654	Hag filter pouch	4	1200	8	30	nie	nie	áno
12664	Kenco Prime Time	4	1200	8	30	nie	nie	áno
12768	A	4	1200	8	30	nie	nie	áno
12770	decaf C pro 191	4	1200	8	30	nie	nie	áno
12772	košík B + C1 pro 191	4	1200	8	30	nie	áno	áno
12774	Košík decaf B	4	1200	8	30	áno	nie	áno
12776	košík decaf C	4	1200	8	30	áno	nie	nie
12778	košík75B+25C1	4	1200	8	30	nie	áno	áno
12780	Košík H pro 190 a 191	4	1200	8	30	áno	nie	áno
12782	Košík B - pro 301a 323	4	1200	8	30	áno	nie	nie
12784	košík C1 - pro 301	4	1200	8	30	áno	nie	áno
12786	Balance HY - složka s kofeinem	4	1200	8	30	nie	nie	áno
12788	Balance HY - složka bez kofeinu	4	1200	8	30	áno	nie	áno
12790	Košík G - pro 301	4	1200	8	30	nie	nie	áno
12792	Košík C1 - pro 323	4	1200	8	30	áno	nie	nie
12794	Košík H - pro 301	4	1200	8	30	áno	nie	nie

C ZDROJOVÉ KÓDY

Listing C.1: Trieda TechSim

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using GeneticAlgorithm;
using TechnologySimulation.ViewHelpers;
using TechnologySimulation.Coffee;
using System.Collections.ObjectModel;


namespace TechnologySimulation.Technology
{

    public class TechSim:IFitnessFunction
    {
        #region fields
        Stock Stock_;
        List<TechElement> MRoaster;
        List<TechElement> MSiloRest;
        List<TechElement> MMill;
        List<TechElement> MSiloDegass;
        List<TechElement> MFillMach;

        private ushort maxCycle;
        public ushort MaxCycle
        { get { return maxCycle; } }
        private List<ushort> OrderList;
        private int orderListCount;
        public int OrderListCount
        { get { return orderListCount; } }
        public int NumOfRoasters()
        {
            return MRoaster.Count();
        }
        public int NumOfSiloRest()
        {
            return MSiloRest.Count();
        }
        public int NumOfMill()
        {
            return MMill.Count();
        }
        public int NumOfSiloDegass()
        {
            return MSiloDegass.Count();
        }
        public int NumOfFillMach()
        {
            return MFillMach.Count();
        }
        #endregion


        public TechSim()
        {
            ClearStatic();
            MRoaster=new List<TechElement>();
            MSiloRest=new List<TechElement>();
            MMill=new List<TechElement>();
            MSiloDegass=new List<TechElement>();
            MFillMach=new List<TechElement>();
            CreateOperation

        }


        /// <summary>
        /// Initializes a new instance of the <see cref="TechSim"/> class.
        /// </summary>
        /// <param name="pOrder"></param> List of numbers of ordered batches of coffee. For example: List
             {2,5} represets 2 orders of 1.coffee  and 5 orders of 2.coffee.
        /// Order of coffee depends on how thez are stored in XLM file MyData.
        /// <param name="pNumOfRoast"></param> Number of roasters in technology must be >0 and <=1000
```

```
/// <param name="pNumOfSiloRest"></param> Number of silos in technology must be >0 and <=1000
/// <param name="pNumOfMill"></param> Number of mills in technology must be >0 and <=1000
/// <param name="pNumOfSiloDegass"></param> Number of slilos in technology must be >0 and <=1000
/// <param name="pNumOfFillMach"></param> Number of roasters in technology must be >0 and <=1000
/// <param name="pSelect"></param> Represents connections between machines/silos. The length of array
     must be equal to number of possible connections between machines
/// <param name="pTracingIndex"></param> Determines the strategy for scheduling the "road of
     production". 0 for EvolutionTracing other for DeterministicTracing
public TechSim(List<ushort> pOrder,int pNumOfRoast, int pNumOfSiloRest,int pNumOfMill,int
     pNumOfSiloDegass,int pNumOfFillMach, bool[] pSelect,ushort pTracingIndex)
{
    if ((pNumOfRoast <= 0) || (pNumOfRoast > 1000))
        throw new ArgumentOutOfRangeException("pNumOfRoast", "Number of roasters must be >0 and
             <=1000");
    if ((pNumOfSiloRest <= 0) || (pNumOfSiloRest > 1000))
        throw new ArgumentOutOfRangeException("pNumOfSiloRest", "Number of slios must be >0 and
             <=1000");
    if ((pNumOfMill <= 0) || (pNumOfMill > 1000))
        throw new ArgumentOutOfRangeException("pNumOfMill", "Number of mills must be >0 and <=1000");
    if ((pNumOfSiloDegass <= 0) || (pNumOfSiloDegass > 1000))
        throw new ArgumentOutOfRangeException("pNumOfRoast", "Number of silos must by >0 and <=1000");
    if ((pNumOfFillMach <= 0) || (pNumOfFillMach > 1000))
        throw new ArgumentOutOfRangeException("pNumOfRoast", "Number of fillers must by >0 and
             <=1000");
    if (pSelect.Length!= pNumOfRoast * pNumOfSiloRest + pNumOfSiloRest * pNumOfMill + pNumOfMill *
         pNumOfSiloDegass + pNumOfSiloDegass * pNumOfFillMach)
        throw new ArgumentException("The length of array must be equal to number of possible
             connections between machines", "pSelect");


    ClearStatic();
    MRoaster = new List<TechElement>();
    MSiloRest = new List<TechElement>();
    MMill = new List<TechElement>();
    MSiloDegass = new List<TechElement>();
    MFillMach = new List<TechElement>();
    SetStatus(pTracingIndex);
    CreateOperation(pOrder, pNumOfRoast, pNumOfSiloRest, pNumOfMill, pNumOfSiloDegass, pNumOfFillMach,
         pSelect);
}
/// <summary>
/// Evaluate function/ fitness function
/// </summary>
/// <param name="chromosome"></param> chromosome to evaluate
/// <returns>Returns fitness</returns>
public int Evaluate(IChromosome chromosome)
{
    ChangeChrom((chromosome as DoubleArrayChromosome));
    StartScheduling();
    orderListCount = Stock_.OrderListCount;
    return GetDownTime().DownTime;
}
/// <summary>
/// Schedule view on specific machine/silo
/// </summary>
/// <param name="op"></param>define type of operation
/// <param name="pIndex"></param> define index
/// <returns>Returns basic info about schedule on specific machine/silo</returns>
public List<ViewModelForSchedule> GetView(Operation op,ushort pIndex)
{
    TechElement element;
    switch(op){
        case Operation.Roaster:
            element = MRoaster[pIndex];
            break;
        case Operation.SiloRest:
            element = MSiloRest[pIndex];
            break;
        case Operation.Mill:
            element = MMill[pIndex];
            break;
        case Operation.SiloDegass:
            element = MSiloDegass[pIndex];
            break;
        case Operation.FillMach:
            element = MFillMach[pIndex];
            break;
        default:
            element = MFillMach[pIndex];
```

```
            break;
    }
    return element.GetView();

}
/// <summary>
/// Info about last calculated schedule
/// </summary>
/// <remarks><para>Function returns <see cref="CScheduleView">CScheduleView</see> class
/// witch encapsulate 4 informations about last calculated schedule:
/// DownTime - represents fitness value
/// NumOfReadyGaps - number of gaps in schedule
/// NumOfCoffeeFilled - number of filled coffee batches*4 (one batch is filled on 4 fillers)
/// ReadyTime - sumary length of gaps in schedule</para>
///
public CScheduleView GetDownTime()
{
    CScheduleView view=new CScheduleView();
    foreach(TechElement element in MFillMach)
    {
        view =view+ element.GetDownTime();
    }
    return view;
}

#region privateMethods
private void SetStatus(ushort index)
{
    switch (index)
    {
        case 0: status = _Status.EvolutionTracing;
            break;
        case 1: status = _Status.DeterministicTracing;
            break;
        default: status = _Status.DeterministicTracing;
            break;
    }
}
private _Status status = _Status.DeterministicTracing;
private void StartScheduling()
{
    TechElement.ActualCycle = 0;

    Stock_.StartScheduling(status);
}
private void CreateOperation(List<ushort> pOrderList,int pNumOfRoast, int pNumOfSiloRest,int
     pNumOfMill,int pNumOfSiloDegass,int pNumOfFillMach, bool[] pSelect)
{
    OrderList = new List<ushort>();
    ushort max;
    try
    {max = (ushort)pOrderList.Count(); }
    catch(OverflowException)
    { max = 65535; }

        for (ushort m = 1; m <= max; m++)
        {
            for (ushort i = 0; i < pOrderList[m-1]; i++)
            {
                OrderList.Add(m);
            }
        }

        maxCycle = (ushort)(OrderList.Count);
        Stock_ = new Stock(OrderList);

    int counter = 0;
    for(int i=0;i<pNumOfRoast;i++)
    {
        new Roaster(Stock_);
    }
    MRoaster = Roaster.GetListStatic;
    for(int i=0;i<pNumOfSiloRest;i++)
    {
        new SiloRest(MRoaster,pSelect.Skip(counter).Take(MRoaster.Count).ToArray());
        counter+=MRoaster.Count;
    }
    MSiloRest = SiloRest.GetListStatic;
    for(int i=0;i<pNumOfMill;i++)
```

```
            {
                new Mill(MSiloRest,pSelect.Skip(counter).Take(MSiloRest.Count).ToArray());
                counter+=MSiloRest.Count;
            }
            MMill = Mill.GetListStatic;
            for(int i=0;i<pNumOfSiloDegass;i++)
            {
                new SiloDegass(MMill,pSelect.Skip(counter).Take(MMill.Count).ToArray());
                counter+=MMill.Count;
            }
            MSiloDegass = SiloDegass.GetListStatic;

            for(int i=0;i<pNumOfFillMach;i++)
            {
                new FillMach(MSiloDegass,pSelect.Skip(counter).Take(MSiloDegass.Count).ToArray());
                counter+=MSiloDegass.Count;
            }
            MFillMach = FillMach.GetListStatic;
        }

        private void ChangeChrom(DoubleArrayChromosome chromosome)
        {
            int skip=0;
            int step=TechElement.MaxCycle;
            int MillConst = 4;
            foreach(TechElement element in TechElement.GetListStatic)
            {
                if (element is Mill)
                {
                    element.ChangeChrom(chromosome.Value.Skip(skip).Take(step * MillConst).ToArray());
                    skip += step*MillConst;
                }
                else
                {
                    element.ChangeChrom(chromosome.Value.Skip(skip).Take(step).ToArray());
                    skip += step;
                }
                element.ClearSchedule();

            }
            Stock_.ChangeChromCoffee(chromosome.Value.Skip(skip).Take(step).ToArray());

        }

        private static void ClearStatic()
        {
            TechElement.ClearStatic();
            Roaster.ClearStatic2();
            SiloRest.ClearStatic2();
            Mill.ClearStatic2();
            SiloDegass.ClearStatic2();
            FillMach.ClearStatic2();
        }
        #endregion

    }
}
```

Listing C.2: CCoffeeData

```
using System;
using System.Diagnostics;
using System.IO;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Xml;
using System.Xml.Linq;
using System.Xml.Serialization;
using System.Collections.ObjectModel;

namespace TechnologySimulation.Coffee
{
    public class CCoffeeData : Dictionary<ushort, SCoffee>
    {
        /// <summary>
        /// Initializes a new instance of the <see cref="CCoffeeData"/> class.
        /// </summary>
```

```
/// <remarks>Constructor also loads data from XML file MyData</remarks>
public CCoffeeData()
{
    List<SCoffee> datcoffee=this.LoadList();
    ushort key=1;
    foreach(SCoffee coffee in datcoffee)
    {
        this.Add(key++, coffee);
    }
}
/// <summary>
/// Load data function
/// </summary>
/// <returns>Returns ObservableCollection of SCoffee stored in XML file MyData</returns>
public ObservableCollection<SCoffee> Load()
{
    return new ObservableCollection<SCoffee>(this.LoadList());
}
/// <summary>
/// Orders for scheduling algorithm
/// </summary>
/// <returns>Returns List of count of orders of each kind of coffee</returns>
public List<ushort> GetOrder
{
    get { return new List<ushort>(this.Values.Select(k => k.Order).ToList()); }
}
/// <summary>
/// Method saves items to XML file MyData
/// </summary>
/// <param name="items"></param> items to be saved
public void Save(ObservableCollection<SCoffee> items)
{

    XDocument xdoc = new XDocument();
    XElement xeRoot = new XElement("Data");

    foreach (var item in items)
    {
        SCoffee lvc = (SCoffee)item;

        XElement xRow = new XElement("Row");
        xRow.Add(new XElement("ID", lvc.ID));
        xRow.Add(new XElement("Des", lvc.Description));
        xRow.Add(new XElement("MaxTimeDegass", lvc.MaxTimeDegass));
        xRow.Add(new XElement("MinTimeDegass", lvc.MinTimeDegass));
        xRow.Add(new XElement("MaxTimeRest", lvc.MaxTimeRest));
        xRow.Add(new XElement("MinTimeRest", lvc.MinTimeRest));
        xRow.Add(new XElement("Order", lvc.Order));
        foreach(var roaster in lvc.AllowedOnRoaster)
        {
            xRow.Add(new XElement("Allowed", roaster));
        }
        xeRoot.Add(xRow);
    }
    xdoc.Add(xeRoot);
    try
    {
        using (XmlTextWriter xw = new XmlTextWriter("MyData.xml", Encoding.UTF8))
        {
            xdoc.WriteTo(xw);
        }
    }
    catch (System.InvalidOperationException ex)
    {
        Debug.WriteLine("{0}", ex);
    }
    catch (System.Xml.XmlException ex)
    {
        Debug.WriteLine("{0}", ex);
    }
    catch (Exception ex)
    {
        Debug.WriteLine("General exception: {0}", ex);
    }
}

private List<SCoffee> LoadList()
{
    List<SCoffee> rows = new List<SCoffee>();
```

```
            if (File.Exists("MyData.xml"))
            {
                // Create the query
                try
                {
                    var rowsFromFile = from c in XDocument.Load(
                                "MyData.xml").Elements(
                                "Data").Elements("Row")
                                       select new { Elem=c, Allowed=c.Descendants("Allowed")

                                       };
                    // Execute the query
                    foreach (var row in rowsFromFile)
                    {
                        List<bool> allowed = new List<bool>();
                         foreach(var roaster in row.Allowed)
                         {
                             allowed.Add(Convert.ToBoolean(roaster.Value));
                         }

                        rows.Add(new SCoffee(Convert.ToUInt32(row.Elem.Element("ID").Value),
                                row.Elem.Element("Des").Value,
                                Convert.ToUInt16(row.Elem.Element("MinTimeDegass").Value),
                                Convert.ToUInt16(row.Elem.Element("MaxTimeDegass").Value),
                                Convert.ToUInt16(row.Elem.Element("MinTimeRest").Value),
                                Convert.ToUInt16(row.Elem.Element("MaxTimeRest").Value),
                                Convert.ToUInt16(row.Elem.Element("Order").Value),
                                allowed));
                    }
                }
                catch (NullReferenceException ex)
                {
                    Debug.WriteLine("{0}", ex);
                }
                catch (System.Xml.XmlException ex)
                {
                    Debug.WriteLine("{0}", ex);
                }
                catch (Exception ex)
                {
                    Debug.WriteLine("General exception: {0}", ex);
                }
            }
            return rows;
        }
    }
}
```

Listing C.3: Trieda ViewModelForSchedule

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace TechnologySimulation.ViewHelpers
{

    public enum Operation { Roaster, SiloRest, Mill, SiloDegass,FillMach }
    public class ViewModelForSchedule
    {
        private string coffeName;
        private string startTime;
        private string endTime;
        private string source;
        private string type;

        public string CoffeeName { get{return coffeName;}  }
        public string StartTime { get { return startTime; } }
        public string EndTime { get { return endTime; } }
        public string Source { get{return source;} }
        public string Type { get { return type; } }

        public ViewModelForSchedule(string pCoffeeName,ushort pStartTime,ushort pEndTime,string pSource,string
             pType)
        {
```

```
            coffeName = pCoffeeName;
            DateTime date=DateTime.Today;
            date += TimeSpan.FromHours(pStartTime);
            startTime = date.ToString("ddd HH':'mm", CultureInfo.CurrentUICulture);
            date = DateTime.Today;
            date += TimeSpan.FromHours(pEndTime);
            endTime = date.ToString("ddd HH':'mm", CultureInfo.CurrentUICulture);
            source = pSource;
            type = pType;
        }
    }
}
```

Listing C.4: Trieda SScheduleView

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace TechnologySimulation.ViewHelpers
{
    public class CScheduleView
    {
        public int DownTime { get; set; }
        public int NumOfReadyGaps { get; set; }
        public int NumOfCoffeeFilled { get; set; }
        public int ReadyTime { get; set; }

        public CScheduleView()
        {
            this.DownTime=0;
            this.NumOfReadyGaps = 0;
            this.NumOfCoffeeFilled = 0;
        }

        public CScheduleView(int p1, int p2, int p3,int pReadyTime)
        {
            this.DownTime = p1;
            this.NumOfCoffeeFilled = p2;
            this.NumOfReadyGaps = p3;
            this.ReadyTime = pReadyTime;
        }

        public static CScheduleView operator +(CScheduleView s1, CScheduleView s2)
        {
            return new CScheduleView(s1.DownTime + s2.DownTime,
                s1.NumOfCoffeeFilled + s2.NumOfCoffeeFilled,
                s1.NumOfReadyGaps + s2.NumOfReadyGaps,
                s1.ReadyTime+s2.ReadyTime);
        }
    }
}
```

Listing C.5: Trieda Population

```
namespace GeneticAlgorithm
{
    using System;
    using System.Collections;
    using System.Collections.Generic;

    /// <summary>
    /// Population of chromosomes.
    /// </summary>
    ///
    /// <remarks><para>The class represents population - collection of individuals (chromosomes)
    /// and provides functionality for common population's life cycle - population growing
    /// with help of genetic operators and selection of chromosomes to new generation
    /// with help of selection algorithm. The class may work with any type of chromosomes
    /// implementing <see cref="IChromosome"/> interface, use any type of fitness functions
    /// implementing <see cref="IFitnessFunction"/> interface and use any type of selection
    /// algorithms implementing <see cref="ISelectionMethod"/> interface.</para>
    /// </remarks>
    ///
    public class Population
    {
        private IFitnessFunction fitnessFunction;
        private ISelectionMethod selectionMethod;
        private List<IChromosome> population = new List<IChromosome>();
        private int size;
        private double randomSelectionPortion = 0.0;
        private bool autoShuffling = false;

        // population parameters
        private double crossoverRate = 0.75;
        private double mutationRate = 0.10;

        // random number generator
        private static Random rand = new Random(5);

        //
        private double fitnessMax = 0;
        private double fitnessSum = 0;
        private double fitnessAvg = 0;
        private IChromosome bestChromosome = null;



        /// <summary>
        /// Crossover rate, [0.1, 1].
        /// </summary>
        ///
        /// <remarks><para>The value determines the amount of chromosomes which participate
        /// in crossover.</para>
        ///
        /// <para>Default value is set to <b>0.75</b>.</para>
        /// </remarks>
        ///
        public double CrossoverRate
        {
            get { return crossoverRate; }
            set
            {
                crossoverRate = Math.Max(0.1, Math.Min(1.0, value));
            }
        }

        /// <summary>
        /// Mutation rate, [0.1, 1].
        /// </summary>
        ///
        /// <remarks><para>The value determines the amount of chromosomes which participate
        /// in mutation.</para>
        ///
        /// <para>Defaul value is set to <b>0.1</b>.</para></remarks>
        ///
        public double MutationRate
        {
            get { return mutationRate; }
            set
            {
```

```
            mutationRate = Math.Max(0.1, Math.Min(1.0, value));
        }
    }

    /// <summary>
    /// Random selection portion, [0, 0.9].
    /// </summary>
    ///
    /// <remarks><para>The value determines the amount of chromosomes which will be
    /// randomly generated for the new population. The property controls the amount
    /// of chromosomes, which are selected to a new population using
    /// <see cref="SelectionMethod">selection operator</see>, and amount of random
    /// chromosomes added to the new population.</para>
    ///
    /// <para>Default value is set to <b>0</b>.</para></remarks>
    ///
    public double RandomSelectionPortion
    {
        get { return randomSelectionPortion; }
        set
        {
            randomSelectionPortion = Math.Max(0, Math.Min(0.9, value));
        }
    }

    /// <summary>
    /// Determines of auto shuffling is on or off.
    /// </summary>
    ///
    /// <remarks><para>The property specifies if automatic shuffling needs to be done
    /// on each <see cref="RunEpoch">epoch</see> by calling <see cref="Shuffle"/>
    /// method.</para>
    ///
    /// <para>Default value is set to <see langword="false"/>.</para></remarks>
    ///
    public bool AutoShuffling
    {
        get { return autoShuffling; }
        set { autoShuffling = value; }
    }

    /// <summary>
    /// Selection method to use with the population.
    /// </summary>
    ///
    /// <remarks><para>The property sets selection method which is used to select
    /// population members for a new population - filter population after reproduction
    /// was done with operators like crossover and mutations.</para></remarks>
    ///
    public ISelectionMethod SelectionMethod
    {
        get { return selectionMethod; }
        set { selectionMethod = value; }
    }

    /// <summary>
    /// Fitness function to apply to the population.
    /// </summary>
    ///
    /// <remarks><para>The property sets fitness function, which is used to evaluate
    /// usefulness of population's chromosomes. Setting new fitness function causes recalculation
    /// of fitness values for all population's members and new best member will be found.</para>
    /// </remarks>
    ///
    public IFitnessFunction FitnessFunction
    {
        get { return fitnessFunction; }
        set
        {
            fitnessFunction = value;

            foreach (IChromosome member in population)
            {
                member.Evaluate(fitnessFunction);
            }

            FindBestChromosome();
        }
    }
```

```
/// <summary>
/// Maximum fitness of the population.
/// </summary>
///
/// <remarks><para>The property keeps maximum fitness of chromosomes currently existing
/// in the population.</para>
///
/// <para><note>The property is recalculate only after <see cref="Selection">selection</see>
/// or <see cref="Migrate">migration</see> was done.</note></para>
/// </remarks>
///
public double FitnessMax
{
    get { return fitnessMax; }
}

/// <summary>
/// Summary fitness of the population.
/// </summary>
///
/// <remarks><para>The property keeps summary fitness of all chromosome existing in the
/// population.</para>
///
/// <para><note>The property is recalculate only after <see cref="Selection">selection</see>
/// or <see cref="Migrate">migration</see> was done.</note></para>
/// </remarks>
///
public double FitnessSum
{
    get { return fitnessSum; }
}

/// <summary>
/// Average fitness of the population.
/// </summary>
///
/// <remarks><para>The property keeps average fitness of all chromosome existing in the
/// population.</para>
///
/// <para><note>The property is recalculate only after <see cref="Selection">selection</see>
/// or <see cref="Migrate">migration</see> was done.</note></para>
/// </remarks>
///
public double FitnessAvg
{
    get { return fitnessAvg; }
}

/// <summary>
/// Best chromosome of the population.
/// </summary>
///
/// <remarks><para>The property keeps the best chromosome existing in the population
/// or <see langword="null"/> if all chromosomes have 0 fitness.</para>
///
/// <para><note>The property is recalculate only after <see cref="Selection">selection</see>
/// or <see cref="Migrate">migration</see> was done.</note></para>
/// </remarks>
///
public IChromosome BestChromosome
{
    get { return bestChromosome; }
}

/// <summary>
/// Size of the population.
/// </summary>
///
/// <remarks>The property keeps initial (minimal) size of population.
/// Population always returns to this size after selection operator was applied,
/// which happens after <see cref="Selection"/> or <see cref="RunEpoch"/> methods
/// call.</remarks>
///
public int Size
{
    get { return size; }
}
```

```
/// <summary>
/// Get chromosome with specified index.
/// </summary>
///
/// <param name="index">Chromosome's index to retrieve.</param>
///
/// <remarks>Allows to access individuals of the population.</remarks>
///
public IChromosome this[int index]
{
    get { return population[index]; }
}


/// <summary>
/// Initializes a new instance of the <see cref="Population"/> class.
/// </summary>
///
/// <param name="size">Initial size of population.</param>
/// <param name="ancestor">Ancestor chromosome to use for population creatioin.</param>
/// <param name="fitnessFunction">Fitness function to use for calculating
/// chromosome's fitness values.</param>
/// <param name="selectionMethod">Selection algorithm to use for selection
/// chromosome's to new generation.</param>
///
/// <remarks>Creates new population of specified size. The specified ancestor
/// becomes first member of the population and is used to create other members
/// with same parameters, which were used for ancestor's creation.</remarks>
///
/// <exception cref="ArgumentException">Too small population's size was specified. The
/// exception is thrown in the case if <paramref name="size"/> is smaller than 2.</exception>
///
public Population(int size,
                  IChromosome ancestor,
                  IFitnessFunction fitnessFunction,
                  ISelectionMethod selectionMethod)
{
    if (size < 2)
        throw new ArgumentException("Too small population's size was specified.");

    this.fitnessFunction = fitnessFunction;
    this.selectionMethod = selectionMethod;
    this.size = size;

    // add ancestor to the population
    ancestor.Evaluate(fitnessFunction);
    population.Add(ancestor.Clone());
    // add more chromosomes to the population
    for (int i = 1; i < size; i++)
    {
        // create new chromosome
        IChromosome c = ancestor.CreateNew();
        // calculate it's fitness
        c.Evaluate(fitnessFunction);
        // add it to population
        population.Add(c);
    }
}

/// <summary>
/// Regenerate population.
/// </summary>
///
/// <remarks>The method regenerates population filling it with random chromosomes.</remarks>
///
public void Regenerate()
{
    IChromosome ancestor = population[0];

    // clear population
    population.Clear();
    // add chromosomes to the population
    for (int i = 0; i < size; i++)
    {
        // create new chromosome
        IChromosome c = ancestor.CreateNew();
        // calculate it's fitness
        c.Evaluate(fitnessFunction);
        // add it to population
```

```
            population.Add(c);
        }
    }

    /// <summary>
    /// Do crossover in the population.
    /// </summary>
    ///
    /// <remarks>The method walks through the population and performs crossover operator
    /// taking each two chromosomes in the order of their presence in the population.
    /// The total amount of paired chromosomes is determined by
    /// <see cref="CrossoverRate">crossover rate</see>.</remarks>
    ///
    public virtual void Crossover()
    {
        // crossover
        for (int i = 1; i < size; i += 2)
        {
            // generate next random number and check if we need to do crossover
            if (rand.NextDouble() <= crossoverRate)
            {
                // clone both ancestors
                IChromosome c1 = population[i - 1].Clone();
                IChromosome c2 = population[i].Clone();

                // do crossover
                c1.Crossover(c2);

                // calculate fitness of these two offsprings
                c1.Evaluate(fitnessFunction);
                c2.Evaluate(fitnessFunction);

                // add two new offsprings to the population
                population.Add(c1);
                population.Add(c2);
            }
        }
    }

    /// <summary>
    /// Do mutation in the population.
    /// </summary>
    ///
    /// <remarks>The method walks through the population and performs mutation operator
    /// taking each chromosome one by one. The total amount of mutated chromosomes is
    /// determined by <see cref="MutationRate">mutation rate</see>.</remarks>
    ///
    public virtual void Mutate()
    {
        // mutate
        for (int i = 0; i < size; i++)
        {
            // generate next random number and check if we need to do mutation
            if (rand.NextDouble() <= mutationRate)
            {
                // clone the chromosome
                IChromosome c = population[i].Clone();
                // mutate it
                c.Mutate();
                // calculate fitness of the mutant
                c.Evaluate(fitnessFunction);
                // add mutant to the population
                population.Add(c);
            }
        }
    }

    /// <summary>
    /// Do selection.
    /// </summary>
    ///
    /// <remarks>The method applies selection operator to the current population. Using
    /// specified selection algorithm it selects members to the new generation from current
    /// generates and adds certain amount of random members, if is required
    /// (see <see cref="RandomSelectionPortion"/>).</remarks>
    ///
    public virtual void Selection()
    {
        // amount of random chromosomes in the new population
```

```
    int randomAmount = (int)(randomSelectionPortion * size);

    // do selection
    selectionMethod.ApplySelection(population, size - randomAmount);

    // add random chromosomes
    if (population.Count < size)
    {
        IChromosome ancestor = population[0];
        int populationCount = population.Count;
        for (int i = 0; i < size-populationCount; i++)
        {
            // create new chromosome
            IChromosome c = ancestor.CreateNew();
            // calculate it's fitness
            c.Evaluate(fitnessFunction);
            // add it to population
            population.Add(c);
        }
    }

    FindBestChromosome();
}

/// <summary>
/// Run one epoch of the population.
/// </summary>
///
/// <remarks>The method runs one epoch of the population, doing crossover, mutation
/// and selection by calling <see cref="Crossover"/>, <see cref="Mutate"/> and
/// <see cref="Selection"/>.</remarks>
///
public void RunEpoch()
{
    Crossover();
    Mutate();
    Selection();

    if (autoShuffling)
        Shuffle();
}

/// <summary>
/// Shuffle randomly current population.
/// </summary>
///
/// <remarks><para>Population shuffling may be useful in cases when selection
/// operator results in not random order of chromosomes (for example, after elite
/// selection population may be ordered in ascending/descending order).</para></remarks>
///
public void Shuffle()
{
    // current population size
    int size = population.Count;
    // create temporary copy of the population
    List<IChromosome> tempPopulation = population.GetRange(0, size);
    // clear current population and refill it randomly
    population.Clear();

    while (size > 0)
    {
        int i = rand.Next(size);

        population.Add(tempPopulation[i]);
        tempPopulation.RemoveAt(i);

        size--;
    }
}

/// <summary>
/// Add chromosome to the population.
/// </summary>
///
/// <param name="chromosome">Chromosome to add to the population.</param>
///
/// <remarks><para>The method adds specified chromosome to the current population.
/// Manual adding of chromosome maybe useful, when it is required to add some initialized
/// chromosomes instead of random.</para>
```

```
///
/// <para><note>Adding chromosome manually should be done very carefully, since it
/// may break the population. The manually added chromosome must have the same type
/// and initialization parameters as the ancestor passed to constructor.</note></para>
/// </remarks>
///
public void AddChromosome(IChromosome chromosome)
{
    chromosome.Evaluate(fitnessFunction);
    population.Add(chromosome);
}

/// <summary>
/// Perform migration between two populations.
/// </summary>
///
/// <param name="anotherPopulation">Population to do migration with.</param>
/// <param name="numberOfMigrants">Number of chromosomes from each population to migrate.</param>
/// <param name="migrantsSelector">Selection algorithm used to select chromosomes to migrate.</param>
///
/// <remarks><para>The method performs migration between two populations - current and the
/// <paramref name="anotherPopulation">specified one</paramref>. During migration
/// <paramref name="numberOfMigrants">specified number</paramref> of chromosomes is choosen from
/// each population using <paramref name="migrantsSelector">specified selection algorithms</paramref>
/// and put into another population replacing worst members there.</para></remarks>
///
public void Migrate(Population anotherPopulation, int numberOfMigrants, ISelectionMethod
     migrantsSelector)
{
    int currentSize = this.size;
    int anotherSize = anotherPopulation.Size;

    // create copy of current population
    List<IChromosome> currentCopy = new List<IChromosome>();

    for (int i = 0; i < currentSize; i++)
    {
        currentCopy.Add(population[i].Clone());
    }

    // create copy of another population
    List<IChromosome> anotherCopy = new List<IChromosome>();

    for (int i = 0; i < anotherSize; i++)
    {
        anotherCopy.Add(anotherPopulation.population[i].Clone());
    }

    // apply selection to both populations' copies - select members to migrate
    migrantsSelector.ApplySelection(currentCopy, numberOfMigrants);
    migrantsSelector.ApplySelection(anotherCopy, numberOfMigrants);

    // sort original populations, so the best chromosomes are in the beginning
    population.Sort();
    anotherPopulation.population.Sort();

    // remove worst chromosomes from both populations to free space for new members
    population.RemoveRange(currentSize - numberOfMigrants, numberOfMigrants);
    anotherPopulation.population.RemoveRange(anotherSize - numberOfMigrants, numberOfMigrants);

    // put migrants to corresponding populations
    population.AddRange(anotherCopy);
    anotherPopulation.population.AddRange(currentCopy);

    // find best chromosomes in each population
    FindBestChromosome();
    anotherPopulation.FindBestChromosome();
}

/// <summary>
/// Resize population to the new specified size.
/// </summary>
///
/// <param name="newPopulationSize">New size of population.</param>
///
/// <remarks><para>The method does resizing of population. In the case if population
/// should grow, it just adds missing number of random members. In the case if
/// population should get smaller, the <see cref="SelectionMethod">population's
/// selection method</see> is used to reduce the population.</para></remarks>
```

```
///
/// <exception cref="ArgumentException">Too small population's size was specified. The
/// exception is thrown in the case if <paramref name="newPopulationSize"/> is smaller than
     2.</exception>
///
public void Resize(int newPopulationSize)
{
    Resize(newPopulationSize, selectionMethod);
}

/// <summary>
/// Resize population to the new specified size.
/// </summary>
///
/// <param name="newPopulationSize">New size of population.</param>
/// <param name="membersSelector">Selection algorithm to use in the case
/// if population should get smaller.</param>
///
/// <remarks><para>The method does resizing of population. In the case if population
/// should grow, it just adds missing number of random members. In the case if
/// population should get smaller, the specified selection method is used to
/// reduce the population.</para></remarks>
///
/// <exception cref="ArgumentException">Too small population's size was specified. The
/// exception is thrown in the case if <paramref name="newPopulationSize"/> is smaller than
     2.</exception>
///
public void Resize(int newPopulationSize, ISelectionMethod membersSelector)
{
    if (newPopulationSize < 2)
        throw new ArgumentException("Too small new population's size was specified.");

    if (newPopulationSize > size)
    {
        // population is growing, so add new rundom members

        // Note: we use population.Count here instead of "size" because
        // population may be bigger already after crossover/mutation. So
        // we just keep those members instead of adding random member.
        int toAdd = newPopulationSize - population.Count;

        for (int i = 0; i < toAdd; i++)
        {
            // create new chromosome
            IChromosome c = population[0].CreateNew();
            // calculate it's fitness
            c.Evaluate(fitnessFunction);
            // add it to population
            population.Add(c);
        }
    }
    else
    {
        // do selection
        membersSelector.ApplySelection(population, newPopulationSize);
    }

    size = newPopulationSize;
}

// Find best chromosome in the population so far
private void FindBestChromosome()
{
    bestChromosome = population[0];
    fitnessMax = bestChromosome.Fitness;
    fitnessSum = fitnessMax;

    for (int i = 1; i < size; i++)
    {
        double fitness = population[i].Fitness;

        // accumulate summary value
        fitnessSum += fitness;

        // check for max
        if (fitness > fitnessMax)
        {
            fitnessMax = fitness;
            bestChromosome = population[i];
```

```
                }
            }
            fitnessAvg = fitnessSum / size;
        }
    }
}
```

Listing C.6: Trieda DoubleArrayChromosome

```
namespace GeneticAlgorithm
{
    using System;
    using System.Linq;
    using System.Text;



    /// <summary>
    /// Double array chromosome.
    /// </summary>
    ///
    /// <remarks><para>Double array chromosome represents array of double values.
    /// Array length is in the range of [2, 65536].
    /// </para>
    ///
    /// <para>See documentation to <see cref="Mutate"/> and <see cref="Crossover"/> methods
    /// for information regarding implemented mutation and crossover operators.</para>
    /// </remarks>
    ///
    public class DoubleArrayChromosome : ChromosomeBase
    {
        /// <summary>
        /// Chromosome generator.
        /// </summary>
        ///
        /// <remarks><para>This random number generator is used to initialize chromosome's genes,
        /// which is done by calling <see cref="Generate"/> method.</para></remarks>
        ///
        protected Random chromosomeGenerator;

        /// <summary>
        /// Mutation multiplier generator.
        /// </summary>
        ///
        /// <remarks><para>This random number generator is used to generate random multiplier values,
        /// which are used to multiply chromosome's genes during mutation.</para></remarks>
        ///
        protected Random mutationLengthGenerator;

        /// <summary>
        /// Mutation addition generator.
        /// </summary>
        ///
        /// <remarks><para>This random number generator is used to generate random addition values,
        /// which are used to add to chromosome's genes during mutation.</para></remarks>
        ///
        protected Random mutationAdditionGenerator;

        /// <summary>
        /// Random number generator for crossover and mutation points selection.
        /// </summary>
        ///
        /// <remarks><para>This random number generator is used to select crossover
        /// and mutation points.</para></remarks>
        ///
        //protected static ThreadSafeRandom rand = new ThreadSafeRandom();
        protected static Random rand = new Random(4);

        /// <summary>
        /// Chromosome's maximum length.
        /// </summary>
        ///
        /// <remarks><para>Maxim chromosome's length in array elements.</para></remarks>
        ///
        const int maxLength=2147483647;
        public static int MaxLength
        {
```

```
    get{return maxLength;}
}

/// <summary>
/// Chromosome's length in number of elements.
/// </summary>
private int length;

/// <summary>
/// Chromosome's value.
/// </summary>
protected double[] val = null;

// balancers to control type of mutation and crossover
private double mutationBalancer = 0.9;
private double crossoverBalancer = 0.5;


/// <summary>
/// Chromosome's length.
/// </summary>
///
/// <remarks><para>Length of the chromosome in array elements.</para></remarks>
///
public int Length
{
    get { return length; }
}

/// <summary>
/// Chromosome's value.
/// </summary>
///
/// <remarks><para>Current value of the chromosome.</para></remarks>
///
public double[] Value
{
    get { return val; }
}

/// <summary>
/// Mutation balancer to control mutation type, [0, 1].
/// </summary>
///
/// <remarks><para>The property controls type of mutation, which is used more
/// frequently. A radnom number is generated each time before doing mutation -
/// if the random number is smaller than the specified balance value, then one
/// mutation type is used, otherwse another. See <see cref="Mutate"/> method
/// for more information.</para>
///
/// <para>Default value is set to <b>0.5</b>.</para>
/// </remarks>
///
public double MutationBalancer
{
    get { return mutationBalancer; }
    set { mutationBalancer = Math.Max(0.0, Math.Min(1.0, value)); }
}

/// <summary>
/// Crossover balancer to control crossover type, [0, 1].
/// </summary>
///
/// <remarks><para>The property controls type of crossover, which is used more
/// frequently. A radnom number is generated each time before doing crossover -
/// if the random number is smaller than the specified balance value, then one
/// crossover type is used, otherwse another. See <see cref="Crossover"/> method
/// for more information.</para>
///
/// <para>Default value is set to <b>0.5</b>.</para>
/// </remarks>
///
public double CrossoverBalancer
{
    get { return crossoverBalancer; }
    set { crossoverBalancer = Math.Max(0.0, Math.Min(1.0, value)); }
}

private DoubleArrayChromosome()
```

```
{
    this.chromosomeGenerator = new Random(rand.Next());
    this.mutationLengthGenerator = new Random(rand.Next());
    this.mutationAdditionGenerator = new Random(rand.Next());
}
/// <summary>
/// Initializes a new instance of the <see cref="DoubleArrayChromosome"/> class.
/// </summary>
///
/// <param name="chromosomeGenerator">Chromosome generator - random number generator, which is
/// used to initialize chromosome's genes, which is done by calling <see cref="Generate"/> method
/// or in class constructor.</param>
/// <param name="mutationMultiplierGenerator">Mutation multiplier generator - random number
/// generator, which is used to generate random multiplier values, which are used to
/// multiply chromosome's genes during mutation.</param>
/// <param name="mutationAdditionGenerator">Mutation addition generator - random number
/// generator, which is used to generate random addition values, which are used to
/// add to chromosome's genes during mutation.</param>
/// <param name="length">Chromosome's length in array elements, [2, <see cref="MaxLength"/>].</param>
///
/// <remarks><para>The constructor initializes the new chromosome randomly by calling
/// <see cref="Generate"/> method.</para></remarks>
///
public DoubleArrayChromosome(int length):this()
{

    // save parameters

    this.length = Math.Max(2, Math.Min(MaxLength, length));
    // allocate array
    val = new double[length];


    // generate random chromosome
    Generate();
}

/// <summary>
/// Initializes a new instance of the <see cref="DoubleArrayChromosome"/> class.
/// </summary>
///
/// <param name="chromosomeGenerator">Chromosome generator - random number generator, which is
/// used to initialize chromosome's genes, which is done by calling <see cref="Generate"/> method
/// or in class constructor.</param>
/// <param name="mutationMultiplierGenerator">Mutation multiplier generator - random number
/// generator, which is used to generate random multiplier values, which are used to
/// multiply chromosome's genes during mutation.</param>
/// <param name="mutationAdditionGenerator">Mutation addition generator - random number
/// generator, which is used to generate random addition values, which are used to
/// add to chromosome's genes during mutation.</param>
/// <param name="values">Values used to initialize the chromosome.</param>
///
/// <remarks><para>The constructor initializes the new chromosome with specified <paramref
     name="values">values</paramref>.
/// </para></remarks>
///
/// <exception cref="ArgumentOutOfRangeException">Invalid length of values array.</exception>
///
public DoubleArrayChromosome( double[] values):this()
{
    if ((values.Length < 2) || (values.Length > MaxLength))
        throw new ArgumentOutOfRangeException("Invalid length of values array.");

    // save parameters


    this.length = values.Length;

    // copy specified values
    val = (double[])values.Clone();
}

/// <summary>
/// Initializes a new instance of the <see cref="DoubleArrayChromosome"/> class.
/// </summary>
///
/// <param name="source">Source chromosome to copy.</param>
///
/// <remarks><para>This is a copy constructor, which creates the exact copy
```

```
/// of specified chromosome.</para></remarks>
///
public DoubleArrayChromosome(DoubleArrayChromosome source)
{
    this.chromosomeGenerator = source.chromosomeGenerator;
    this.mutationLengthGenerator = source.mutationLengthGenerator;
    this.mutationAdditionGenerator = source.mutationAdditionGenerator;
    this.length = source.length;
    this.fitness = source.fitness;
    this.mutationBalancer = source.mutationBalancer;
    this.crossoverBalancer = source.crossoverBalancer;

    // copy genes
    val = (double[])source.val.Clone();
}

/// <summary>
/// Get string representation of the chromosome.
/// </summary>
///
/// <returns>Returns string representation of the chromosome.</returns>
///
public override string ToString()
{
    StringBuilder sb = new StringBuilder();

    // append first gene
    sb.Append(val[0]);
    // append all other genes
    for (int i = 1; i < length; i++)
    {
        sb.Append(' ');
        sb.Append(val[i]);
    }

    return sb.ToString();
}


public override bool ChromosomeEqual(IChromosome pair)
{
    DoubleArrayChromosome p = (DoubleArrayChromosome)pair;
    return Enumerable.SequenceEqual(p.Value, this.Value);
}
/// <summary>
/// Generate random chromosome value.
/// </summary>
///
/// <remarks><para>Regenerates chromosome's value using random number generator.</para>
/// </remarks>
///
public override void Generate()
{
    for (int i = 0; i < length; i++)
    {
        // generate next value
        val[i] = chromosomeGenerator.NextDouble();
    }
}

/// <summary>
/// Create new random chromosome with same parameters (factory method).
/// </summary>
///
/// <remarks><para>The method creates new chromosome of the same type, but randomly
/// initialized. The method is useful as factory method for those classes, which work
/// with chromosome's interface, but not with particular chromosome type.</para></remarks>
///
public override IChromosome CreateNew()
{
    return new DoubleArrayChromosome(length);
}

/// <summary>
/// Clone the chromosome.
/// </summary>
///
/// <returns>Return's clone of the chromosome.</returns>
///
```

```
/// <remarks><para>The method clones the chromosome returning the exact copy of it.</para>
/// </remarks>
///
public override IChromosome Clone()
{
    return new DoubleArrayChromosome(this);
}

/// <summary>
/// Mutation operator.
/// </summary>
///
/// <remarks><para>The method performs chromosome's mutation, adding random number
/// to chromosome's gene or multiplying the gene by random number. These random
/// numbers are generated with help of <see cref="mutationMultiplierGenerator">mutation
/// multiplier</see> and <see cref="mutationAdditionGenerator">mutation
/// addition</see> generators.</para>
///
/// <para>The exact type of mutation applied to the particular gene
/// is selected randomly each time and depends on <see cref="MutationBalancer"/>.
/// Before mutation is done a random number is generated in [0, 1] range - if the
/// random number is smaller than <see cref="MutationBalancer"/>, then
/// mutation is done.
/// </para></remarks>
///
public override void Mutate()
{
    double factor = rand.NextDouble();
    if (rand.Next(2) == 0)
        factor = -factor;

    for (int i = 0; i < (mutationLengthGenerator.Next(length)); i++)
    {
        int mutationGene = rand.Next(length);
        double portion = mutationAdditionGenerator.NextDouble()/2*factor;

        if (rand.NextDouble() < mutationBalancer)
        {
            val[mutationGene] = ((val[mutationGene] + portion < 1) && (val[mutationGene] + portion >
                  0)) ? val[mutationGene] + portion :
               (((val[mutationGene] - portion > 0) && (val[mutationGene] - portion < 1)) ?
                    val[mutationGene] - portion : val[mutationGene]);
        }

    }
}

/// <summary>
/// Crossover operator.
/// </summary>
///
/// <param name="pair">Pair chromosome to crossover with.</param>
///
/// <remarks><para>The method performs crossover between two chromosomes, selecting
/// randomly the exact type of crossover to perform, which depends on <see cref="CrossoverBalancer"/>.
/// Before crossover is done a random number is generated in [0, 1] range - if the
/// random number is smaller than <see cref="CrossoverBalancer"/>, then the first crossover
/// type is used, otherwise second type is used.</para>
///
/// <para>The <b>first crossover type</b> is based on interchanging
/// range of genes (array elements) between these chromosomes and is known
/// as one point crossover. A crossover point is selected randomly and chromosomes
/// interchange genes, which start from the selected point.</para>
///
/// <para>The <b>second crossover type</b> is aimed to produce one child, which genes'
/// values are between corresponding genes of parents, and another child, which genes'
/// values are outside of the range formed by corresponding genes of parents. </para>
/// </remarks>
///
public override void Crossover(IChromosome pair)
{
    DoubleArrayChromosome p = (DoubleArrayChromosome)pair;

    // check for correct pair
    if ((p != null) && (p.length == length))
    {
        if (rand.NextDouble() < crossoverBalancer)
        {
            // crossover point
```

```
                int crossOverPoint = rand.Next(length - 1) + 1;
                // length of chromosome to be crossed
                int crossOverLength = length - crossOverPoint;
                // temporary array
                double[] temp = new double[crossOverLength];

                // copy part of first (this) chromosome to temp
                Array.Copy(val, crossOverPoint, temp, 0, crossOverLength);
                // copy part of second (pair) chromosome to the first
                Array.Copy(p.val, crossOverPoint, val, crossOverPoint, crossOverLength);
                // copy temp to the second
                Array.Copy(temp, 0, p.val, crossOverPoint, crossOverLength);
            }
            else
            {
                double[] pairVal = p.val;

                double factor = rand.NextDouble();
                if (rand.Next(2) == 0)
                    factor = -factor;

                for (int i = 0; i < length; i++)
                {
                    double portion = (val[i] - pairVal[i]) * factor;

                    val[i] = ((val[i] + portion < 1) && (val[i] + portion > 0)) ? val[i] + portion :
                        (((val[i] - portion > 0) && (val[i] - portion <1) )? val[i] - portion : val[i]);
                    pairVal[i] = ((pairVal[i] - portion > 0) && (pairVal[i] - portion < 1)) ? pairVal[i] -
                         portion :
                        (((pairVal[i] + portion < 1) && (pairVal[i] + portion > 0)) ? pairVal[i] + portion
                             : pairVal[i]);
                }
            }
        }
    }
}
```