VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

OPTIMALIZACE TVORBY KONSTRUKČNÍCH TÝMŮ POMOCÍ GENETICKÝCH ALGORITMŮ

OPTIMALIZATION OF CONSTRUCTIONAL TEAMS CREATION BY GENETIC ALGORITHMS

DIZERTAČNÍ PRÁCE
DOCTORAL THESIS

AUTOR PRÁCE — Ing. JIŘÍ ŠPAČEK
AUTHOR

VEDOUCÍ PRÁCE — prof. Ing. PAVEL OŠMERA, CSc.
SUPERVISOR

BRNO 2010

ABSTRAKT

Disertační práce se zabývá optimalizací tvorby pracovních týmů ve firmách. Základem je práce Dr. Mereditha Belbina z Henley Management College, který je autorem tzv. Belbinovy teorie týmových rolí. Tato teorie definuje základní role v týmu včetně popisu vzorců jejich chování, přičemž pro správné fungování libovolného týmu je důležité, aby v něm byly zastoupeny všechny role.

V praxi je však potřeba, aby konkrétní lidé splňovali nejen osobnostní a psychologické předpoklady, ale také odborné znalosti či jiné požadavky. Doplněním těchto parametrů konkrétním lidem však vzniká obrovské množství možných variant výsledného týmu, které není možné tradičními metodami snadno a v reálném čase vyhodnotit. Z toho důvodu byly zvoleny k vyhodnocení tzv. genetické algoritmy, které jsou inspirované přírodními vývojovými procesy popsanými již dříve J. G. Mendelem a Ch. Darwinem. Genetické algoritmy se vyznačují poskytnutím dobrých výsledků řešené úlohy ve velmi krátkém čase, přičemž úloha nemusí mít exaktní zadání a správných řešení proto může být více.

V rámci dizertační práce byl sestaven program v jazyce Java, jehož jádrem je genetický algoritmus a v kterém bylo uskutečněno modelování konkrétních týmů. Následovalo ověření výsledků programu sestavením týmů pro realizaci nových úkolů a sledováním jejich činnosti v praxi. Rovněž byla provedena modelová verifikace týmů sestavených již dříve pouze na základě zkušeností vedoucích pracovníků a byly porovnány výsledky.

KLÍČOVÁ SLOVA

genetický algoritmus, optimalizace, Belbinova teorie, tým, budování týmu, spolupráce

ABSTRACT

The thesis pertains to optimisation of workgroups in companies. It is based on the work of Dr. Meredith Belbin from the Henley Management College, who is the author of the so-called Belbin's team role theory. The theory defines fundamental roles within a team including specifications of the behavioural patterns while stipulating that in order to ensure proper functionality of a team, it is essential for all the roles to be represented in it.

However, in practice it is necessary for specific people to comply not only with certain personal and psychological requirements but also professional expertise and other requirements. Nevertheless, by the means of adding these parameters to specific people, an enormous number of possible alternatives of the resulting team, which may not be evaluated (easily and in the real time) using traditional methods, proves to come to existence. Therefore, the so-called genetic algorithms inspired by natural development processes originally described by J. G. Mendel and Ch. Darwin were selected for evaluation purposes. The genetic algorithms feature good solutions to the task to be resolved in a very short time while the task does not have to be based on exact specifications and therefore several solutions might exist.

A Java application was created within the scope of the thesis; its core comprises a genetic algorithm and it was used for the purpose of modelling of specific teams. The results provided by the application were subsequently verified by the means of creation of teams used for completion of new tasks and monitoring their activities in practice. Furthermore, the model verification of teams previously created solely on the basis of experience of executives was performed and the respective results were compared.

KEY WORDS

genetic algorithm, optimisation, Belbin theory, team, teambuilding, cooperation

BIBLIOGRAFICKÁ CITACE

ŠPAČEK, J. *OPTIMALIZACE TVORBY KONSTRUKČNÍCH TÝMŮ POMOCÍ GENETICKÝCH ALGORITMŮ*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2010. 114 s. Vedoucí disertační práce prof. Ing. Pavel Ošmera, CSs.

MÍSTO ULOŽENÍ PRÁCE

Oddělení pro vědu a výzkum FSI VUT v Brně

PROHLÁŠENÍ AUTORA

Prohlašuji, že jsem předloženou práci vypracoval samostatně pod vedením doc. Ing. Josefa Šupáka, CSc. a prof. Ing. Pavla Ošmery, CSs a že veškerá použitá literatura je v této práci uvedena.

……

Ing. Jiří Špaček

PODĚKOVÁNÍ

Na tomto místě bych rád poděkoval svému školiteli prof. Ing. Pavlovi Ošmerovi, CSs. za jeho přístup a podporu při vypracovávání této disertační práce.

Dále bych chtěl poděkovat PhDr. Emilii Frankové, Ph.D. za její cenné rady v oblasti psychologie, personalistiky a hodnocení týmů ve firmách a také Ing. Jiřímu Vepřekovi za jeho pomoc při sestavování programu využívajícího GA v programovacím jazyce Java.

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM GRAFŮ

1 ÚVOD

Všechny konstrukční, výrobní i jiné podniky a firmy pociťují v současnosti ze strany zákazníků velký tlak na vysokou kvalitu svých výrobků a služeb. Pro vytvoření požadované kvality v často krátkých termínech je klíčové, aby byl sestavený vhodný pracovní tým, který má daný projekt řešit. Sestavování vhodných týmů pro jednotlivé firemní projekty a zakázky je tedy pro každou firmu vyloženě existenční záležitostí. Tomu může pomoci důkladné poznání osobnostních vlastností (za pomoci ověřených psychologických metod) a odborných předpokladů jednotlivých lidí ve firmě. Vzhledem ke značnému množství výsledných variant různých týmů je vhodné pro podporu manažerského rozhodování využít moderních výpočetních metod, kterými se tato disertační práce zabývá.

2 CÍL PRÁCE

Tato práce se zaměřuje na sestavení optimálního pracovního týmu z množiny možných řešení. Předpokladem výběru je, že o každou pracovní pozici může soutěžit více jedinců, z nichž je následně vybrán takový jedinec, který se nejlépe přibližuje požadované kvalitě. Definice požadované kvality je v tomto případě multioborová, jelikož spojuje psychologické a technické aspekty. Z tohoto důvodu vznikla potřeba navrhnout ucelené řešení, které definuje způsob ohodnocení reálných vstupních parametrů a jejich následné zakódování do numerické podoby. Pro zpracování numerických kódů je dále potřeba navrhnout fitness funkci, vhodné vstupní parametry a sestavit program, který s využitím genetického algoritmu vypočítá optimální pracovní tým.

Cílem práce je navržení následujících konkrétních bodů:

- matice hodnotících kritérií,
- fitness funkce,
- vhodné vstupní parametry genetického algoritmu,
- způsob zakódování jedinců,
- způsob zadání požadavků na výsledný tým,
- váhová matice významnosti dílčích vlastností,
- vlastní algoritmus pro testování této hypotézy,
- způsob interpretace výsledků.

3 GENETICKÉ ALGORITMY

3.1 HISTORIE VZNIKU

V současnosti je čím dál častější používat na řešení složitých matematických, technických i netechnických problémů evoluční výpočetní techniky, respektive evoluční algoritmy. Tyto metody, či lépe řečeno algoritmy, v podstatě napodobují principy biologické evoluce. Jak je možné vidět v živé přírodě kolem nás, evoluce představuje v podstatě jednoduchý, ale na druhou stranu velmi robustní a výkonný optimalizační prostředek. Biologové říkají, že platí pro jednobuněčné organismy stejně tak, jak pro nejsložitější organismy sestávající se s tisíců miliard buněk.

Vědci se těmito myšlenkami náhodných genetických změn a přirozeného výběru začali zabývat a snažili se vnést tyto principy do řešení praktických problémů, se kterými se lidé denně setkávají. Simulace tisíců až milionů evolučních cyklů, které probíhají v přírodě, vyžaduje výkonné počítače, které v počátcích rozvoje těchto metod nebyly k dispozici. Proto až ve druhé polovině 20. století začali na různých pracovištích ve světě při řešení odlišných problémů vznikat podle vzoru přirozené evoluce různé, ale v určitých ohledech podobné přístupy. V Německu v polovině 60. let vyvíjeli I. Rechenberg a H.P. Schwefel při optimalizaci konstrukčních úloh tzv. evoluční strategie. Lawrence Fogel v USA při modelování a návrhu automatů zavedl techniku s názvem evoluční programování. Jako počátek genetických algoritmů jsou považované práce skupiny pod vedením Johna Hollanda z University of Michigan v USA v 70. letech 20. století. Historicky mladší genetické programování je evoluční přístup Johna Kozu (USA) na přelomu 80. a 90. let určený zejména na automatizovaný vývoj a optimalizaci struktur a programů nebo na strojové učení. Tyto směry, popřípadě ještě několik dalších, dnes zastřešuje pojem evoluční algoritmy.

Všechny tyto přístupy se vyvíjely a vyvíjejí dodnes a současně se navzájem ovlivňují, takže hranice mezi nimi se čím dál víc ztrácejí. Všechny mají společné vlastnosti, jejichž základ tvoří optimalizace na bázi stochastických změn a soutěžení jednotlivých potenciálních řešení. Nejpopulárnějším představitelem této skupiny jsou právě genetické algoritmy.

3.2 PRINCIPY A ZÁKLADNÍ POJMY

Genetické algoritmy jsou univerzálním prohledávacím nebo optimalizačním přístupem. V ohraničeném prostoru přípustných řešení daného problému je možné najít globální optimum z pohledu zvolené účelové funkce nebo se mu alespoň přiblížit. Při tom se uplatňují principy vypozorované v živé přírodě, především náhodné změny v populaci, přežití nejsilnějších, respektive nejpřizpůsobivějších jedinců, nevyhnutelnost zániku nejslabších, neživotaschopných, respektive nepřizpůsobivých jedinců.

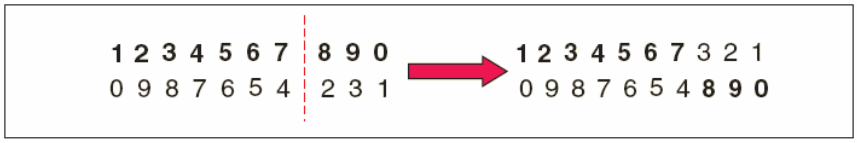
Genetický algoritmus pracuje se skupinou více potenciálních řešení daného problému – s tzv. populací. Každé potenciální řešení (nebo jedinec) je přitom reprezentované uspořádanou množinou parametrů nebo hodnot, které úplně charakterizují jeho vlastnosti a jejichž nejlepší kombinaci hledáme.

Prvky této množiny se nazývají geny a jejich typy mohou být:

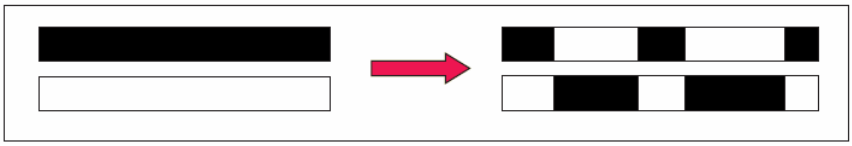
- binárně-číselné,
- celo-číselné,
- reálně-číselné,
- symbolové,
- kombinované.

Závisí přitom vždy na charakteru řešeného problému. Prvky jsou uspořádané do posloupnosti, která se nazývá řetězec či chromozóm.

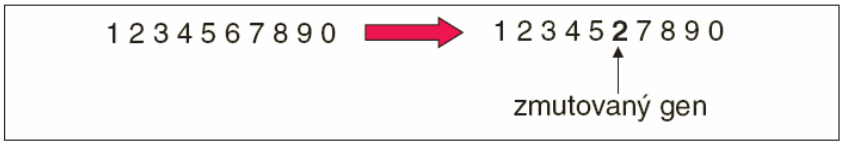
Genetická operace křížení náhodně zkombinuje geny dvou nebo více tzv. rodičovských řetězců do jednoho nebo více tzv. potomků. Při běžném způsobu křížení se dva rodičovské řetězce rozdělí na jednom nebo více náhodných místech (přitom na obou řetězcích se jedná o stejné místo) a potomkové získají střídavě každou doplňkovou část takto oddělených podřetězců od každého z rodičů. Operace mutace náhodně změní náhodně zvolené geny náhodně vybraných jedinců. Existuje více typů operací křížení a mutací a jejich volba může záviset na konkrétní aplikaci.



Obr. 1: Princip jednobodového křížení dvou řetězců [1]



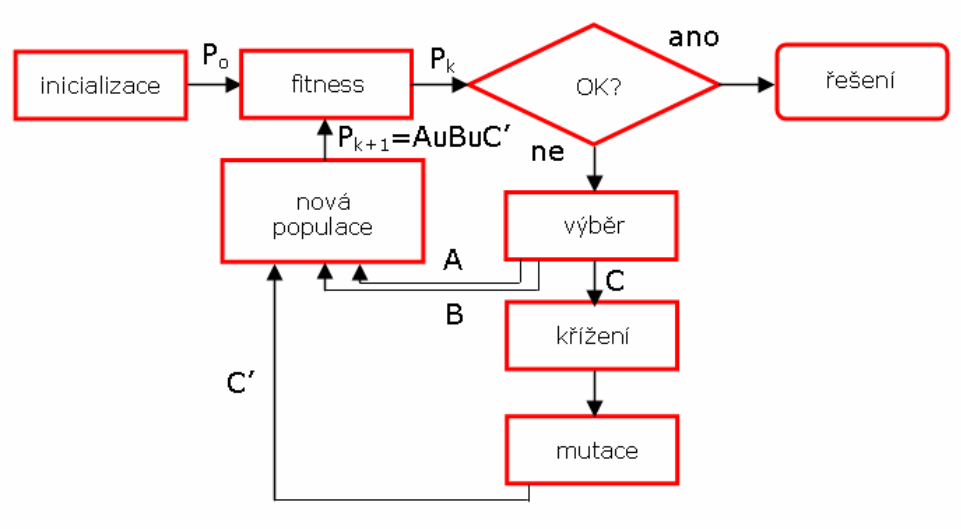
Obr. 2: Vícebodové křížení dvou řetězců [1]



Obr. 3: Příklad mutace řetězce [1]

3.3 POSTUP VÝPOČTU

Schéma výpočtu genetického algoritmu je na obr. 6. Počáteční populace řetězců (P0) před prvním výpočtovým cyklem (výpočtovým cyklem je zde myšlena generace) se získá zpravidla náhodným vygenerováním jejich genů v rámci uvažovaných ohraničení.V každém výpočtovém cyklu se pro každý řetězec vyčíslí hodnota účelové funkce (nazývá se fitness) a to např. výpočtem, počítačovou simulací atd. Má význam míry vhodnosti nebo úspěšnosti nebo úspěšnosti daného řetězce. Potom se vytvoří tři skupiny řetězců. Skupina A obsahuje nejlepší jedince (alespoň jednoho). Skupina jedinců B, které mohou být vybrané například náhodně, se dostanou do nové populace nezměněné. Někdy se používají i jiné metody výběru, např. ruletový výběr, turnajový výběr atd.



Obr. 4: Blokové schéma genetického algoritmu [1]

Dále se některou z uvedených metod vybere skupina jedinců C, která je určená na inovaci. V této skupině se vytvoří náhodné páry řetězců, s kterými se uskuteční genetická operace křížení a následně se na této skupině realizuje ještě mutace. Takto zmodifikovaní jedinci (označené jako skupina C') dokompletují novou populaci Pk+1. Ta se stane objektem stejného postupu v další generaci. Přitom je důležité, že při výběru do skupiny C, případně také do B, mají větší pravděpodobnost přežití úspěšnější jedinci, ale určitou malou šanci mají i méně úspěšní jedinci.

Pokud se uvedený postup opakuje v rámci mnoha generací (např. stokrát, tisíckrát nebo milionkrát), řešení konverguje ke globálnímu optimu. Počet potřebných generací závisí na povaze a složitosti řešeného problému. Algoritmus (běh programu) se může ukončit po dosáhnutí požadovaného, respektive přijatelného řešení nebo nejčastěji po ukončení požadovaného počtu generací.

Uvedené schéma genetického algoritmu není jediné možné a jediné používané. Volba struktury genetického algoritmu může záviset na typu úlohy, stejně jako na zvyklostech a zkušenostech jeho autora. Podobně je to i při genetických operacích křížení, mutací a při výběrech, kde existuje více modifikací.

3.4 VLASTNOSTI GA A OPTIMALIZAČNÍ METODY

Genetické algoritmy se od většiny konvenčních optimalizačních metod liší několika znaky:

- dokáží vyváznout z okolí lokálního extrému a přibližovat se ke globálnímu extrému (na rozdíl od běžných gradientových metod),
- uskutečňují paralelní prohledávání ve více směrech současně,
- nevyžadují pomocné informace o vývoji řešení - např. gradient účelové funkce atd.,
- intenzivně využívají stochastické jevy,

- jsou schopné řešit optimalizační problémy s desítkami až stovkami proměnných,
- poměrně jednoduchá aplikace na řešení širokého spektra různých typů problémů,
- patří k výpočetně nejnáročnějším přístupům.

3.5 OPERÁTORY GENETICKÝCH ALGORITMŮ

Nejběžnější používané operátory jsou selekce, křížení a mutace.

3.5.1 Selekce

Operátor selekce (výběr) vytváří novou populaci P(t+1) výběrem jednotlivců s možným opakováním ze staré populace P(t). Výběr může být proveden několika způsoby. Nejběžnější je náhodný výběr pomocí rulety (roulette wheel selection), kde pravděpodobnost výběru jednotlivce ps(xi) každého jednotlivce je úměrná jeho fitness. Selekce jedinců představuje významnou část genetických algoritmů. Výběr jedinců do reprodukčního procesu musí na jednu stranu dostatečně upřednostňovat jedince s vyšší hodnotou fitness, na druhou stranu musí novou populaci vybrat dostatečně různorodou. Jestliže selekční algoritmus nesplňuje jeden z těchto požadavků vede to v prvním případě k pomalé konvergenci algoritmu, ve druhém k tzv. předčasné konvergenci (do lokálního optima funkce).

3.5.2 Křížení

Operátor křížení (crossing-over) je charakteristický pro genetické algoritmy a představuje pro ně základní operátor pro evoluci populace. Zastánci genetických algoritmů vyzdvihují obvykle přínos křížení pro výměnu informací mezi jedinci. Odpůrci genetických algoritmů naopak považují křížení za rozbíjení bloků bitů a operátor křížení aplikují stejně jako mutaci s velmi malou pravděpodobností.

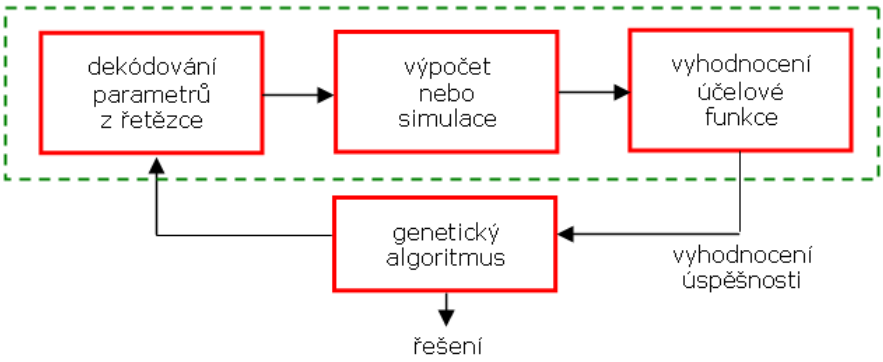
Teorie stavebních bloků (building blocks) vysvětluje konvergenci genetických algoritmů. Genetické algoritmy jsou podle této teorie schopné identifikovat kvalitní bloky genů (bitů) a pomocí rekombinačního operátoru (křížení) sestavovat bloky s rostoucí velikostí. Tento růst se projevuje navenek konvergencí algoritmu k maximální fitness. Operátor křížení je prováděn s pravděpodobností pc. Existuje celá řada variant. Základem je náhodný výběr dvojice jednotlivců, u kterých dochází k výměně genové informace (rekombinaci) tak, že od bodu křížení dojde k výměně genů. Často se tato operace neprovádí se 100% pravděpodobností, ale např. s pravděpodobností okolo 95%. Tímto způsobem je část jedinců pouze reprodukována bez výměny genů.

3.5.3 Mutace

Posledním ze základních operátoru genetických algoritmů je operátor mutace. Standardní operátor mutace modifikuje (vytváří mutanty) genů s pravděpodobností pm. Nejběžnější je bitová negace, která se používá s pravděpodobností 0,0005 až 0,01.

Mutace je pro genetické algoritmy zdrojem nových informací. Vliv mutace může být zcela zanedbatelný nebo naopak s fatálními důsledky pro jedince. Příliš velká pravděpodobnost mutace pm způsobuje nestabilitu vývoje populace a naopak příliš malá mutace nedokáže přinášet dostatek nových informací pro další vývoj.

Existuje celá řada speciálních mutačních operátorů pro konkrétní úlohy. Např. operátor inverze. Tento operátor invertuje pořadí jednotlivých elementů (bitů) mezi dvěmi náhodně vybranými body uvnitř chromozómu.



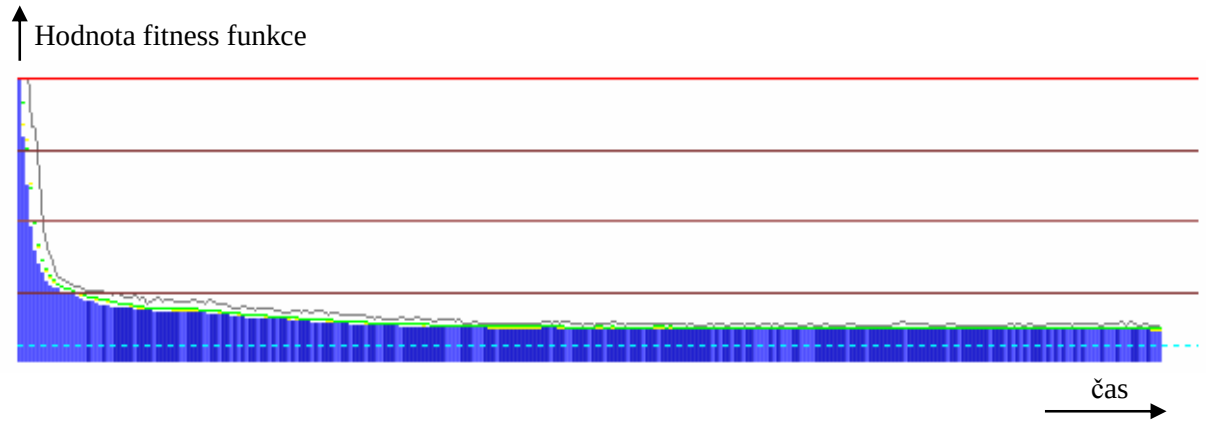
Obr. 5: Použití GA při řešení problému [1]

3.6 MOŽNOSTI APLIKACE GENETICKÝCH ALGORITMŮ

Genetické algoritmy se dají využít na řešení velmi širokého spektra úloh. Tyto metody jsou vhodné pro řešení problémů reálného světa, které jsou plné nepřesností, potřebné informace jsou nezřídka nedostupné a analyzované systémy jsou nepřesně nebo neúplně definované (např. v ekonomických nebo společenských systémech). Podmínkou je schopnost definovat účelovou funkci, která se má minimalizovat nebo maximalizovat. Při minimalizaci se zpravidla jedná o snižování odchylky od požadovaného stavu, o minimalizaci spotřeby energie, paliva, ztrát, nákladů, minimalizaci nežádoucích účinků atd. Při maximalizaci se zpravidla jedná o zvyšování účinnosti, výkonu, zisku atd. Další podmínkou je existence počítačové reprezentace optimalizovaného problému. Tímto je myšleno, že pro každý libovolný bod prohledávaného prostoru, respektive pro libovolné potenciální řešení lze počítačem vyčíslit hodnotu jeho účelové funkce, tedy ohodnotit ho z hlediska úspěšnosti a míry splnění požadovaného cíle. Přitom vůbec nezáleží na typu daného procesu, který může být např. ekonomického, společenského, fyzikálního, chemického nebo biologického charakteru.

Optimalizovaný problém je z hlediska genetického algoritmu černou skříňkou, která poskytuje velké množství více či méně smysluplných možností řešení, přičemž každou z nich musíme umět ohodnotit např. číselně nebo alespoň porovnat úspěšnost vůči jiným možnostem. Optimalizací je potom možné nazývat hledání takové možnosti (struktury vnitřních vazeb, sady parametrů atd.), která nejlépe splňuje určité požadavky.

Existují problémy, které jsou s použitím konvenčních optimalizačních přístupů a metod řešitelné jen těžko nebo dokonce vůbec. V takovém případě lze využít genetických algoritmů. Mezi takové problémy patří např. hledání globálních extrémů nelineárních multimodálních funkcí, těžké kombinatorické nebo grafově orientované problémy (zde řadíme např. problém obchodního cestujícího), mnohaparametrové problémy, úlohy s kombinovanými typy proměnných (binární, celočíselné, symbolové atd.), úlohy s velkým počtem různých typů omezení (nerovnosti, rovnosti, logické podmínky) a úlohy s výpočetně náročným vyhodnocením účelové funkce (konstrukční výpočty, počítačové simulace).



Obr. 6: Konvergence GA ke globálnímu optimu je velmi rychlá [17]

Významným polem působnosti jsou inženýrské aplikace. Genetické algoritmy jsou silným nástrojem při optimalizaci elektrických obvodů, optimalizace provozu vlakové sítě, návrhu antén, filtrů, technologických procesů, regulačních obvodů atd. V oboru stavebnictví je možné genetickými algoritmy optimalizovat konstrukce budov, dopravních komunikací, inženýrských sítí atd. Další obecně využívané řešení je v oblasti distribučních a dopravních úloh, kde hovoříme především o hledání nejkratší nebo nejlevnější cesty. V oboru strojírenství lze řešení najít v oblasti návrhu převodovek nebo řezných plánů.

Konkrétním příkladem je využití genetických algoritmů při optimalizaci motorů Boeingu 777, kde se zdánlivě malou konstrukční optimalizací získala na dané poměry mimořádně významná úspora paliva cca 2,5%. Převedeno na finanční ukazatele představuje tato úspora při celoročním provozu jednoho letadla 2 miliony amerických dolarů.

Další zajímavou aplikací je využití genetických algoritmů pro "vyšlechtění" nového typu monopostu stájemi BMW Williams a Jordan. Na vozech formule F1 se upravují tisíce parametrů, které jsou v závodech rozhodujícím faktorem úspěchu (velikost zadních křídel, výběr pneumatik, nastavení výšky sedadla, úpravy rychlostních stupňů atd.). Požadavek kladený na genetický algoritmus byl jednoznačný – vytvořit monopost s takovými aerodynamickými vlastnostmi, aby zajetí jednoho kola bylo co nejrychlejší. Na začátku se vzala dvojice stávajících vozů a vědci Peter Bentley a Krysztof Wloch z londýnské University College vybrali hodnoty 68 náhodných parametrů a užili je při optimalizaci. Po 40-ti generacích výpočtů byly vyvinuté vozy mnohem rychlejší než auta braná jako "Adam a Eva". Konkrétně se jednalo o zkrácení času potřebného pro zajetí okruhu v Nürburgringu o sedm sekund, což je při závodech vozů F1 velmi podstatná úspora. [1, 2, 5, 6, 7, 9, 18, 19, 20, 21, 22, 23, 24, 25, 26]

4 TÝMOVÁ PRÁCE

4.1 VÝHODY PRÁCE V TÝMU

Každý člen má jisté zkušenosti, dovednosti, svůj způsob myšlení a vidění světa. Při společném řešení se zkušenosti a dovednosti jednotlivých členů kombinují, na problém je nahlíženo z různých úhlů pohledu a tím se i nabízejí širší možnosti řešení. Vzájemnou komunikací a spoluprací se zároveň rozvíjí každý člen. Omezený úhel pohledu jednotlivých lidí na určité věci se eliminuje spektrem rozmanitých pohledů všech členů týmu.

4.2 SYNERGICKÝ EFEKT

Výkon týmu převyšuje pouhý součet možností všech členů týmu. Týmová činnost zvyšuje efektivnost práce, spojuje lidi, kteří se vzájemně doplňují, obohacují a inspirují nápady svých kolegů. Tak se zvýší nejen celkový výkon skupiny, ale i výkon každého člena.

4.3 POZITIVA TÝMU

- Budují se v něm vztahy mezi lidmi.
- Zlepšuje se komunikace členů.
- Pracovní atmosféra je tak příjemnější.
- Využívá znalostí, dovedností a zkušeností všech.
- Pracuje s tvořivostí a fantazií lidí.
- Učí respektu a úctě k druhým.
- Někdy urychlí cestu ke správnému řešení.
- Snižuje u svých členů obavy z neúspěchu a zodpovědnosti.
- Zvyšuje jejich sebevědomí.
- Přisuzuje jim určité postavení a role.
- Poskytuje jim jisté uznání.

4.4 POTENCIÁLNÍ NEGATIVA TÝMU

- Hrozba konfliktů.
- Může potlačit individualitu.
- Někteří členové nedokážou vyjít s ostatními.
- Vyžaduje přizpůsobení se normám a pravidlům.
- Některá krajní, neobvyklá řešení jsou potlačena, byť mohou být značným přínosem.
- Nutnost společného cíle.
- Spolupráce může být časově náročná.

4.5 VYBRANÉ ZNAKY TÝMU

- Společné prvky (minulost, úkol, vztah, úspěch, návyk, pravidla…).
- Ztotožnění se s cíli (skupinová diskuze, rozhodování).
- Komunikace (vyjádření svých názorů, získávání i poskytování informací).
- Soudržnost (fyzická blízkost členů, podobná práce, kombinace vlastností, atmosféra, skupinové normy, pravidla, standardy, struktura a organizace – formálnost či neformálnost). [11]

Tým si má počínat jako podnikatel v podniku, a to v rozdílném rozsahu (v závislosti na koncepci). Podle zásady, že skupiny jsou schopny řídit samy sebe, se tak zlepšuje schopnost podniků reagovat a redukuje se komplexnost starého principu liniového managementu.

Klasické týmy (projektové a jiné speciální týmy) lze charakterizovat následujícím způsobem:

- Počet osob zpravidla 3-8
- Fungují na principu sebeorganizace a sebeřízení
- Sestaveny často mezioborově, eventuálně zahrnují i více hierarchických rovin.
- Vedeny oficiálním vedoucím týmu.
- Po splnění úkolu se rozpouštějí. [12]

4.6 VÝKONNOSTNÍ KŘIVKA TÝMU

4.6.1 Pracovní skupina

Pracovní skupina nemá žádnou zvláštní potřebu se vyvíjet a zlepšovat svůj výkon. Lidé pracují ve skupině, aby mohli sdílet informace, společně se rozhodovali a koordinovali svou činnost. V pracovní skupině je vždy kladen důraz na to, aby mohl každý jedinec převzít odpovědnost za svou oblast. Odpovědnost se v pracovních skupinách nesdílí.

4.6.2 Pseudotým

Pseudotým je skupina lidí, kteří jsou označováni jako tým nebo si tak sami říkají a kteří by potenciálně týmem mohli být. Ve skutečnosti však v jejich skupině zcela chybí koordinace činnosti či kolektivní odpovědnost. Lidé v pseudotýmu jednaní ve skutečnosti individuálně a zajímají se jen o svá oddělení či své povinnosti.

4.6.3 Potencionální tým

Je v daleko lepší pozici, než pracovní skupina a pseudotým. Jedná se o skupinu lidí, která vidí, že je třeba snažit se o lepší výkon a také v tomto smyslu jedná. Drží je však při zemi především nedostatečně jasné společné cíle a pracovní postupy, které zdůrazňují osobní odpovědnost a brání skutečné koordinaci činností. S dobrým vedením a managementem se může potenciální tým velmi rychle proměnit v tým skutečný a zvýšit tak svoji produktivitu. Mnohem častěji se však potenciální tým dál potácí bez jasného směřování.

4.6.4 Skutečný tým

Skutečný tým je malé seskupení lidí, kteří všichni oddaně směřují ke společnému cíli. Sdílejí cíle a každý jednotlivý člen týmu je odpovědný za výsledky celého týmu a také za obecný pracovní přístup týmu. To však neznamená, že jsou všichni členové týmu stejní. Skutečný tým se skládá z lidí, jejichž dovednosti se doplňují a kteří jsou zároveň připraveni osvojit si další dovednosti, bude-li to splnění úkolu vyžadovat. Protože však pracují společně, jsou schopni dosáhnout mnohem lepších výsledků, než by se jim to podařilo, kdyby pracovali jednotlivě nebo v pracovní skupině.

4.6.5 Vysoce výkonný tým

Tento tým je naplněním týmového potenciálu. Členové vysoce výkonných týmů jsou hluboce oddáni nejen tomu, aby tým uspěl, ale i vzájemnému růstu a rozvoji, což je vztah, který je výsledkem úzké interakce a sdílené odpovědnosti za činnost týmu. Tyto týmy mají obvykle výborné výsledky a často dosahují cílů, které se z hlediska podniku na počátku jevily nedosažitelné. I když může mít sám tým krátkou životnost (často se tým po splnění úkolu rozchází), těsné pracovní vazby, které se mezi členy týmu vytvoří, dál působí jako pozitivní prvek v jejich dalším pracovním životě. [3, 4, 10, 13]

5 BELBINOVA TEORIE TÝMOVÝCH ROLÍ

5.1 DEFINICE

Týmová role definovaná Dr. Meredithem Belbinem je: „Tendence chovat se, přispívat a být ve vzájemném vztahu s ostatními lidmi specifickým způsobem.“

5.2 PODSTATA TEORIE

Hodnota teorie týmových rolí dle Dr. Belbina spočívá v tom, že umožňuje jednotlivci i týmu přizpůsobit se vnějším požadavkům a využít sebepoznání k úspěchu. Týmové role dle Dr. Belbina popisují vzorce chování, které ovšem nejsou neměnné a jsou ovlivněny mnoha různými faktory, kterými konkrétní osoba ve svém pracovním i osobním životě prochází.

Belbinova práce na Henley Management College identifikuje celkem devět různých typů chování, každý z nich se nazývá týmovou rolí (původní práce z roku 1981 uvažuje s využitím osmi týmových rolí a revize práce v roce 1993 zavádí ještě devátou roli – Specialista). Každá týmová role má svou vlastní kombinaci přínosů a přípustných slabin. Je běžné, že jeden člověk v sobě zahrnuje více týmových rolí (zpravidla tři až čtyři), které střídá v závislosti na aktuální situaci. Jen velmi vzácně se najdou lidé, kteří představují pouze jedinou týmovou roli. [28]

5.3 POPIS TÝMOVÝCH ROLÍ

5.3.1 Inovátor, IN (Plant, PL)

Jiný název: chrlič

Přínosy:

- Je tvůrčí, nápaditý a neortodoxní. Dokáže řešit náročné problémy.

Přípustné slabiny:

- Ignoruje podružnosti. Je velmi zaujatý vlastními myšlenkami na úkor efektivní komunikace.

Nejpravděpodobněji řekne:

- „A co takhle… “
- „Podívejme se tomu na zoubek…“
- „Mělo by to být oranžové…“
- „Když to obrátíme vzhůru nohama, tak z toho dostaneme…“
- „Nesmíme přehlížet účinky gravitace.“
- „Proč se nevrátíme zpět k podstatě…“

5.3.2 Vyhledávač zdrojů, VZ (Resource Investigator, RI)

Jiný název: shánil

Přínosy:

- Je nadšený a komunikativní extrovert. Objevuje příležitosti. Rozvíjí kontakty.

Přípustné slabiny:

- Je nadmíru optimistický. Může ztratit zájem po opadnutí počátečního nadšení.

Nejpravděpodobněji řekne:

- „To je ale skvělý nápad…“
- „Znám někoho, kdo by mohl…“
- „Nebojte se – mohu je získat z velkoobchodu…“
- „I kdyby hrom bil – bude to bez problémů – můj bratranec totiž...“
- „Dokážu přesvědčit odbyt, aby...“

5.3.3 Koordinátor, KO (Co-ordinator, CO)

Jiný název: předseda

Přínosy:

- Je vyzrálý a sebejistý. Vyjasňuje cíle. Dává lidi dohromady, aby podpořil týmovou diskuzi.

Přípustné slabiny:

- Může se zdát, že manipuluje. Usnadňuje si osobní práci.

Nejpravděpodobněji řekne:

- „Důvod, proč jsme tady, je to, abychom pracovali na… “
- „Nejprve se zaměříme na… a později…“
- „Abych to shrnul, vypadá to, že hlavním bodem je…“
- „Možná, že byste mohli… a pak uděláme…“
- „Abychom se vrátili k základnímu problému, mohli byste…“

5.3.4 Usměrňovač, US (Shaper, SH)

Jiný název: režisér, ředitel, formovač

Přínosy:

- Vyzývá k výkonu, je dynamický, prospívá mu tlak. Má průbojnost a odvahu překonávat překážky.

Přípustné slabiny:

- Má sklony provokovat. Může urážet ostatní.

Nejpravděpodobněji řekne:

- „Musíme udělat…“
- „Ztrácíme čas – musíme…“
- „Ne – mýlíte se – nejdůležitější otázkou je…“

- „Když dáme dohromady to, co jste řekl, s tímto návrhem, tak můžeme.“
- „Když se vám to nelíbí – vyzkoušejte tohle.“

5.3.5 Monitor vyhodnocovač, MV (Monitor Evaluator, ME) 

Jiný název: rejpal, hodnotící kritik

Přínosy:

- Je vážně založený, je stratég a má vysoké nároky. Vidí všechny možnosti. Má přesný úsudek.

Přípustné slabiny:

- Může mu chybět hnací síla a schopnost inspirovat ostatní.

Nejpravděpodobněji řekne:

- „Ten problém s… “
- „Musíme si dávat pozor na…“
- „Nepřehlížejme…“
- „Když se zaměříme na podstatu této věci, měli bychom…“
- „Promyslím si to a konečnou odpověď vám dám zítra.“
- „Obě strany mají svou pravdu.“

5.3.6 Týmový pracovník, TP (Teamworker, TW) 

Jiný název: hasič

Přínosy:

- Spolupracuje, je mírný, vnímavý a diplomatický. Naslouchá, buduje a odvrací třenice.

Přípustné slabiny:

- Je nerozhodný v klíčových situacích.

Nejpravděpodobněji řekne:

- „Josefe, myslím si, že bys měl poslouchat Františka…“
- „Dejme Honzově nápadu šanci…“
- „Nepotřebujeme se dohadovat o…“
- „Proč neřeknete více o...“
- „Až se Petr vrátí z nemocnice, tak bychom mohli...“

5.3.7 Realizátor, RE (Implementer, IMP) 

Jiný název: tahoun

Přínosy:

- Je disciplinovaný, spolehlivý, konzervativní v návycích. Má schopnost činit praktické kroky a akce.

Přípustné slabiny:

- Je poněkud nepružný. Může pomalu reagovat na nové možnosti.

Nejpravděpodobněji řekne:

- „S ohledem na čas, který máme, bychom mohli…“
- „Zcela jistě můžeme v rámci našeho rozpočtu udělat X…“
- „Gravitační analýza je šílený nápad…, ale mohli bychom na spodní část položit těžké závaží.“
- „Napišme si to na tabuli.“
- „Pokud dokážeme přesně definovat tuto část, můžeme si být mnohem jistější výsledkem.“

5.3.8 Kompletovač finišer, KF (Completer Finisher, CF) 

Jiný název: dotahovač

Přínosy:

- Je pečlivý, svědomitý, dělá si starosti. Hledá chyby a přehlédnutí. Plní termíny.

Přípustné slabiny:

- Má sklony přehnaně se strachovat. Neochotně nechává ostatní podílet se na své práci.

Nejpravděpodobněji řekne:

- „Rád bych se ujistil, že… “
- „Nikdy neuděláme…, pokud…“
- „A co takhle…“
- „Ne – musíme… všechno, aby to fungovalo.“
- „A co uděláme s ustanovením 3 v pododstavci (iv) článku G v devátém vydání?“

5.3.9 Specialista, SP (Specialist, SP) 

Přínosy:

- Je cílevědomý, iniciativní a oddaný své profesi. Poskytuje vědomosti a dovednosti, které jsou vzácné.

Přípustné slabiny:

- Přispívá pouze v úzké oblasti. Zaobírá se osobními speciálními zájmy.

Nejpravděpodobněji řekne:

- „Uvidím, jestli se mi podaří sehnat nějaké informace.“
- „Musíme dodržovat profesionální standardy.“
- „Víte, že na výrobu hranaté trubky o stejné kapacitě, jakou má kruhový hrnec, spotřebujete o 7,6 % více materiálu?“
- „Neobtěžujme se vzájemnou schůzkou. Jeden z nás může prostě rozhodnout.“ [14, 15, 28, 29]

6 VLASTNÍ PROGRAM

6.1 UNIKÁTNOST ŘEŠENÍ

Na základě prostudování stávajících publikovaných řešení a dostupných informačních zdrojů jsem dospěl k závěru, že dosud nebylo publikováno žádné ucelené řešení kombinující osvědčenou sociálně-psychologickou metodu sestavování týmů a další odborně-technické znalosti a požadavky na kvalitu jedinců budoucího týmu. Tato disertační práce navrhuje spojení těchto dvou faktorů do jednoho celku.

Problémem numerického sestavování týmů se zabývá rovněž práce od O. Hlaoittinuna, E. Bonjoura, M. Dulmeta [16]. Tato práce však řeší jiné souvislosti.

6.2 POPIS PROBLEMATIKY

Pro dosažení cíle disertační práce byl vytvořen optimalizační program (v jazyku Java, vývojové prostředí NetBeans 5.0 IDE), který slouží k hledání pracovních týmů dle zadaných požadavků. Jádrem programu je genetický algoritmus (GA). V programu je možné, mimo jiné, aplikovat teoretické poznatky týkající se oblasti „personální sociologie“.

GA a jiné algoritmy tohoto druhu patří do skupiny tzv. smíšených algoritmů, využívajících statistické a deterministické metody. GA jsou součástí skupiny algoritmů nazývaných evoluční algoritmy. Ty jsou inspirovány přírodními vývojovými procesy, které již dříve popsali J. G. Mendel a Ch. Darwin.

Princip fungování a nastavení celého programu je vysvětlen na příkladu řešené firmy tak, aby bylo vše jasně a srozumitelně pochopitelné. V tab. 2 je patrné, jak široce je možné pojmout ohodnocení každého člověka pro účely optimalizace níže uvedeným genetickým algoritmem. Pro výpočty konkrétních týmů v konkrétních firmách je nutno nastavit vždy odpovídající parametry pro danou situaci. Vzhledem k tomu, že vytvořený experimentální program nedisponuje grafickým uživatelským rozhraním, je v příloze č.2 uveden i celý zdrojový kód. Uváděná čísla řádků v dalším textu odpovídají zobrazení zdrojového kódu v nezalamované formě. Reálné zadání, nastavení, ohodnocení lidí a vyhodnocení výsledků je podrobně uvedeno v kapitole č. 7.

Pomocí tohoto programu, jehož součástí má být GA, je možné provést optimalizaci z více hledisek, které může uživatel navíc různě upřednostňovat. Program dle zadaných parametrů GA nejprve vytvoří skupinu jedinců (tzv. populaci). Ti budou v jednotlivých cyklech programu, simulující vývojový čas, podrobeni procesům selekce a také rekombinačním změnám, tj. mutace. Křížení nebylo v programu zahrnuto. Jedinec (pracovní tým) bude kódován několika celočíselnými řetězci, oproti obvyklému jedinému řetězci, ve kterých budou uloženy informace o znalostech, schopnostech a osobních vlastnostech. Význam číslic si uživatel bude moci navolit dle potřeb konkrétního úkolu. Uživatel programu má možnost nastavit ideální parametry GA a to tak, že provede určitý počet opakovaných výpočtů, které následně statisticky [27] vyhodnotí (střední hodnoty fitness populace, rozptyl populace, nejlepší řešení v populaci). Vnořením programu do vnějšího cyklu FOR jsou realizovány opakované výpočty (viz řádky programu 67-68 a 1388). Jestliže je na řádku 67 zvolena hodnota 1, provádí program pouze jediný výpočet.

Po ukončení výpočtu jsou veškerá data uložena v několika souborech. Prvních šest souborů (případně jiný počet dle nastavení konkrétní úlohy) slouží k uložení kódů nejlepších jedinců v každé kategorii (včetně celé historie jejich vývoje odpovídající počtu nastavených iterací). Poslední čtyři soubory ukládají statistická data a slouží k nastavení ideálních parametrů GA.

- nejLogistika.txt
- nejObchod.txt
- nejPErsonalistika.txt
- nejPRojekce.txt
- nejStavbyvedouci.txt
- nejUcetni.txt
- statistikaOptiPracTym.txt
- prumernaFitnessStatistika.txt
- rozptylFitnessStatistika.txt
- nejlepsiFitnessStatistika.txt

Význam programového hledání optimálního pracovního týmu lze spatřit v následujících bodech:

- Prohledání velkého prostoru možných řešení v rozumném čase (pozn.: I při malém počtu zaměstnanců v podniku a požadavku složit tým o minimálním počtu členů, bude počet variant možných pracovních týmů velký, viz níže).
- Nalezení těch nejlepších řešení pro podnik.
- Možný finanční a časový přínos programu pro podnik apod.

Na základě primárních zlepšení lze dosáhnout také sekundárních pozitivních efektů, např. zlepšení osobních vztahů na pracovišti apod.

6.3 ZÁKLADNÍ FILOZOFIE PROGRAMU

6.3.1 Algoritmus optimalizačního programu

Na následujícím obrázku je zobrazeno jádro optimalizačního programu v pseudokódu, kterým je genetický algoritmus (GA).

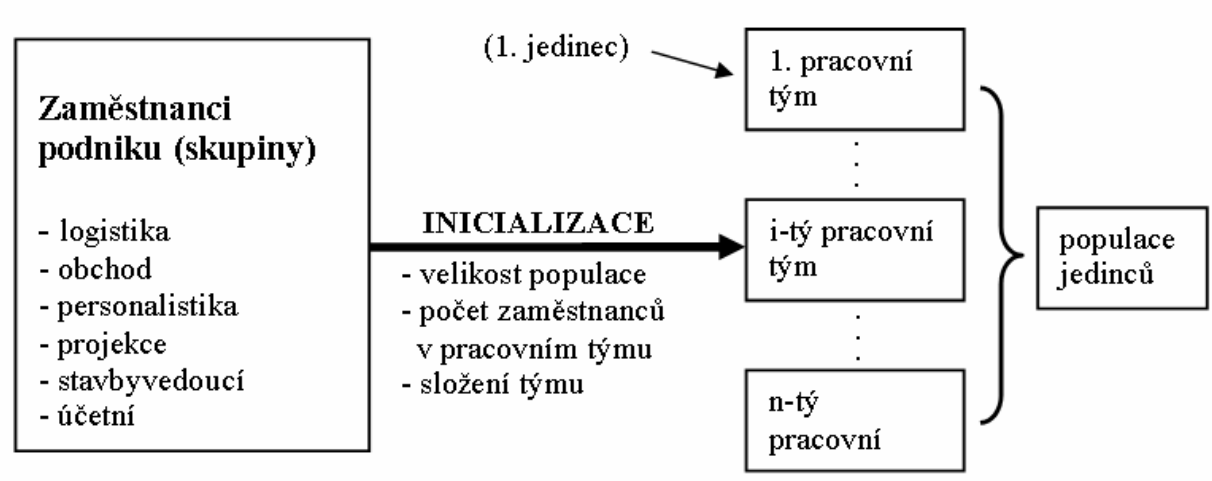
```
begin
       tv := 0;
       inicializace G(tv)
       vyhodnocení G(tv);
       while (not zastavovací_pravidlo) do
       begin
       tv := tv+1;
       selekce G(tv) z G(tv-1);
              změna G(tv);
              vyhodnocení G(tv);
       end
end
```

Obr. 7: Základní tvar evolučního (genetického) algoritmu

6.3.2 Inicializace populace

Předpokládejme ideální nastavení parametrů GA sestávající z vhodných hodnot velikosti populace, pravděpodobnosti mutace a počtu jedinců v turnaji (viz řádky programu 101, 103 a 632). Dále předpokládejme, že máme v programu vytvořenou databázi zaměstnanců (viz řádky programu 16-55), že byly zadány požadavky na velikost pracovního týmu (viz řádek programu 73) a požadované známky u jednotlivých skupin zaměstnanců tým (viz řádky programu 153-158). Po spuštění optimalizačního programu je pomocí generátoru pseudonáhodných čísel, který je implementován v programu, vytvořena počáteční populace určité velikosti, kterou lze navolit (viz řádek programu 101). Populace je vytvořena náhodným výběrem existujících jedinců ze zadané databáze programu (viz řádky programu 16-55). Program tedy nevytváří kódy neexistujících zaměstnanců, ale přímo využívá uživatelem naplněné databáze. Tato populace je poté uložena ve 3D polích (viz řádky programu 108-113). Počet těchto polí odpovídá počtu pracovních skupin, které jsou v programu definované a zároveň tento počet také odpovídá počtu skupin v daném podniku, pro který je program aktuálně nastaven. Způsob inicializace populace bude uveden na příkladu, na který je nyní program přizpůsoben.

V řešeném podniku je *N* zaměstnanců, které lze rozdělit do *M* skupin (logistika, obchod, personalistika, projekce, stavbyvedoucí, účetní). Z těchto *M* skupin zaměstnanců je náhodným způsobem vybrána skupina *K* zaměstnanců předem zadaného složení. Např. je třeba vytvořit pracovní tým s 10-ti zaměstnanci (*K = 10*) s určeným složením pracovního týmu: 1 logistik, 2 obchodníci, 1 personalista, 4 projektanti, 1 stavbyvedoucí a 1 účetní. Dle uvedeného příkladu je náhodným způsobem ze všech skupin vybrán požadovaný počet lidí. Konkrétně ze skupiny logistiků je vybrán 1 zaměstnanec, poté ze skupiny obchodníků jsou náhodně vybráni další 2 zaměstnanci, atd. Tímto způsobem je vytvořen 1. pracovní tým, tj. první jedinec populace (dle terminologie GA). Tento postup se *n*-krát opakuje, dokud nebude vytvořena celá populace jedinců (pracovních týmů) dle zadané velikosti.



Obr. 8: Proces inicializace jedinců (pracovních týmů) z množiny zaměstnanců podniku

Je-li v podniku 35 zaměstnanců ve složení: 7 logistiků, 5 obchodníků, 5 personalistů, 12 projektantů, 4 stavbyvedoucí a 2 účetní, pak je možné vytvořit 1.386.000 různých pracovních

týmů o 10-ti zaměstnancích v požadovaném složení: 1 logistik, 2 obchodníci, 1 personalista, 4 projektanti, 1 stavbyvedoucí a 1 účetní. K výpočtu byl použit následující vztah:

$$C_p(o) = \binom{o}{p} = \frac{o!}{p!(o-p)!}, \text{ kde } (o \geq p). \quad (1)$$

Tento vztah představuje **kombinaci bez opakování**, tj. kombinace *p*-té třídy z *o*-prvkové množiny. Je to každá *p*-prvková podmnožina *o*-prvkové množiny [8]. U kombinace bez opakování nepřihlížíme k uspořádání prvků a každý prvek z daných *o* prvků se v jedné kombinaci může vyskytnou nejvýše jednou. Výpočet podle výše uvedeného vztahu je pro náš příklad následující:

$$\text{počet variant} = \binom{7}{1} \cdot \binom{5}{2} \cdot \binom{5}{1} \cdot \binom{12}{4} \cdot \binom{4}{1} \cdot \binom{2}{1} = 7 \cdot 10 \cdot 5 \cdot 495 \cdot 4 \cdot 2 = 1386000 . \quad (2)$$

Z velkého počtu variant pracovních týmů a z většího počtu požadovaných optimalizačních kritérií vyplývá, že jsou GA vhodným nástrojem k optimalizaci **pracovního týmu**.

6.3.3 Zakódování jedinců

Jak už bylo v úvodu řečeno, bude jedinec (pracovní tým) kódován celočíselnými řetězci určitých délek. Ty lze samozřejmě také změnit (viz databáze v programu). Máme-li dle příkladu vytvořit pracovní tým ve složení: 1 logistik, 2 obchodníci, 1 personalista, 4 projektanti, 1 stavbyvedoucí a 1 účetní, bude jedinec (pracovní tým) tvořen 1+2+1+4+1+1=10-ti celočíselnými řetězci. Je-li stanoveno, že má být vygenerována populace 500 jedinců, bude to znamenat velikost populace, která je složena z 5000 celočíselných řetězců.

V programu je možné navolit individuální délky celočíselných řetězců pro každou skupinu zaměstnanců zvlášť. Např. logistici mohou mít celočíselný řetězec o délce 9-ti číslic, obchodníci řetězec 10-ti číslic, personalisté řetězec 8-mi číslic, projektanti řetězec 8-mi číslic, stavbyvedoucí řetězec 8-mi číslicemi a účetní řetězec 6-ti číslic (volí se podle podrobnosti požadavků na členy konkrétní skupiny). Několik posledních číslic u všech skupin zaměstnanců je vyhrazeno pro uložení informací o osobních vlastnostech (viz řádek 145). Je samozřejmé, že je možné přidávat, resp. ubírat počet pracovních skupin v programu, měnit délky kódů jednotlivých pracovních skupin a vymezit pro ně určitý počet koncových číslic, které budou charakterizovat osobní vlastnosti. Měnit lze také celočíselný rozsah, ze kterého jsou kódy jedinců vytvořeny (viz řádky programu 16-55 a 153-158).

Tabulku je tedy možné přizpůsobit pro konkrétní typ řešené úlohy. Celkový počet hodnocených vlastností je *k*, přičemž odborných vlastností je *n* a osobních vlastností je *k*-*n*. Tento celkový počet hodnocených vlastností *k* dává celkovou délku číselného klíče, kterým je každý zaměstnanec zakódovaný pro vstup do programu.

Tab. 1: Principiální model hodnotící tabulky

Vlastnosti	**Poř. číslo**	**Popis**	**Ohodnocení**	**Rozsah hodnocení**
Odborné	**1**			1-9
	2			1-9
	…			1-9
	n			1-9
Osobní	**n+1**			1-9
	…			1-9
	k			1-9

V následující tabulce je uvedeý příklad, jaký význam mohou mít číslice kódu zaměstnance určité libovolné pracovní skupiny (význam a rozsah hodnocení si zvolí uživatel programu).

Tab. 2: Ukázková definice významu číslic kódu zaměstnance určité pracovní skupiny

Vlastnosti	**Poř. číslo**	**Popis**	**Ohodnocení**	**Rozsah hodnocení**
Odborné	1	Word	**1**	5 = neumí, 4 = jen text, 3 = vkládání obrázků a grafů, 2 = tvorba tabulek, 1 = užívání a tvorba šablon
	2	Excel	**3**	5 = neumí, 4 = jednoduché tabulky, 3 = vkládání grafů a tvorba funkcí, 2 = tvorba maker, 1 = programování
	3	Angličtina	**5**	5 = neumí, 4 = A1, 3 = B1, 2 = C1, 1 = C2 (mezinár. stup. A1-C2)
	4	Technické normy	**2**	5 = nezná , 4 = po zkušební době, 3 = rok ve firmě nebo základní školení, 2 = do tří let nebo podrobnější školení, 1 = více let a podrobnější školení
	5	CAD systémy	**2**	5 = neumí, 4 = prohlížení výkresů, 3 = tvorba výrobních výkresů, 2 = tvorba sestav, 1 = využívání a sdílení dat výkresové dokumentace rozsáhlých projektů
Osobní	6	Týmový hráč	**4**	5 = vůbec, 4 = v malém týmu je-li veden , 3 = v malém týmu sám, 4 = ve středním týmu sám, 5 = ve velkém týmu sám
	7	Vztah s kolegy	**1**	5 = velmi konfliktní, 4 = občas konfliktní, 3 = neutrální, 2 = dobré, 1 = vynikající
	8	Schopnost učit se	**3**	5 = žádná iniciativa, 4 = pokud musí, 3 = částečně dobrovolně, jinak pokud musí, 2 = dobrovolně, 1 = dobrovolně z vlastní iniciativy bez vedení v rámci školení nebo kursu
	9	Spolehlivost	**2**	5 = problémová, 4 = pod dozorem, 3 = pravidelné kontroly, 2 = občasné kontroly, 1 = bez dozoru

Kód uvedeného zaměstnance bude mít tedy podobu: **135224132**. Celkový počet hodnocených vlastností je *k=9*, přičemž odborných vlastností je *n=5* a osobních vlastností je *k-n=9-5=4*. Chceme-li u každé skupiny zaměstnanců sledovat jiný počet vlastností a případně současně i s jiným číselným rozsahem hodnocení, musíme vytvořit příslušný počet hodnotících (kódovacích) tabulek. Pokud chceme sledovat stejné vlastnosti napříč všemi skupinami zaměstnanc, stačí jediná hodnotící (kódovací) tabulka.

6.3.4 Ohodnocení jedince (pracovního týmu)

Nezbytnou částí genetického algoritmu (GA) je přiřazení ohodnocení každému jedinci (pracovnímu týmu). V optimalizačním programu je k dispozici několik dílčích fitness funkcí, kterými lze postihnout různé požadavky na pracovní tým (snaha postihnout jak znalosti, zkušenosti, osobní vlastnosti jednotlivců v týmu, tak docílit rovnoměrného rozložení činnosti mezi nimi). Tyto dílčí fitness funkce se pak skládají do společné fitness funkce. Každá dílčí fitness funkce je navíc násobená váhou o určité hodnotě před tím, než se provede vlastní skládání. Tím lze upřednostňovat určité kritérium na pracovní tým před jinými.

Na základě ohodnocení lze dle definovaných fitness funkcí (hodnotících funkcí) stanovit, jakou kvalitu má jedinec. Dle fitness funkce jsou jedinci v další fázi algoritmu selektováni.

V programu byly zavedeny 3 hodnotící funkce:

- Dílčí fitness funkce FZZ (znalosti a zkušenosti)
- Dílčí fitness funkce FSE (synergický efekt)
- Dílčí fitness funkce FOV (osobní vlastnosti)

Cílem optimalizace programu je upřednostňovat jedince s nižší, a tím lepší hodnotou fitness funkce.

Vstupní část kódu k určení fitness funkce populace je na řádcích programu [*řádek 145-151*]. Na řádku 147 jsou ve 2D poli uvedeny váhy k dílčím fitness funkcím. Dohromady je jich 18, protože jsou v programu definovány 3 dílčí fitness funkce a 6 pracovních týmů. Dílčí fitness funkce, FZZ, FSE a FOV, jsou definovány na [*řádek 149-151*]. Nazývají se: FZZ – fitness funkce znalostí a zkušeností, FSE – fitness funkce synergického efektu, FOV – fitness funkce osobních vlastností. Na [*řádek 153-158*] jsou určeny naše požadavky na hledaný pracovní tým. Jedná se o známky, které požadujeme po zaměstnancích určitých pracovních skupin. Může se stát, že budeme skládat pracovní tým, ve kterém bude např. více zaměstnanců určité pracovní skupiny.

Jak už bylo řečeno, celková fitness funkce pracovního týmu FT je rovna součtu fitness funkcí pracovních skupin násobených váhami:

$$FT = \sqrt{(\sum_{j=1}^{m} V_{1j} \cdot FZZ_j)^2 + (\sum_{j=1}^{m} V_{2j} \cdot FSE_j)^2 + (\sum_{j=1}^{m} V_{3j} \cdot FOV_j)^2} \qquad (3)$$

kde: m – počet pracovních skupin zaměstnanců, V_{1j}, V_{2j}, V_{3j} – váhy dílčích fitness funkcí, FZZ_j – dílčí fitness funkce znalostí a zkušeností, FSE_j – dílčí fitness funkce synergického efektu, FOV_j – dílčí fitness funkce osobních vlastností. Váhy V_{ij} lze nastavit zvlášť pro každou

pracovní skupinu, takže můžeme u jedné pracovní skupiny upřednostnit více synergický efekt a u jiné např. osobní vlastnosti apod. (kde: počet dílčích fitness i = 1-3). Fitness funkci FT určenou vztahem (3) si lze představit jako vektor, jehož složkami jsou dílčí fitness funkce FZZ_j, FSE_j a FOV_j násobené příslušnými váhami.

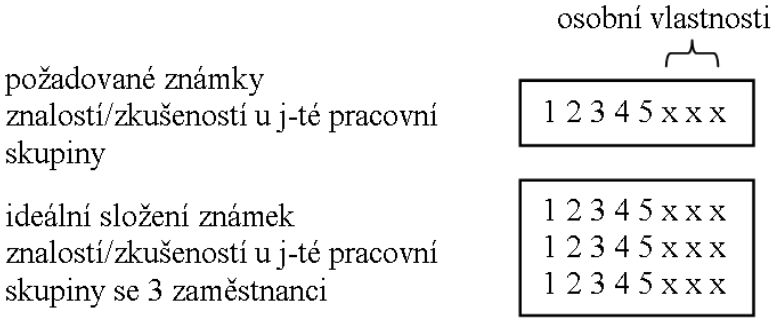
Dílčí fitness funkce znalostí a zkušeností FZZ_j bude mít pro j-tou pracovní skupinu následující podobu:

$$FZZ_j = \sum_{k=1}^{DCH_j - POV_j} abs(PZS_j \cdot PZ_{jk} - SZZZ_{jk}), \qquad (4)$$

kde: PZS_j – počet zaměstnanců j-té pracovní skupiny, DCH_j – délka chromozómu j-té pracovní skupiny, POV_j – počet osobních vlastností j-té pracovní skupiny v chromozómu, PZ_{jk} – požadovaná známka k-té znalosti/zkušenosti u j-té pracovní skupiny, $SZZZ_{jk}$ – součet k-tých známek (z náhradních polí) znalostí/zkušeností zaměstnanců u j-té pracovní skupiny daného pracovního týmu. Na základě tohoto vztahu je vypočítána dílčí fitness FZZ j-té pracovní skupiny. Vztah představuje sumu rozdílů v absolutní hodnotě (vzhledem k tomu, že by dle skutečných výpočtů v programu přibyla řada dimenzí (indexů) v uvedených vztazích, bylo provedeno jisté názornější zjednodušení.

Na [*řádek 194-199*] je pro danou pracovní skupinu volána metoda (podprogram) **urceniFunkceFZZ**. Zdrojový kód této metody je na [*řádek 1440-1448*]. Tato metoda stanoví hodnoty dílčí fitness funkci FZZ a uloží je do 2D pole FZZ.

Dílčí fitness funkce FZZ_j vyhodnotí nejlépe takovou j-tou pracovní skupinu ($FZZ_j = 0$), u které budou známky znalostí/zkušeností co nejvíce totožné s naším požadavkem. Ideální případ, všichni zaměstnanci j-té pracovní skupiny budou mít všechny známky znalostí/zkušeností shodné a tyto známky budou odpovídat našemu požadavku, viz obr. 11.



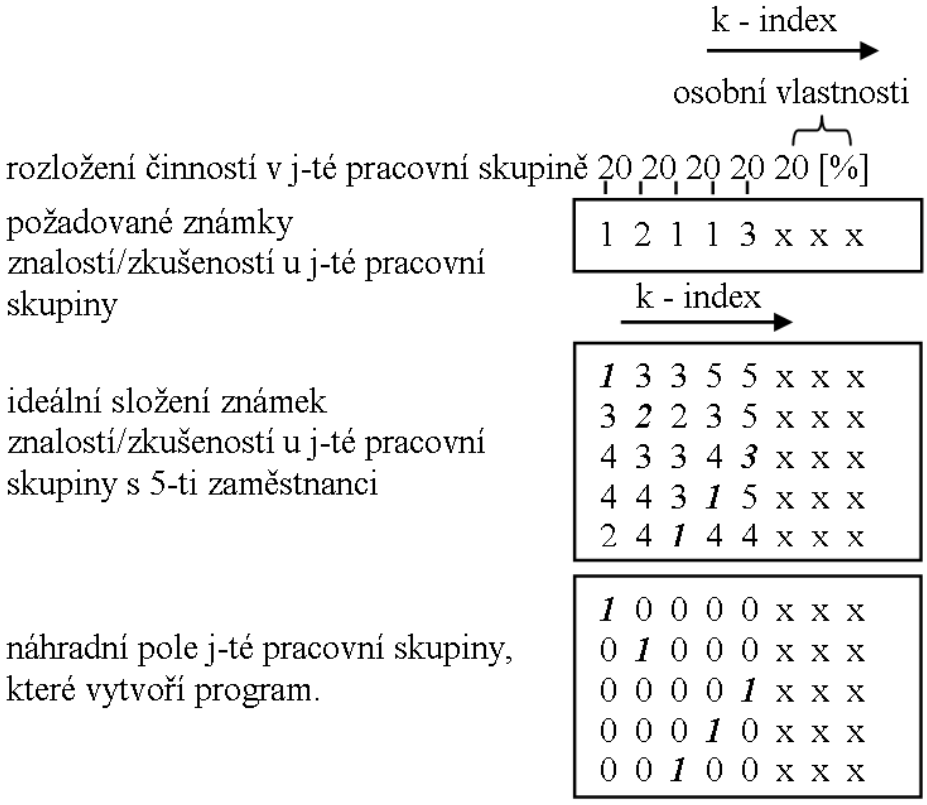
Obr. 9: Ideální příklad složení j-té pracovní skupiny dle dílčí fitness funkce FZZj

Před stanovením dílčí fitness funkce synergického efektu FSE_j vytvoří program z 3D polí populace pomocná 3D pole. V těchto pomocných 3D polích budou některé pozice v chromozómech zaměstnanců nahrazeny nulami podle určitého klíče (bude-li známka zaměstnance lepší nebo rovna požadavku, v pomocném poli bude nahrazena hodnotou 1, bude-li známka horší než požadavek, v pomocném poli se objeví 0). Máme zadáno, jaké

známky u j-té pracovní skupiny požadujeme. Dále je zadáno (to se provede před výpočtem) procentuální rozložení činností v j-té pracovní skupině (např. u projektantů je 60% práce v CAD systému, 10% psaní ve Wordu, 5% čtení technických norem atd. Součet procent jednotlivých činností v i-té pracovní skupině musí být roven 100%). Program dle počtu zaměstnanců v pracovní skupině vypočítá, kolik z těchto zaměstnanců musí být použito na danou činnost při požadované hodnotě známky. Tento počet je označen ve vztahu (5) jako PPZ_{ij}. Program dále zjistí, kolik zaměstnanců v j-té pracovní skupině má k-tou známku v požadované kvalitě, to je ve vztahu (5) označeno jako $SPPZ_{jk}$. Dílčí fitness funkce synergického efektu bude mít následující podobu:

$$FSE_j = V_1 \cdot \sum_{k=1}^{DCH_j - POV_j} \left[abs\left(PPZ_{jk} - SPPZ_{jk} \right) \right] + V_2 \cdot \sum_{k=1}^{DCH_j - POV_j} \sum_{l=1}^{PZS_j} \left[abs(SR_{jl} - SH_{jk}) \right] \qquad (5)$$

Váhy V1 a V2 jsou v programu na řádcích 382-383. Ostatní veličiny: SH_{jk} – střední hodnota známky v pomocném poli j-té pracovní skupiny u k-té znalosti. SR_{jl} – součet známek v pomocném poli j-té pracovní skupiny na l-tém řádku (řádek odpovídá danému zaměstnanci).



Obr. 10: Ideální př. složení j-té prac. sk. dle dílčí fitness funkce FZZj se zobr. k-indexů

Dílčí fitness funkce osobních vlastností je určena tak, že se v j-té pracovní skupině stanoví průměrná hodnota j-té osobní vlastnosti ze všech zaměstnanců a porovná se s požadovanou hodnotou. Vztah má následující podobu:

$$FOV_j = \sum_{j=DCH_j-POV_j+1}^{DCH_j} abs\left[PRZ_{jk} - PZ_{jk}\right], \tag{6}$$

kde: PRZ_{jk} – průměrná k-tá známka osobní vlastnosti u j-té pracovní skupiny. PZ_{jk} – požadovaná k-tá známka osobní vlastnosti u j-té pracovní skupiny. Výpočet je dle vztahu (6) jednoduchý, takže není zapotřebí uvádět příklad.

6.3.5 Selekce populace

Následně je v cyklu programu provedena selekce jedinců z vygenerované populace [*řádek 629-718*] a [*řádek 1570-1586*]. Ta probíhá následně. Náhodným způsobem je vybrán určitý počet jedinců [*řádek 636*], kteří se podrobí turnaji, tj. vyhledá se jedinec s nejnižší hodnotou fitness. Ten postupuje do nové populace. Tento postup se provede opakovaně dle zvolené velikosti populace. Nová populace jedinců je poté uložena opět v 6-ti 3D polích. K selekci populace jedinců byl zvolen turnajový mechanismus. Popis mechanismu je uveden v [9]. Tento mechanismus je z hlediska programové implementace jednoduchý a patří k těm lepším, důvody viz [9]. Pomocí turnajového mechanismu selekce lze snadno změnit selekční tlak, viz [*řádek 632*].

6.3.6 Mutace jedinců v populaci

Selekce nepřináší do nové populace nic nového. K vytváření nových jedinců v populaci jsou vhodné rekombinační operátory křížení a mutace [9]. V optimalizačním programu je implementován pouze operátor mutace. Ten je postačující k tomu, aby do vývojového procesu přinesl něco nového [*řádek 720-747*] a [*řádek 1587-1631*]. Jeho princip je následující. S určitou pravděpodobností P_m [*řádek 103*] je nad každou elementární části všech kódů (chromozomů) proveden náhodný pokus. Je-li tento pokus pozitivní, je jedinec mutován. Tj. tato elementární část kódu (představujícího zaměstnance určité pracovní skupiny podniku) je vyměněna náhodným způsobem za jiný elementární kód (z databáze zaměstnanců za jiného jedince stejné pracovní skupiny). Mutovaná populace je poté uložena opět do nových 6-ti 3D polí.

6.3.7 Nastavení ideálních parametrů GA

Nelze zcela obecně říct, jakou hodnotu by měly mít optimální parametry GA. Doporučuje se, aby populace nebyla příliš malá. Selekční tlak by neměl být příliš velký. Při selekci dochází ke ztrátě variability populace. Pokud by byl počet jedinců t_T vstupujících do turnaje příliš velký, mohlo by dojít k předčasné konvergenci. Rozsah této ztráty je popsán matematickým vztahem v [9]. Řešený optimalizační problém je také závislý na způsobu zakódování populace, viz [9]. Pravděpodobnost mutace P_m by neměla být příliš malá ani příliš

velká. Pokud by byla příliš malá, nepřinášela by do populace mnoho nových informací. To by mohlo vést ke konvergenci do lokálního extrému (v našem případě do lokálního minima). Pokud by byla příliš velká, průběh řešení by nekonvergoval (to by bylo patrné z průběhů **průměrných hodnot fitness funkce** a z **průběhů rozptylu populace** apod.). Všeobecné informace k nastavení parametrů GA lze najít v [6, 7, 8, 9, 13].

Doporučené startovní hodnoty k nalezení vhodných parametrů: velikost populace N = 500, selekce t_T = 2, pravděpodobnost mutace Pm = 1/30, počet iterací pc = 100. Rozsáhlým testováním vytvořeného programu, cílenými krokovými změnami jednotlivých parametrů a následným statistickým vyhodnocením výsledků byly pro řešení zkoumaného typu úlohy nalezeny nové, lepší hodnoty jednotlivých parametrů (viz kapitola 7).

6.3.8 Úpravy optimalizačního programu

Možnosti změny parametrů genetického algoritmu a počtu cyklů programu byly již vysvětleny. Pokud je zapotřebí přizpůsobit program na jiný podnik s určitým počtem zaměstnanců a pracovních skupin, musíme postupovat následovně. Pokud potřebujeme změnit počet zaměstnanců a jejich hodnocení, musíme se podívat na [*řádek 16-55*]. Např. u pracovní skupiny **LOGISTIKA**, viz následující obrázek, jsou definováni 3 zaměstnanci. Pokud potřebujeme změnit hodnocení konkrétního zaměstnance, změníme čísla na konkrétním řádku. Pokud potřebujeme přidat nebo (ubrat) zaměstnance, přidáme nebo (ubereme) 7 čísel ve vnitřní složené závorce na libovolný řádek. Všichni zaměstnanci jsou hodnoceni stejným počtem čísel.

```
//* 1=LOGISTIKA *
        int logistika[][]={
            {2,1,1,1,1,3,2},
            {4,4,3,1,8,2,3},
            {3,2,2,1,6,5,1}
        };
```

Obr. 11: Úsek zdrojového kódu, data o pracovní skupině LOGISTIKA

Nesmíme zapomenout, kolik posledních číslic necháváme pro osobní vlastnosti, což může být u každé skupiny pracovníků jiné, dle požadavků uživatele, viz [*řádek 145*].

Pokud je zapotřebí změnit počet pracovních skupin, skládá se postup z více kroků. Přidáme (ubereme) pracovní skupinu definovanou zdrojovým kódem. Budeme tedy pro zjednodušení uvažovat, že přidáme pracovní skupinu (opačný postup je analogický). Přidejme pracovní skupinu **INVESTICE**. Pro přehlednost tuto skupinu definujme za řádkem č. 56, tj. za 6. pracovní skupinou **ÚČETNÍ**. Zdrojový kód nové skupiny bude analogický tomu, co bylo na předchozím obrázku. Počet zaměstnanců této pracovní skupiny bude odpovídat počtu řádků s čísly ve vnitřních složených závorkách. Počet hodnocených znalostí, zkušenostní a osobních vlastností bude odpovídat počtu číslic na řádku. Po přidání této skupiny do naší „databáze“ v programu je zapotřebí přidat na řádku č. 73 do složených závorek 1D pole tym[] za poslední 6. číslici další číslo. Na tomto řádku se definují požadavky na složení pracovního týmu:

```
int tym[]={2,2,1,4,4,1,nové číslo};
```

Obr. 12: Definice složení pracovního týmu

Na řádcích 108-113 jsou definovány 3D pole inicializace populace. Zde je také zapotřebí přidat nové 3D pole. Dodržujme konvekci, tj. přidejme nové 3D pole za poslední 6. pole **inicializaceU[][][]**. Číslování je od nuly, tedy 7. tým bude mít v hranatých závorkách pořadové číslo 6. Toto přidání vypadá následujícím způsobem:

```
int inicializaceI[][][]=new int[tym[6]][investice[0].length][velPop];
```

Obr. 13: Definice nového 3D pole inicializaceI

Dále přidejme volání metody **plneniPoleInicializace** nejlépe za řádek č. 121:

```
inicializaceI=plneniPoleInicializace(inicializaceI,investice,tym[6],velPop);
```

Obr. 14: Volání metody plneniPoleInicializace

Dále je zapotřebí přidat celé číslo do složené závorky na řádek č. 145 za poslední 6. číslici:

```
int POV[] = {3,3,3,3,3,3,nová číslice};
```

Obr. 15: Definice počtu osobních vlastností u pracovních skupin

Tím definujeme, kolik posledních číslic u kódů zaměstnanců pracovních skupin bude připadat na osobní vlastnosti. Máme-li jednotnou hodnotící tabulku, budou všechna čísla stejná (jak bylo zmíněno výše v kapitole 6.2.3).

Na řádku 147. musíme přidat váhy, kterými se budou násobit příslušné dílčí fitness funkce:

```
double vahy[][]={{1.0,1.0,1.0},{1.0,1.0,1.0},{1.0,1.0,1.0},{1.0,1.0,1.0},
{1.0,1.0,1.0},{1.0,1.0,1.0},{nové váhy}};
```

Obr. 16: Váhy dílčích fitness funkcí

Na řádcích 149-151 musíme změnit číslici 6 za číslici 7 (máme nyní 7 pracovních skupin):

```
double FZZ[][]=new double[velPop][7];
double FSE[][]=new double[velPop][7];
double FOV[][]=new double[velPop][7];
```

Obr. 17: Definice dílčích fitness funkcí FZZ, FSE a FOV

Na řádek 159 přidáme požadované známky pro novou pracovní skupinu s ohledem na kódovací tabulku:

```
int pozadovaneI[]={3,1,1,1,8,6,2};
```

Obr. 18: Definice požadovaných známek u nové skupiny zaměstnanců

Na řádcích č. 160-165 jsou definovány 2D pole SZZZ1-6. Za poslední pole SZZZ6 přidáme pole SZZZ7:

```
int SZZZ7[][]=new int[velPop][personal[0].length];
```

Obr. 19: Definice 2D pole k určení součtu známek zaměstnanců u pracovních skupin

Na řádcích č. 167-172 je volána metoda **souctyZnamekSkupina**. Zde musíme přidat za poslední 6. volání metody další volání:

```
SZZZ7=souctyZnamekSkupina(SZZZ7,inicializaceI,velPop);
```

Obr. 20: Volání metody souctyZnamekSkupina

Na řádcích 194-199 je volána metoda **urceniFunkceFZZ**. Zde musíme na řádek 200 přidat volání metody pro novou skupinu:

```
FZZ=urceniFunkceFZZ(FZZ,SZZZ7,inicializaceI,personal,POV[6],pozadovaneI,velPop,6)
;
```

Obr. 21: Volání metody urceniFunkceFZZ

Na řádcích 207-212 jsou definována pomocná pole. Za to poslední přidáme nové pomocné pole:

```
int pomocneI[][][]=new int[tym[6]][investice[0].length-POV[6]][velPop];
```

Obr. 22: Definice nového pomocného pole

Na řádcích 214-219 jsou definovány 1D pole, která určují procentuální rozložení činností v pracovních skupinách. Tato pole mají význam u dílčí fitness funkce FSE. Na následujícím obrázku je zobrazen způsob definice nového 1D pole procentaI:

```
double procentaI[] = {čísla procentuálního rozložení činností};
```

Obr. 23: Definice 1D pole procentaI

Na řádcích 221-226 je volána metoda **plneniPomocPoli**. Za poslední volání metody musíme přidat volání:

```
pomocneI=plneniPomocPoli(velPop,inicializaceI,POV[6],pomocneI,pozadovaneI);
```

Obr. 24: Volání metody plneniPomocPoli

Na řádcích 248-253 jsou definovány pole PPZ1-6. Za poslední definici přidáme následující kód:

```
double PPZ7[] = new double[inicializaceI[0].length-POV[6]];
```

Obr. 25: Definice pole PPZ6

Tato pole stanoví počet požadovaných známek z procentuálního rozložení činností. To má význam v případě hodnocení pomocí dílčí fitness funkce FSE. Na řádcích 255-260 je volána metoda **prirazeniHodnot**. Přidáme následující kód:

```
PPZ7=prirazeniHodnot(inicializaceI,velPop,POV[6],PPZ7,procentaI);
```

Obr. 26: Volání metody prirazeniHodnot

Na řádcích 282-287 jsou definována 2D pole SPPZ1-6, která slouží ke stanovení skutečného počtu požadovaných známek. Souvisí s dílčí fitness funkcí FSE. Přidáme definici pole:

```
int SPPZ7[][]=new int[velPop][inicializaceI[0].length-POV[6]];
```

Obr. 27: Definice 2D pole SPPZ7

Na řádcích 289-294 je volána metoda **skutPocetZnamek**. Za poslední řádek přidáme volání:

```
SPPZ7=skutPocetZnamek(velPop,inicializaceI,POV[6],pomocneI,SPPZ7);
```

Obr. 28: Volání metody skutPocetZnamek

Na řádcích 316-321 jsou definovaná 2D pole, která slouží ke stanovení součtu prvků náhradních polí v řadách. Za tyto definice doplníme:

```
double sumRadaI[][]=new double[pomocneI.length][velPop];
```

Obr. 29: Definice 2D pole sumRadaI

Na řádcích 323-328 je volána metoda **souctyPrvkuNahPole**. Za poslední volání metody doplníme:

```
sumRadaI=souctyPrvkuNahPole(velPop,pomocneI,sumRadaI);
```

Obr. 30: Volání metody souctyPrvkuNahPole

Na řádku 350 upravíme definici 2D pole **stredHodn**:

```
double stredHodn[][]=new double[7][velPop];
```

Obr. 31: Úprava definice pole stredHodn[][]

Na řádcích 351-356 jsou definované proměnné, ke kterým přidáme novou definici proměnné **pocetZamI**:

```
double pocetZamI=pomocneI.length;
```

Obr. 32: Nová definice proměnné pocetZamI

Na řádcích 357-362 je volána metoda **prumHodnotaSoucet**. K těmto voláním přidáme:

```
stredHodn=prumHodnotaSoucet(sumRadaI,velPop,pomocneI,stredHodn,pocetZamI,6);
```

Obr. 33: Nové volání metody prumHodnotaSoucet

Na řádku 369 je definováno 2D pole sumDelta[][]. V něm jsou uloženy součty odchylek od středních hodnot (význam viz program):

```
double sumDelta[][]=new double[velPop][7];
```

Obr. 34: Úprava definice 2D pole sumDelta

Na řádcích 370-375 je volána metoda **souctyOdchylekOdStredHodnot**. K těmto voláním musíme přidat:

```
sumDelta=souctyOdchylekOdStredHodnot(velPop,pomocneI,sumRadaI,stredHodn,sumDelta,
6);
```

Obr. 35: Nové volání metody souctyOdchylekOdStredHodnot

Na řádcích 384-389 je volána metoda **dilciFitnessFSE**. K těmto voláním přidáme další:

```
FSE=dilciFitnessFSE(velPop,pomocneI,vaha1,vaha2,sumDelta,PPZ7,SPPZ7,FSE,6);
```

Obr. 36: Nové volání metody dilciFitnessFSE

Na řádcích 397-402 jsou definována 2D pole, ve kterých budou uloženy průměrné známky osobních vlastností. K nim přidáme:

```
double prumVlastnostI[][]=new double[velPop][POV[6]];
```

Obr. 37: Definice 2D pole prumVlastnostI[][]

Na řádcích 404-409 je volána metoda **prumVlastnostFOV**. K těmto voláním přidáme:

```
prumVlastnostI=prumVlastnostFOV(velPop, POV, inicializaceI, prumVlastnostI,6);
```

Obr. 38: Volání metody prumVlastnostI

Na řádcích 431-436 je volána metoda **dilciFitnessFOV**. K těmto voláním přidáme následující:

```
FOV=dilciFitnessFOV(velPop,POV,FOV,prumVlastnostI,pozadovaneI,6);
```

Obr. 39: Volání metody dilciFitnessFOV

Na řádcích 443-445 jsou definována 2D pole, ve kterých budou uloženy dílčí fitness funkce roznásobené váhami. Vyměníme čísla 6 za 7:

```
double vahyFZZ[][]=new double[velPop][7];
double vahyFSE[][]=new double[velPop][7];
double vahyFOV[][]=new double[velPop][7];
```

Obr. 40: Změna definice 2D polí vahyFZZ[][], vahyFSE[][] a vahyFOV[][]

Na řádcích 504-509 jsou definována 2D pole, ve kterých budou uloženy kódy nejlepších jedinců z populace:

```
int castNejJedinceI[][]=new int[tym[6]][investice[0].length];
```

Obr. 41: Definice 2D pole castNejJedincePER[][]

Na řádcích 512-517 jsou volání metody **ulozeniNejJedince**. K nim je zapotřebí přidat:

```
castNejJedinceI=ulozeniNejJedince
(velPop,castNejJedinceI,inicializaceI,poradiNejJedince,nejFT,FT);
```

Obr. 42: Volání metody ulozeniNejJedince

Na řádcích 550-555 jsou definována 3D pole, ve kterých jsou uloženy nejlepší kódy jedinců populace:

```
int nejJedinciI[][][]=new int[tym[6]][investice[0].length][pocetIteraci+1];
```

Obr. 43: Definice 3D pole nejJedinciI

Na řádcích 557-586 jsou definována ukládání kódů nejlepších jedinců. Za řádek 586 přidáme následující zdrojový kód:

```
for (int i=0;i<tym[6];i++) {
    for (int j=0;j<investice[0].length;j++) {
        nejJedinciI[i][j][0]=castNejJedinceI[i][j];
    }
}
```

Obr. 44: Definice ukládání kódů nejlepších jedinců

Na řádcích 607-612 jsou definována 3D pole, ve kterých bude uložená mutovaná populace. Přidáme,:

```
int mutaceI[][][]=new int[tym[6]][investice[0].length][velPop];
```

Obr. 45: Definice pole mutaceI

Na řádcích 613-618 jsou definována 3D pole, ve kterých bude uložená selektovaná populace:

```
int selektovanaPopulaceI[][][]=new int[tym[6]][investice[0].length][velPop];
```

Obr. 46: Definice 3D pole selektovanaPopulaceI

Na řádcích 693-698 jsou volání metody **selektovaniPopulace**. K nim musíme přidat následující:

```
selektovanaPopulaceI=selektovaniPopulace(velPop,investice,tym[6],selektovanaPopul
aceI,inicializaceI,postupujici,mutaceI,pc);
```

Obr. 47: Volání metody selektovaniPopulaceI

Na řádcích 722-727 je volána metoda **mutovaniPopulace**, ke které musíme přidat následující:

```
mutaceI=mutovaniPopulace(velPop, investice, mutaceI, selektovanaPopulaceI, tym,
Pm);
```

Obr. 48: Voláni metody mutovaniPopulaceI

Na řádcích 752-757 je volána metoda **souctyZnamekSkupina**. K těmto voláním přidáme:

```
SZZZ7=souctyZnamekSkupina(SZZZ7,mutaceI,velPop);
```

Obr. 49: Volání metody souctyZnamekSkupina

Na řádcích 785-790 je volána metoda **urceniFunkceFZZ**. K těmto voláním musíme přidat:

```
FZZ=urceniFunkceFZZ(FZZ,SZZZ7,mutaceI,investice,POV[6],pozadovaneI,velPop,6);
```

Obr. 50: Volání metody urceniFunkceFZZ

Na řádcích 799-840 jsou nulována pomocná pole. Za řádek 840 přidáme následující zdrojový kód:

```
for (int i=0;i<pomocneI.length;i++) {
                for (int j=0;j<pomocneI[0].length;j++) {
                    for (int k=0;k<pomocneI[0][0].length;k++) {
                        pomocneI[i][j][k]=0;
                    }
                }
}
```

Obr. 51: Nulování pomocného pole pomocneI

Na řádcích 841-846 je volána metoda **plneniPomocPoli**. Na poslední volání metody doplníme:

```
pomocneI=plneniPomocPoli(velPop,mutaceI,POV[6],pomocneI,pozadovaneI);
```

Obr. 52: Volání metody plneniPomocPoli

Na řádcích 868-897 jsou nulována pole SPPZ1-6. Za řádek 897 přidáme následující zdrojový kód:

```
for (int i=0;i<SPPZ7.length;i++) {
                for (int j=0;j<SPPZ7[0].length;j++) {
                    SPPZ7[i][j]=0;
                }
            }
```

Obr. 53: Nulování pole SPPZZ7

Na řádcích 899-904 je volána metoda **skutPocetZnamek**. Přidáme následující volání:

```
SPPZ7=skutPocetZnamek(velPop,mutaceI,POV[6],pomocneI,SPPZ7);
```

Obr. 54: Volání metody skutPocetZnamek

Na řádcích 927-956 jsou nulována pole, která slouží ke stanovení součtu prvků náhradních polí v řadách. Za řádek 956 přidáme následující zdrojový kód:

```
for (int i=0;i<sumRadaI.length;i++) {
                for (int j=0;j<sumRadaI[0].length;j++) {
                    sumRadaI[i][j]=0;
                }
            }
```

Obr. 55: Nulování pole sumRadaI

Na řádcích 957-962 je volána metoda **souctyPrvkuNahPole**. Za řádek 962 přidáme volání:

```
sumRadaI=souctyPrvkuNahPole(velPop,pomocneI,sumRadaI);
```

Obr. 56: Volání metody souctyPrvkuNahPole

Na řádcích 990-995 je volána metoda **prumHodnotaSoucet**. Za poslední volání přidáme:

```
stredHodn=prumHodnotaSoucet(sumRadaI,velPop,pomocneI,stredHodn,pocetZamI,6);
```

Obr. 57: Volání metody prumHodnotaSoucet

Na řádcích 1008-1013 je volána metoda **souctyOdchylekOdStredHodnot**. Přidáme následující volání metody:

```
sumDelta=souctyOdchylekOdStredHodnot(velPop,pomocneI,sumRadaI,stredHodn,sumDelta,
6);
```

Obr. 58: Volání metody souctyOdchylekOdStredHodnot

Na řádcích 1026-1031 je volána metoda **dilciFitnessFSE** k určení dílčí fitness funkce FSE.

```
FSE=dilciFitnessFSE(velPop,pomocneI,vaha1,vaha2,sumDelta,PPZ7,SPPZ7,FSE,6);
```

Obr. 59: Volání metody dilciFitnessFSE

Na řádcích 1040-1069 jsou nulována pole. Za řádek 1069 přidáme následující zdrojový kód:

```
for (int i=0;i<prumVlastnostI.length;i++) {
                for (int j=0;j<prumVlastnostI[0].length;j++) {
                    prumVlastnostI[i][j]=0;
                }
            }
```

Obr. 60: Nulování přidaného pole

Na řádcích 1070-1075 je volána metoda **prumVlastnostFOV**. Přidáme následující volání:

```
prumVlastnostI=prumVlastnostFOV(velPop, POV, mutaceI, prumVlastnostI,6);
```

Obr. 61: Volání metody prumVlastnostFOV

Na následujících řádcích 1103-1108 je volána metoda ke stanovení dílčí fitness funkce FOV. Přidáme volání:

```
FOV=dilciFitnessFOV(velPop,POV,FOV,prumVlastnostI,pozadovaneI,6);
```

Obr. 62: Volání dílčí fitness funkce FOV

Na řádcích 1179-1184 je volána metoda **ulozeniNejJedince**. Za řádek 1184 přidáme následující zdrojový kód:

```
  castNejJedinceI=ulozeniNejJedince
(velPop,castNejJedinceI,mutaceI,poradiNejJedince,nejFT,FT);
```

Obr. 63: Volání metody ulozeniNejJedince

Na následujících řádcích 1186-1215 jsou ukládáni nejlepší jedinci z 2D polí do nových 3D polí (postihnutí celého výpočtu). Za řádek 1215 přidáme následující zdrojový kód:

```
for (int i=0;i<tym[6];i++) {
            for (int j=0;j<investice[0].length;j++) {
                nejJedinciI[i][j][pc+1]=castNejJedinceI[i][j];
                }
            }
```

Obr. 64: Kopírování nejlepších jedinců do jiných polí

Na řádcích 1245-1386 jsou do textových souborů ukládány kódy nejlepších jedinců z každé iterace (včetně počáteční inicializace populace). Za řádek 1386 přidáme následující zdrojový kód:

```
//* ulozeni nejlepsich investicnich pracovniku
File nejI = new File("C:\\CESTA\\nejInvestice.txt");
FileWriter in = new FileWriter(nejPer);
for (int i=0;i<pocetIteraci+1;i++) {
    for (int k=0;k<tym[6];k++) {
        for (int j=0;j<personal[0].length;j++) {
            in.write(String.valueOf(nejJedinciPER[k][j][i]));
            in.write("\t");
        }
        in.write(separator);
    }
    in.write(separator);
}
if (povyp==povy-1) {
in.close();
}
```

Obr. 65: Uložení nejlepších pracovníků ze skupiny Investice

6.3.9 Nastavení správných cest

Při práci s programem (ať už se výše uvedené úpravy provedou nebo ne) je důležité nastavit si správné cesty pro ukládání výsledných dat z výpočtů podle toho, kam je chceme na daném počítači ukládat. Týká se řádků 59, 61, 63, 1245, 1292, 1308, 1324, 1340, 1356 a 1372. Pokud si přidáme daší skupiny pracovníků, je důležité správnou cestu nastavit i pro ně.

Upozornění: pro oddělení různých úrovní adresářů nelze použít klasické jedno lomítko „\“, ale dvě lomítka „\\“, aby to byl korektní zápis cesty pro jazyk Java, viz následující obrázek. Stejně tak je vhodné nepoužívat v cestách a názvech souborů mezery a diakritiku.

```
File PFS = new File("C:\CESTA\prumernaFitnessStatistika.txt");
File PFS = new File("C:\\CESTA\\prumernaFitnessStatistika.txt");
```

Obr. 66: První řádek ukazuje špatný zápis, druhý řádek ukazuje správný zápis

7 KONKRÉTNÍ OPTIMALIZACE

7.1 ÚVOD

Výběr vhodného pracovního týmu byl proveden ve stavební firmě, která plánovala sestavit tým osob pro realizaci nového projektu výstavby obytných domů. V této firmě jsou kmenoví zaměstnanci členěni do následujících skupin (řazeno abecedně).

Tab. 3: Seznam organizačních skupin ve firmě včetně počtu osob

Název skupiny	**Počet osob ve skupině**
Investice	1
Logistika	3
Obchod	3
Personalistika	2
Projekce	6
Recepce	3
Ředitel	2
Sekretariát	2
Správce	2
Stavbyvedoucí	7
Účetní	3
CELKEM	34

Tabulka neuvažuje se skupinou řadových dělníků a jiných najímaných osob na konkrétní úkoly, protože ti se často mění a nelze s nimi mnohdy počítat ani na polovinu projektu. V rámci sestavování týmu jde o tým lidí, kteří se podílejí na plánování, projektování, řízení, ekonomické činnosti, obchodní činnosti a dohledem nad realizací na stavbě.

7.2 ZADÁNÍ

Bylo požadováno sestavit tým osob v následujícím složení:

Tab. 4: Požadavek na tým z hlediska počtu lidí ve skupinách

Název organizační skupiny	**Počet osob ve skupině**
Logistika	2
Obchod	2
Personalistika	1
Projekce	4
Stavbyvedoucí	4
Účetní	1
CELKEM	14

Počet možných variant hledaného týmu stanovíme následujícím vztahem pomocí kombinací bez opakování:

$$\text{Počet variant} = \binom{3}{2} \cdot \binom{3}{2} \cdot \binom{2}{1} \cdot \binom{6}{4} \cdot \binom{7}{4} \cdot \binom{3}{1} = 3 \cdot 3 \cdot 2 \cdot 15 \cdot 35 \cdot 3 = 28.350$$

7.3 ZAKÓDOVÁNÍ

7.3.1 Kódovací tabulka

Pro účely numerického zpracování řešené úlohy byla sestavena následující tabulka s číselným ohodnocením všech sledovaných vlastností. Tato tabulka předepisuje pořadí jednotlivých hodnocených vlastností a současně obsahuje kódovací klíč, který slovní vyjádření každého možného stavu jednotlivých vlastností převádí na jednoznačnou číselnou interpretaci.

Tab. 5: Souhrnná hodnotící (kódovací) tabulka pro každého zaměstnance

Vlastnosti	**Poř. číslo**	**Název klíčových vlastností**	**Ohodno-cení**	**Rozsah hodnocení**
Odborné	**1**	Angličtina		1 = plynně slovem i písmem 2 = dobrá znalost slova i písma 3 = průměrná znalost slova i písma 4 = několik základních frází 5 = neumí
	2	Znalosti PC		1 = výborná znalost 2 = dobrá znalost 3 = průměrná znalost 4 = základní úkony 5 = neumí
	3	Odbornost v oboru		1 = výborná znalost 2 = dobrá znalost 3 = průměrná znalost 4 = uspokojivá znalost 5 = nedostatečná znalost
	4	Řidičský průkaz sk. B		1 = má 2 = nemá ŘP nebo o něj přišel/a
Osobní	**5**	Primární týmová role dle Belbina		1 = Inovátor 2 = Vyhledávač zdrojů 3 = Koordinátor 4 = Usměrňovač 5 = Monitor vyhodnocovač 6 = Týmový pracovník 7 = Realizátor 8 = Kompletovač finišer
	6	Sekundární týmová role dle Belbina		1 = Inovátor 2 = Vyhledávač zdrojů 3 = Koordinátor 4 = Usměrňovač 5 = Monitor vyhodnocovač 6 = Týmový pracovník 7 = Realizátor 8 = Kompletovač finišer
	7	Subjektivní hodnocení člověka jako celku vedením firmy		1 = slušný, pohodový, iniciativní a bezkonfliktní člověk 2 = slušný a iniciativní člověk, někdy konfliktní 3 = člověk průměrných osobních kvalit 4 = vážnější výhrady k některým osobním vlastnostem 5 = konfliktní, neiniciativní a problémový člověk

7.3.2 Způsob hodnocení

Zaměstnanci potřební pro vytvoření požadovaného týmu se nacházeli v organizačních skupinách Logistika, Obchod, Personalistika, Projekce, Stavbyvedoucí a Účetní. Na každého zaměstnance připadala jedna hodnotící tabulka a takto byly postupně vyplněny hodnotící tabulky pro všechny zaměstnance nacházející se v těchto skupinách. Tímto vznikla dostatečně velká databáze jako vstup do vytvořeného optimalizačního programu. Odborné znalosti (poř. č. 1-4) a subjektivní hodnocení člověka jako celku (poř. č. 7) byly u každého člověka ohodnoceny jeho nadřízeným. Osobní vlastnosti, respektive týmové role dle Belbina (poř. č. 5-6), byly zjištěny na základě vypracování Belbinova testu (viz příloha č. 1). Belbinův test neobsahuje 9. roli – Specialistu. Pro účely navrženého algoritmu to nijak nevadí, jelikož speciální schopnosti jednotlivých členů týmu postihuje vlastním způsobem kódování jejich schopností.

Belbinovým testem rovněž prošli všichni zaměstnanci a každý si vyplnil svůj vlastní test. Dostali jej v papírové podobě a k jeho vyplnění měli dostatek času, tzn. nebyly uplatňovány žádné restrikce pro zaměstnance, kteří test vyplňovali déle, než ostatní. Vzhledem k většímu množství hodnocených osob bylo testování rozděleno do několila etap. Žádný zaměstnanec nikdy dříve o Belbinově testu neslyšel a neznal jej. Z důvodu zachování objektivity hodnocení navíc zaměstnanci nedostali do ruky výslednou tabulku, ve které se jednotlivé body sčítají a vyplývají z ní jejich týmové role. Tyto součty byly provedeny následně až po získání všech vyplněných testů od všech zaměstnanců. Jednotliví zaměstnanci byli v několika následujících dnech individuálně seznámeni se svými výsledky a vždy proběhla krátká diskuze, zda si myslí, že výsledky Belbinova testu odpovídají tomu, co si o sobě sami myslí. V naprosté většině případů došlo ke shodě a u zbytku zaměstnanců se objevil spíše údiv nad tím, že někdo pojmenovává jejich vlastnosti, o kterých nikdy předtím sami neuvažovali a ani se je nesnažili pojmenovat. Do hodnotící tabulky byly zaneseny vždy jen dvě nejdominantnější týmové role každého pracovníka. Další týmové role, které se případně v malé míře u zaměstnanců v testech objevily, jsou minoritní a tudíž pro tuto optimalizaci nezajímavé.

7.3.3 Výsledné hodnocení a zakódování

Výsledky hodnocení všech zaměstnanců jsou zobrazeny přehledným způsobem v následující tabulce. V posledním sloupci je uvedený výsledný kód, pod kterým vstupují všichni zkoumaní zaměstnanci do optimalizačního programu.

Tab. 6: Výsledné zakódování všech zaměstnanců

Název skupiny dle organizačního členění	Jméno a příjmení	1: Angličtina	2: Znalosti PC	3: Odbornost v oboru	4: Řidičský průkaz sk. B	5: Primární týmová role dle Belbina	6: Sekundární týmová role dle Belbina	7: Subjektivní hodnocení	Výsledný kód
Logistika	Milan Jandák	2	1	1	1	1	3	2	**2111132**
	Tomáš Myslín	4	4	3	1	8	2	3	**4431823**
	Michal Toman	3	2	2	1	6	5	1	**3221651**
Obchod	Jiří Borovec	2	1	2	1	1	2	2	**2121122**
	Petr Štěpán	1	1	1	1	1	3	1	**1111131**
	Alena Malá	2	3	2	1	2	4	1	**2321241**
Perso-nali-stika	Iva Spurná	3	2	1	1	6	7	1	**3211671**
	Pavla Šplíchová	5	3	2	1	5	3	3	**5321533**
Projekce	Pavel Slovák	2	1	1	1	8	3	1	**2111831**
	Barbora Zubrová	3	2	2	2	2	6	1	**3222261**
	Jana Vesecká	3	3	3	1	5	6	3	**3331563**
	Miroslav Křeček	1	1	1	1	1	3	1	**1111131**
	Romana Švermová	2	1	2	1	8	4	2	**2121842**
	Kamila Nováková	3	2	1	2	8	6	2	**3212862**
Stavbyvedoucí	Kamil Černý	4	2	2	1	5	6	1	**4221561**
	Igor Stanovec	5	5	2	1	7	6	3	**5521763**
	Roman Podlužný	4	3	1	1	7	8	2	**4311782**
	Ivan Borovský	3	2	1	1	1	5	1	**3211151**
	Aleš Bodlák	2	2	1	1	4	3	3	**2211433**
	Martin Ondráček	3	3	2	1	7	4	2	**3321742**
	Libor Procházka	5	4	2	1	8	6	2	**5421862**
Účetní	Petr Kotouček	1	1	2	1	8	1	1	**1121811**
	Romana Dubová	2	2	2	2	2	6	2	**2222262**
	Anna Kočí	3	3	1	1	4	7	3	**3311473**

7.3.4 Přepis do zdrojového kódu

Výsledné kódování zaměstnanců je následně zaneseno do zdrojového kódu programu. Řádková lokalizace jednotlivých důležitých úseků uváděného zdrojového kódu je podrobně popsána v kapitole 6.2.8. Následně uváděné řádky se zabývají nezbytně nutnými numerickými úpravami ve zdrojovém kódu, aby mohl výpočet proběhnout. Nejdříve je nutné naplnit databázi programu reálnými kódy všech zaměstnanců.

```
            //*** 1. DATABÁZE ZAMĚSTNANCŮ ***
            //zaměstnanci jsou rozdělení do pracovních skupin
            //1=Logistika, 2=Obchod, 3=Personalistika, 4=Projekce, 5=Stavbyvedoucí,
6=Účetní
            //* 1=LOGISTIKA *
            int logistika[][]={
                {2,1,1,1,1,3,2},
                {4,4,3,1,8,2,3},
                {3,2,2,1,6,5,1}
            };
            //* 2=OBCHOD*
            int obchod[][]={
                {2,1,2,1,1,2,2},
                {1,1,1,1,1,3,1},
                {2,3,2,1,2,4,1}
            };
            //* 3=PERSONALISTIKA *
            int personalistika[][]={
                {3,2,1,1,6,7,1},
                {5,3,2,1,5,3,3}
            };
            //* 4=PROJEKCE *
            int projekce[][]={
                {2,1,1,1,8,3,1},
                {3,2,2,2,2,6,1},
                {3,3,3,1,5,6,3},
                {1,1,1,1,1,3,1},
                {2,1,2,1,8,4,2},
                {3,2,1,2,8,6,2}
            };
            //* 5=STAVBYVEDOUCÍ *
            int stavbyvedouci[][]={
                {4,2,2,1,5,6,1},
                {5,5,2,1,7,6,3},
                {4,3,1,1,7,8,2},
                {3,2,1,1,1,5,1},
                {2,2,1,1,4,3,3},
                {3,3,2,1,7,4,2},
                {5,4,2,1,8,6,2}
            };
            //* 6=ÚČETNÍ *
            int ucetni[][]={
                {1,1,2,1,8,1,1},
                {2,2,2,2,2,6,2},
                {3,3,1,1,4,7,3}
            };
```

Obr. 67: Založení databáze zaměstnanců

Následně je zaveden počet požadovaných zaměstnanců z každé organizační skupiny, kteří se mají objevit ve výsledném týmu.

```
int tym[]={2,2,1,4,4,1};
```

Obr. 68: Definice počtu zaměstnanců z každé skupiny ve výsledném týmu

Určení počtu osobních vlastností vychází z tabulky č. 5. Uvedený počet znamená počet osobních vlastností v kódech zaměstnanců (počítano zprava). V požadovaném týmu byli všichni hodnoceni jednotným způsobem, proto je počet osobních vlastností stejný u každé organizační skupiny.

```
int POV[] = {3,3,3,3,3,3};
```

Obr. 69: Určení počtu osobních vlastností (zprava z kódu zaměstnance)

Dále se dle požadavků vedoucích pracovníků určily ideální známky všech sledovaných vlastností pro každou organizační skupinu zvlášť (na stavbyvedoucího jsou kladeny jiné požadavky, jako na účetního apod.). Optimalizační program se při výběru konkrétních zaměstnanců do výsledného týmu snaží co nejvíce těmto požadavkům přiblížit.

```
int pozadovaneL[]={3,2,1,1,8,3,2};
int pozadovaneO[]={1,1,1,1,1,2,1};
int pozadovanePE[]={3,2,2,1,6,3,1};
int pozadovanePR[]={2,1,1,1,8,4,2};
int pozadovaneS[]={3,2,2,1,7,5,2};
int pozadovaneU[]={3,2,1,1,8,7,1};
```

Obr. 70: Určení požadovaných známek u organizačních skupin

Pro detailnější možnost nastavení důležitosti a upřednostnění či potlačení vlivu jednotlivých vlastností v rámci každé organizační skupiny je možné nastavit následující procentuální váhy. Rozdělení vah v každém řádku musí v součtu dávat 100%. Např. pro stavbyvedoucího je řidičský průkaz mnohem cennější a pro práci potřebnější (neustále se pohybuje v terénu), než schopnosti práce s počítačem (administrativu za něj může dělat sekretářka). V popisované optimalizaci proto byl řidičský průkaz ohodnocen důležitostí 30% a schopnost práce na počítači pouze důležitostí 5%.

Tab. 7: Tabulka s určením procentuální váhy jednotlivých sledovaných parametrů

Název skupiny dle organizačního členění	1: Angličtina	2: Znalosti PC	3: Odbornost v oboru	4: Řidičský průkaz sk. B	5: Primární týmová role dle Belbina	6: Sekundární týmová role dle Belbina	7: Subjektivní hodnocení	Celkem
Logistika	10%	20%	30%	15%	10%	10%	5%	100%
Obchod	15%	20%	20%	20%	10%	10%	5%	100%
Personalistika	10%	30%	30%	5%	10%	10%	5%	100%
Projekce	5%	20%	35%	15%	10%	10%	5%	100%
Stavbyvedoucí	5%	5%	35%	30%	10%	10%	5%	100%
Účetní	10%	20%	40%	5%	10%	10%	5%	100%

```
double procentaL[] = {0.1, 0.2, 0.3, 0.15, 0.1, 0.1, 0.05};
double procentaO[] = {0.15, 0.2, 0.2, 0.2, 0.1, 0.1, 0.05};
double procentaPE[] = {0.1, 0.3, 0.3, 0.05, 0.1, 0.1, 0.05};
double procentaPR[] = {0.05, 0.2, 0.35, 0.15, 0.1, 0.1, 0.05};
double procentaS[] = {0.05, 0.05, 0.35, 0.3, 0.1, 0.1, 0.05};
double procentaU[] = {0.1, 0.2, 0.4, 0.05, 0.1, 0.1, 0.05};
```

Obr. 71: Přepis procentuální váhy jednotlivých parametrů do kódu programu

7.4 HLEDÁNÍ OPTIMÁLNÍCH PARAMETRŮ GA

Získání nejlepších výsledků v rozumně krátké době podstatnou měrou závisí na volbě optimálních vstupních parametrů, na základě kterých optimalizační program vypočítá řešení. Konkrétně se jedná o tyto vstupní parametry:

- velikost populace N,
- selekce tT,
- pravděpodobnost mutace Pm,
- počet iterací pc.

V kapitole 6.2.7 byly popsány obecně doporučované hodnoty pro nastavení genetického algoritmu. Množstvím cílených změn vstupních parametrů byla vyzkoušena citlivost optimalizačního programu na jejich změnu. Z tohoto důvodu byly veškeré výpočty s danými vstupními parametry vždy provedeny 50x a výsledky byly statisticky a graficky zpracovány. Z výsledků byl následně učiněn závěr a vzniklo doporučení pro další používání optimalizačního programu.

7.4.1 Citlivost na změnu velikosti populace

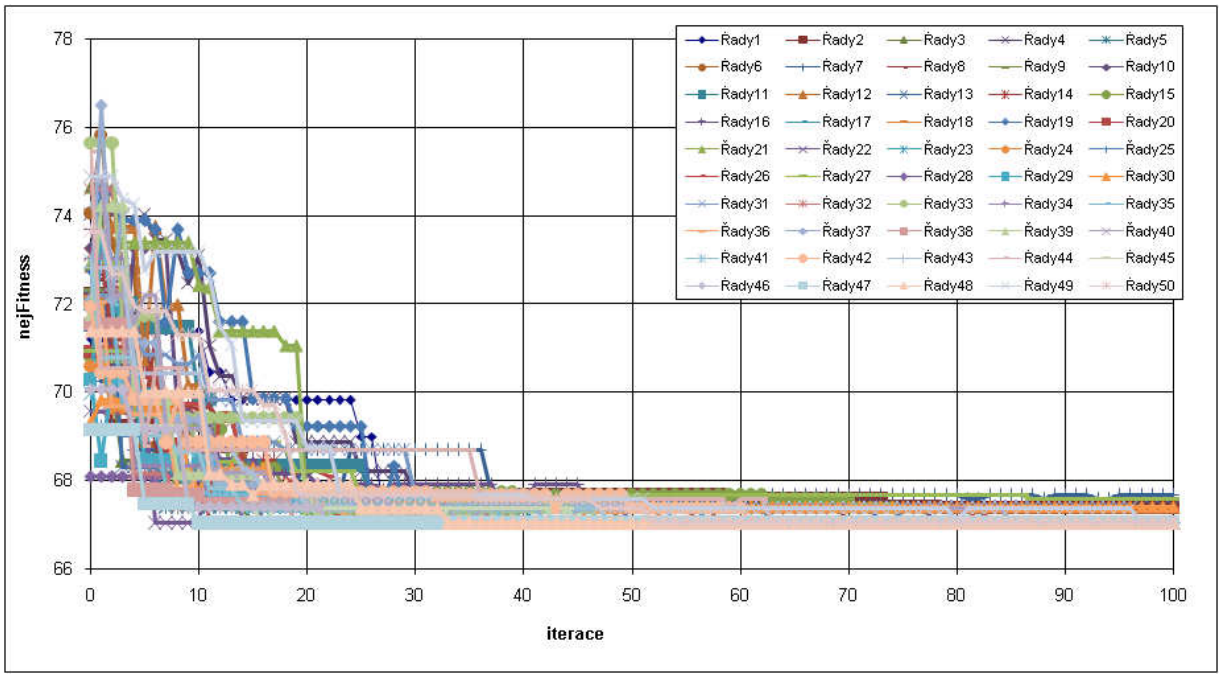
Pro účely sledování citlivosti na změnu velikosti populace bylo uvažováno s následujícími konstantními vstupními parametry:

- selekce tT = 2
- pravděpodobnost mutace Pm = 1/30
- počet iterací pc = 100

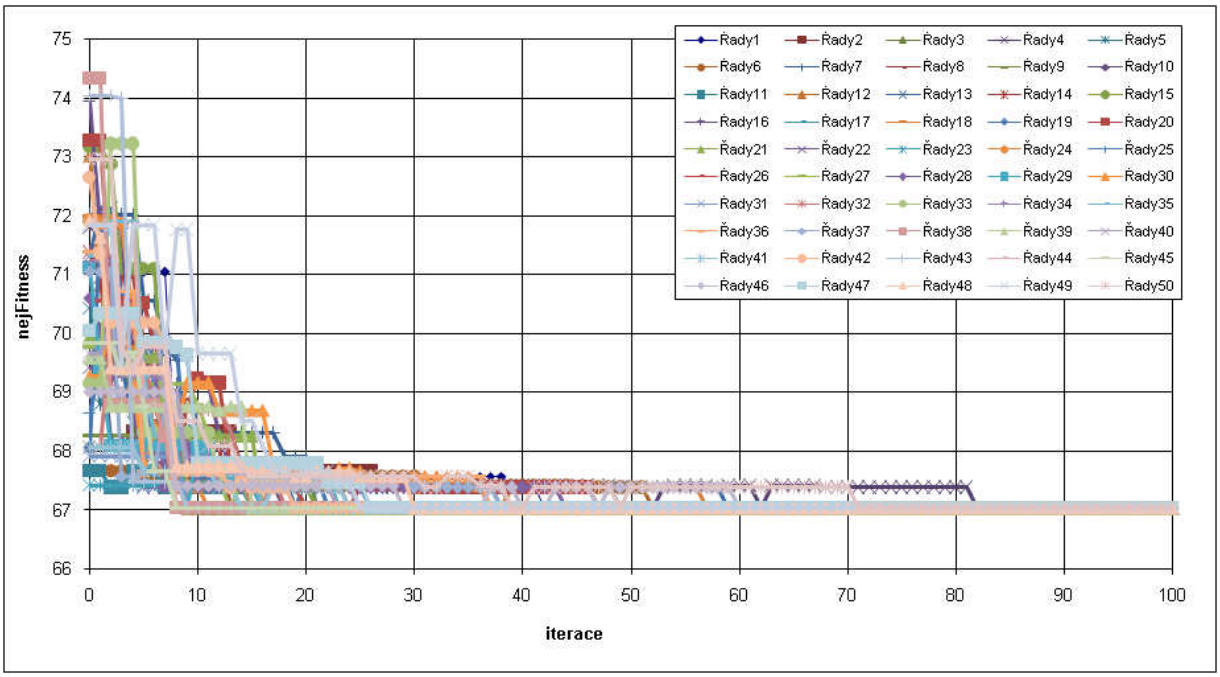
Pro jednotlivé výpočty byly následně měněny vstupní hodnoty v krocích:

- velikost populace N = 50, 100, 250, 500, 2000, 5000

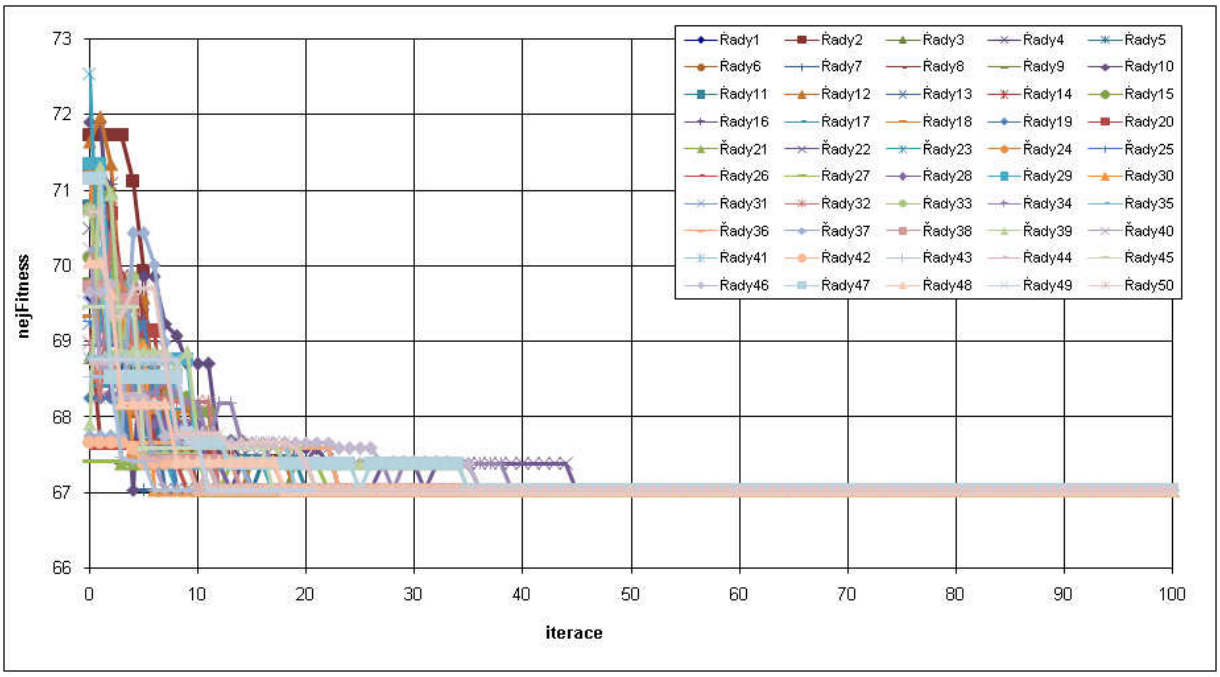
Následující grafy ukazují průběh hodnoty nejlepší fitness funkce. Výpočet byl kvůli statistickému posouzení proveden vždy 50x.



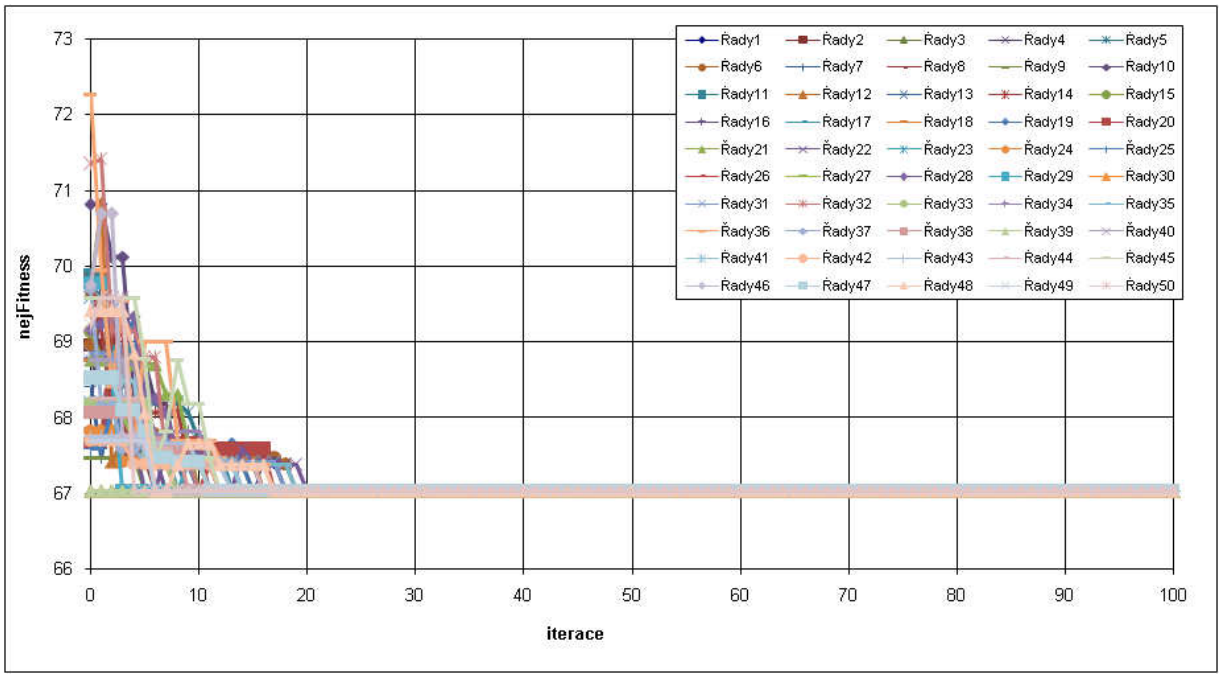
Graf 1: Vstup tT=2, Pm=1/30, pc=100, N=50



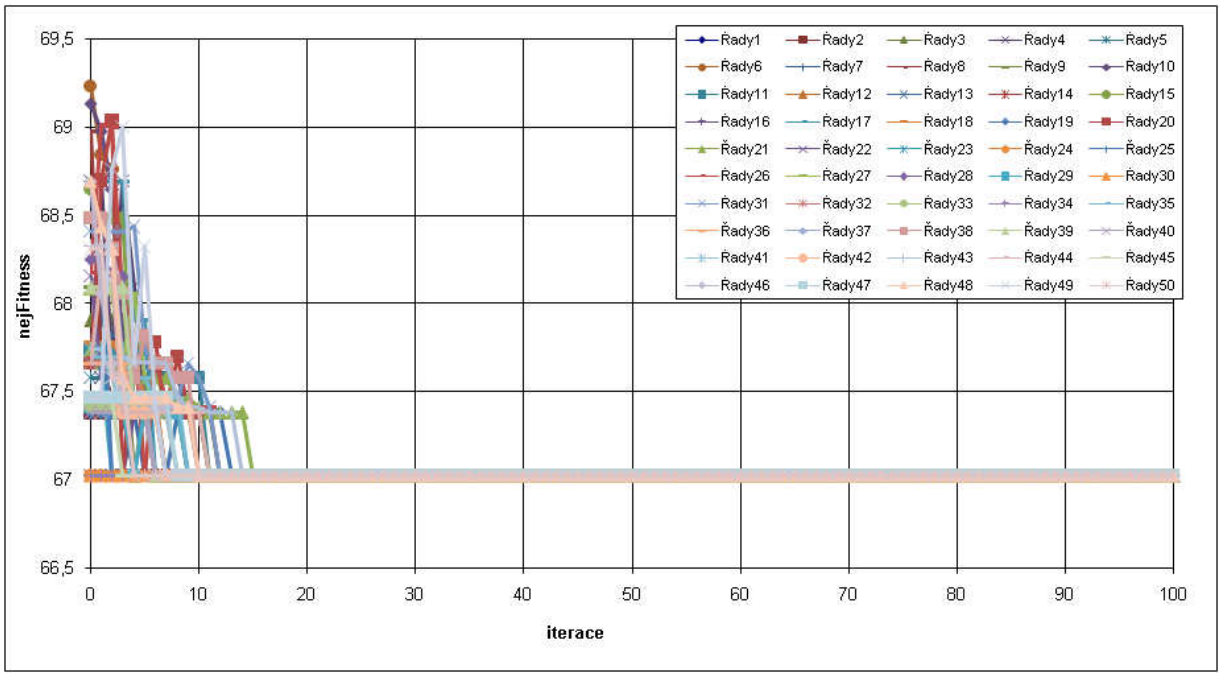
Graf 2: Vstup tT=2, Pm=1/30, pc=100, N=100



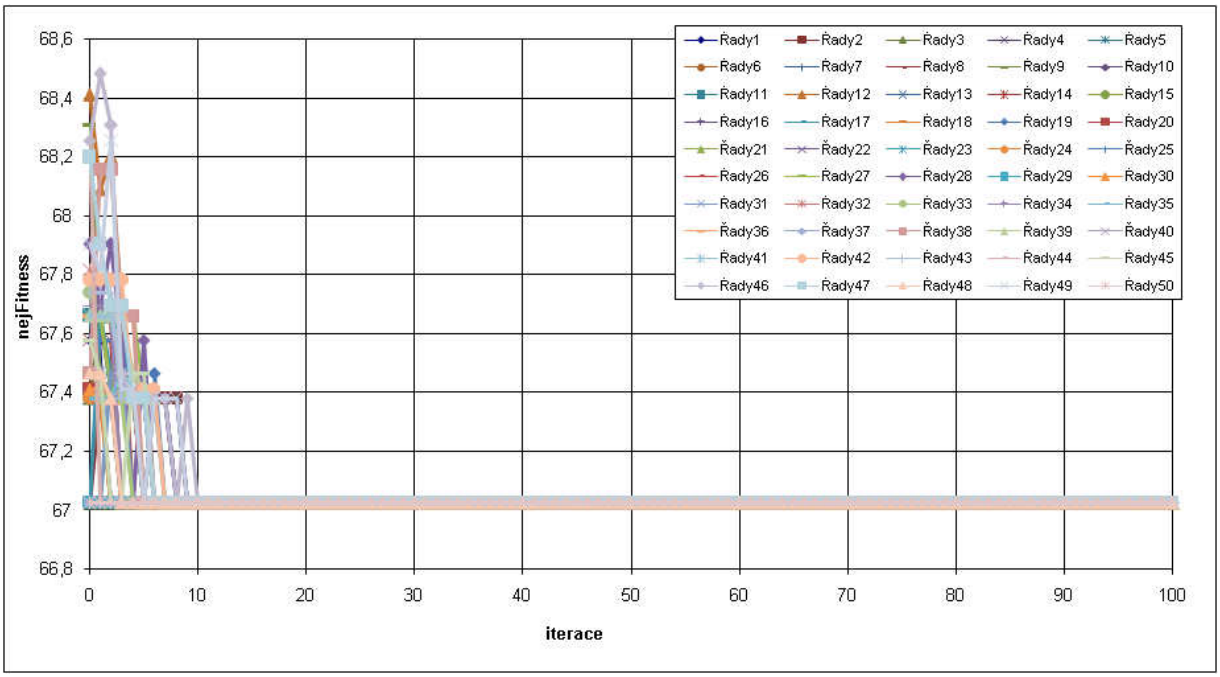
Graf 3: Vstup tT=2, Pm=1/30, pc=100, N=250



Graf 4: Vstup tT=2, Pm=1/30, pc=100, N=500



Graf 5: Vstup tT=2, Pm=1/30, pc=100, N=2000



Graf 6: Vstup tT=2, Pm=1/30, pc=100, N=5000

Prokázalo se, že zvýšení velikosti populace v dané úloze má za následek konvergenci již po několika málo cyklech. Z časového hlediska je ovšem přílišné zvětšování velikosti populace nežádoucí, protože významně prodlužuje dobu běhu optimalizačního algoritmu. Je vhodné ponechat velikost populace na hodnotě N=500.

7.4.2 Citlivost na změnu pravděpodobnosti mutace

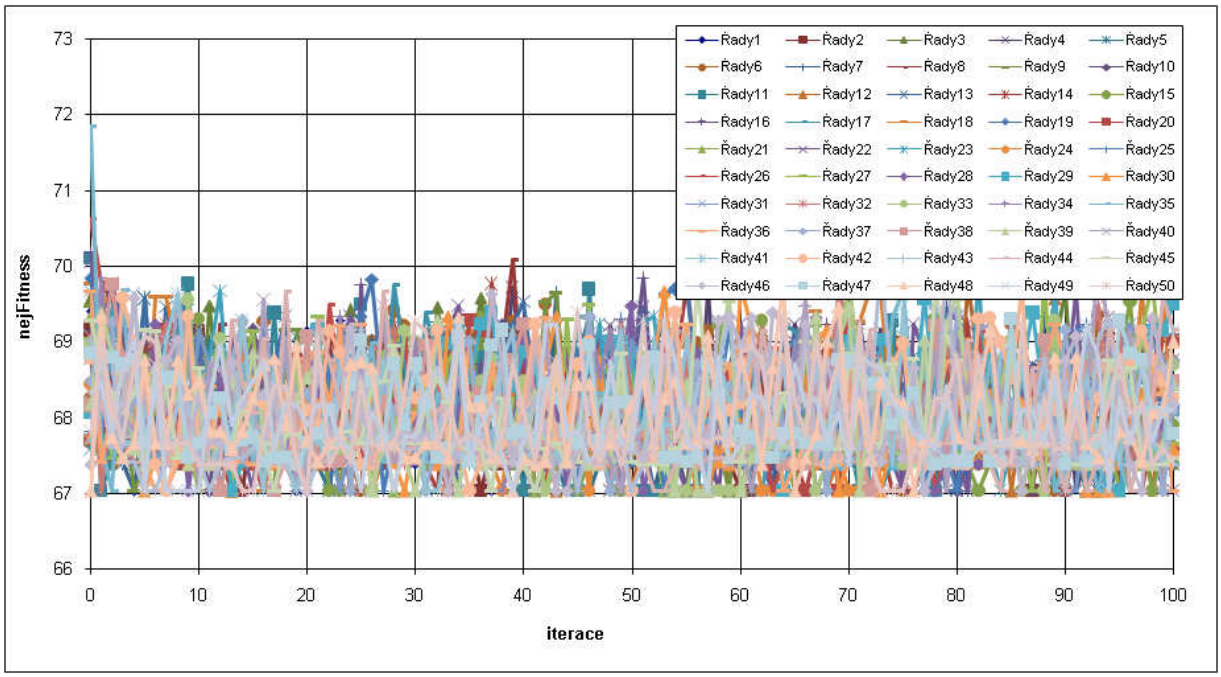
Pro účely sledování citlivosti na změnu pravděpodobnosti mutace bylo uvažováno s následujícími konstantními vstupními parametry:

- selekce tT = 2
- počet iterací pc = 100
- velikost populace N = 500

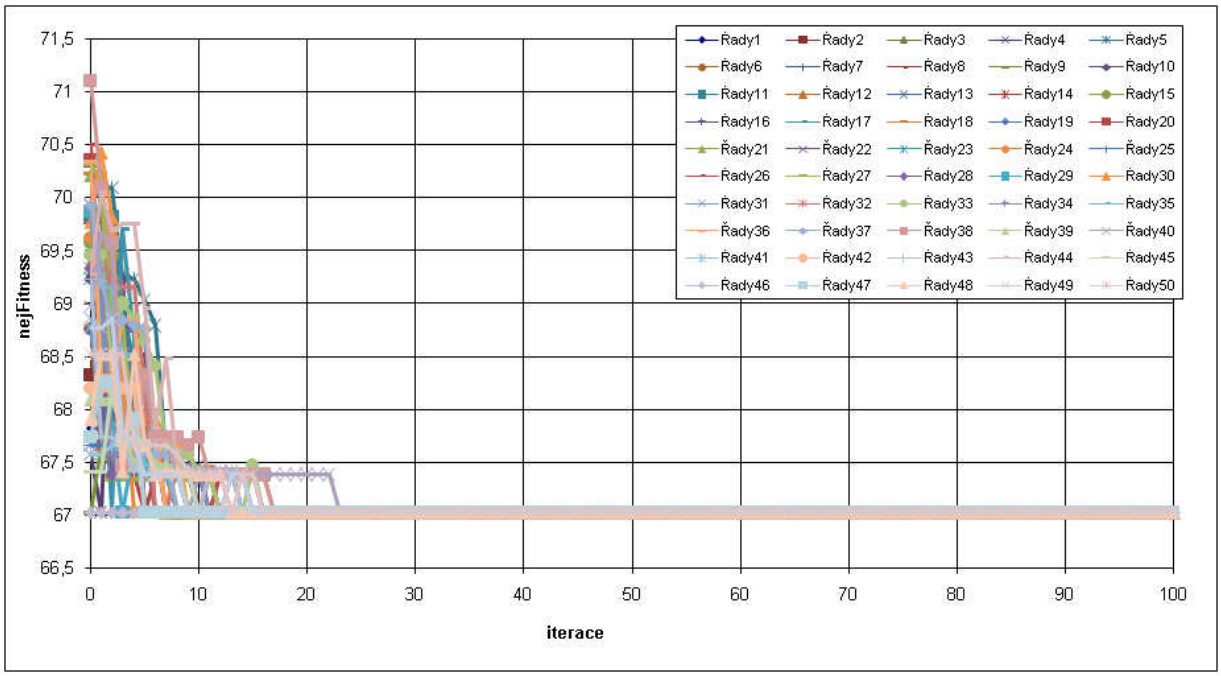
Pro jednotlivé výpočty byly následně měněny vstupní hodnoty v krocích:

- pravděpodobnost mutace Pm = 1/2, 1/10, 1/20, 1/30, 1/50, 1/70

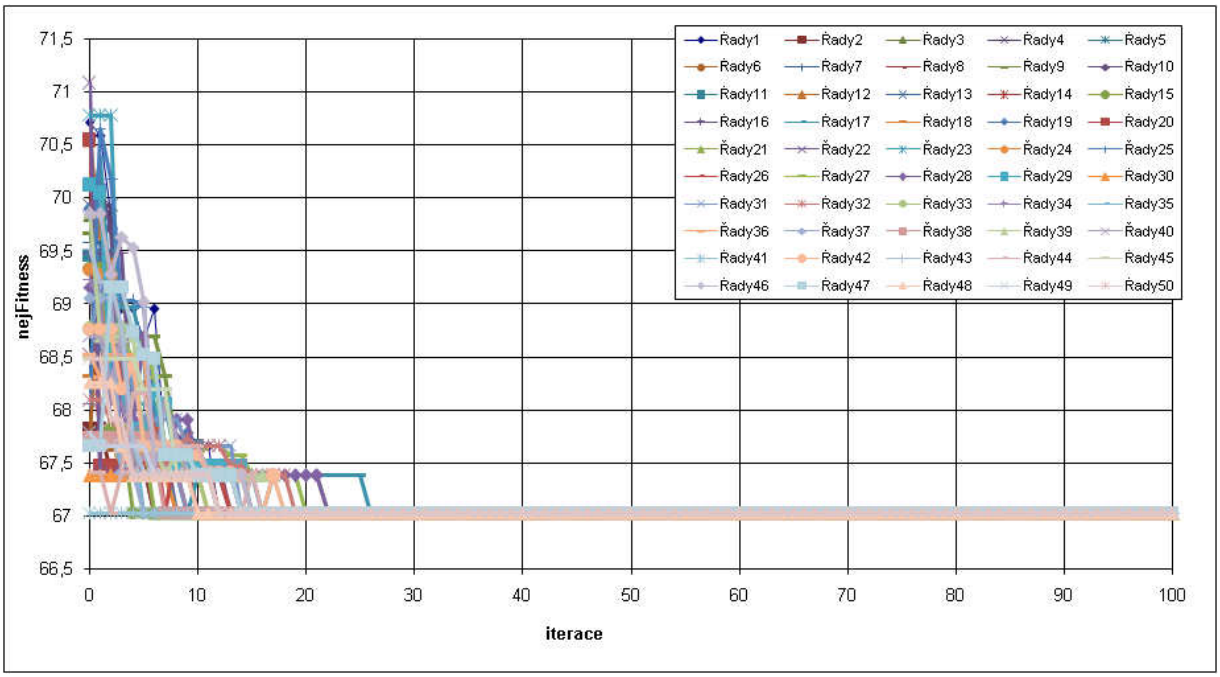
Následující grafy ukazují průběh hodnoty nejlepší fitness funkce. Výpočet byl kvůli statistickému posouzení proveden vždy 50x.



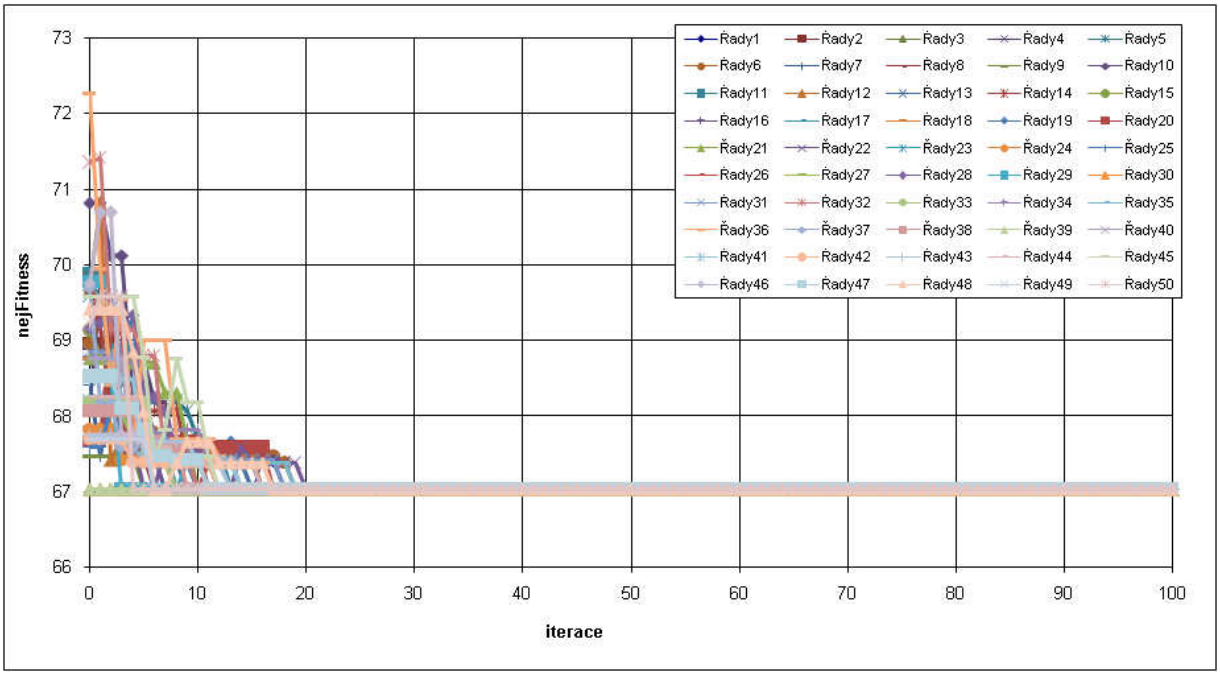
Graf 7: Vstup tT=2, pc=100, N=500, Pm=1/2



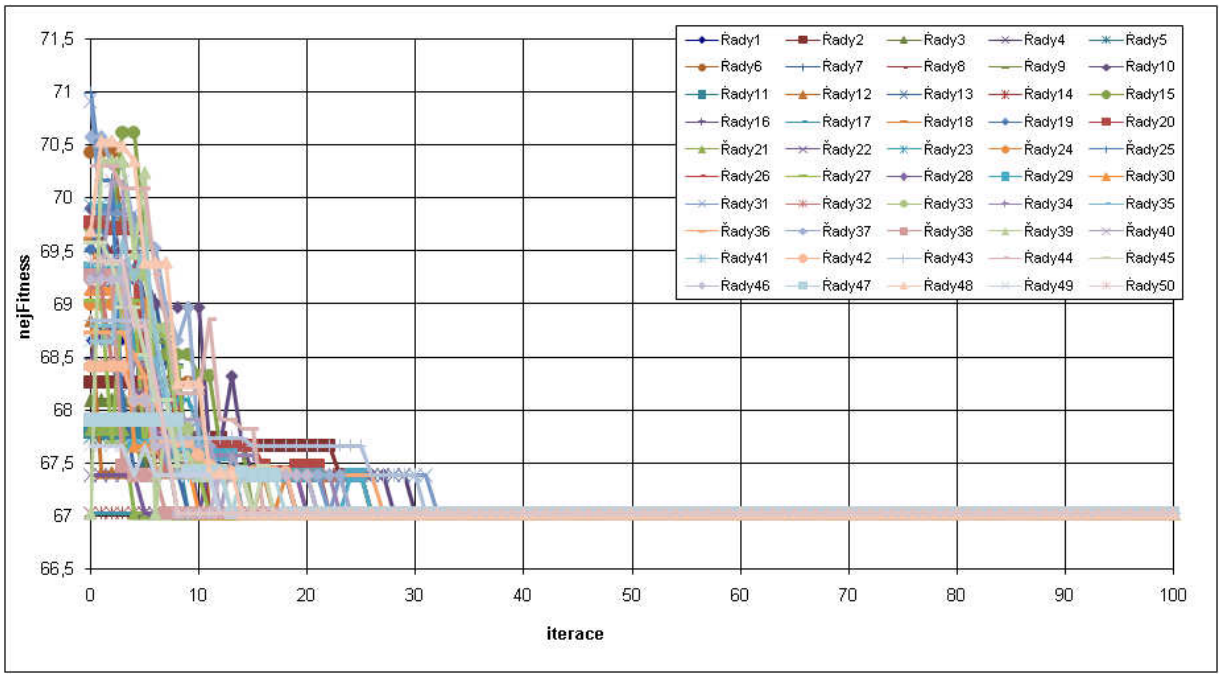
Graf 8: Vstup tT=2, pc=100, N=500, Pm=1/10



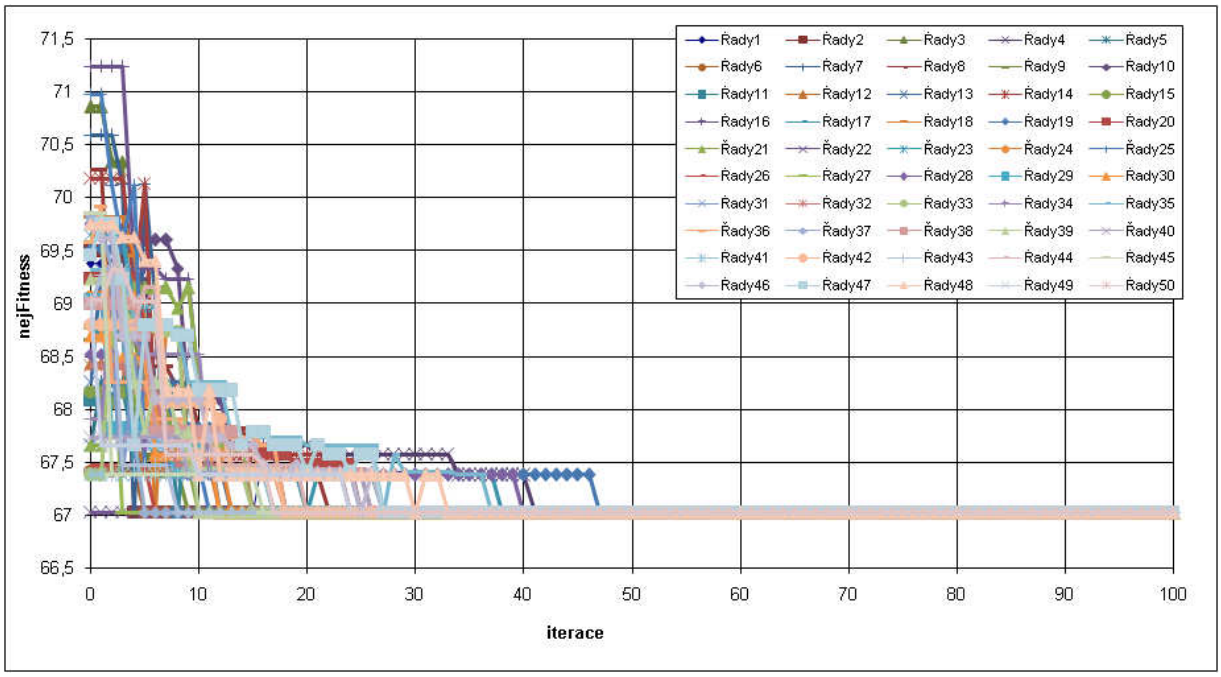
Graf 9: Vstup tT=2, pc=100, N=500, Pm=1/20



Graf 10: Vstup tT=2, pc=100, N=500, Pm=1/30



Graf 11: Vstup tT=2, pc=100, N=500, Pm=1/50



Graf 12: Vstup tT=2, pc=100, N=500, Pm=1/70

Prokázalo se, že zvýšení pravděpodobnosti mutace v dané úloze má za následek konvergenci již po několika málo cyklech. Toto ovšem platí jen do určitého okamžiku, protože při větší pravděpodobnosti, než Pm=1/10 se úloha stává nekonvergentní, jak dokazuje např. graf s pravděpodobností mutace Pm=1/2. Je vhodné ponechat pravděpodobnost mutace na hodnotě Pm=1/30.

7.4.3 Citlivost na změnu počtu jedinců v turnaji

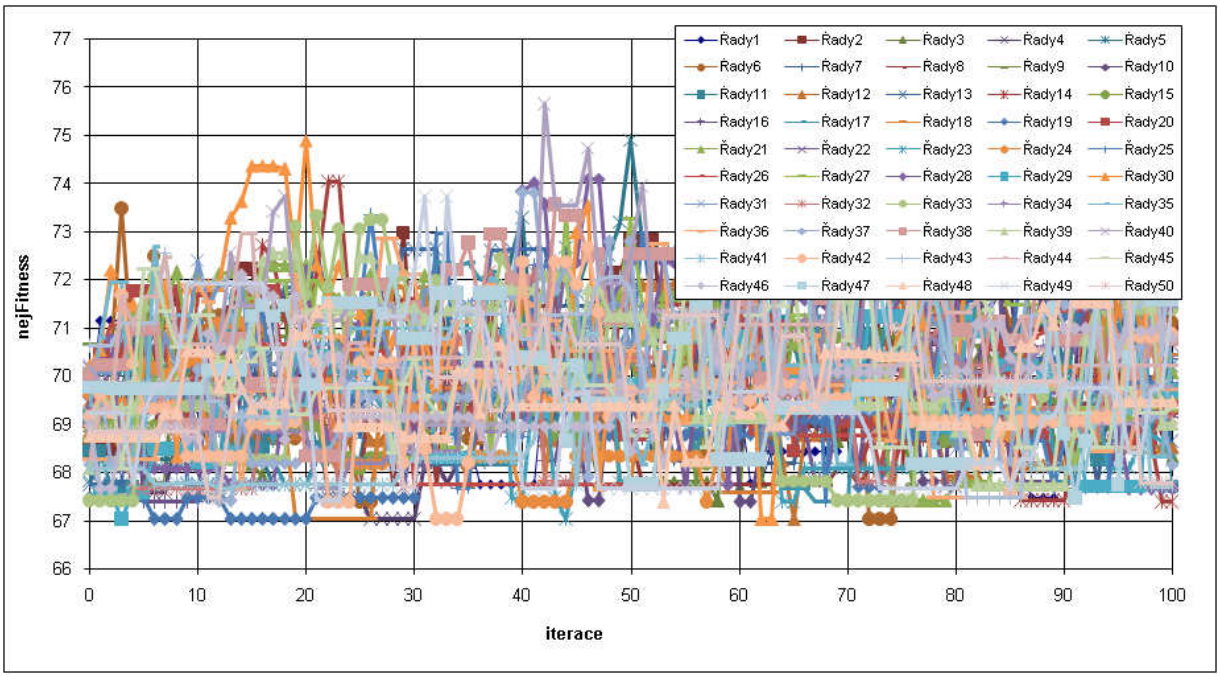
Pro účely sledování citlivosti na změnu počtu jedinců v turnaji bylo uvažováno s následujícími konstantními vstupními parametry:

- pravděpodobnost mutace Pm = 1/30
- počet iterací pc = 100
- velikost populace N = 500

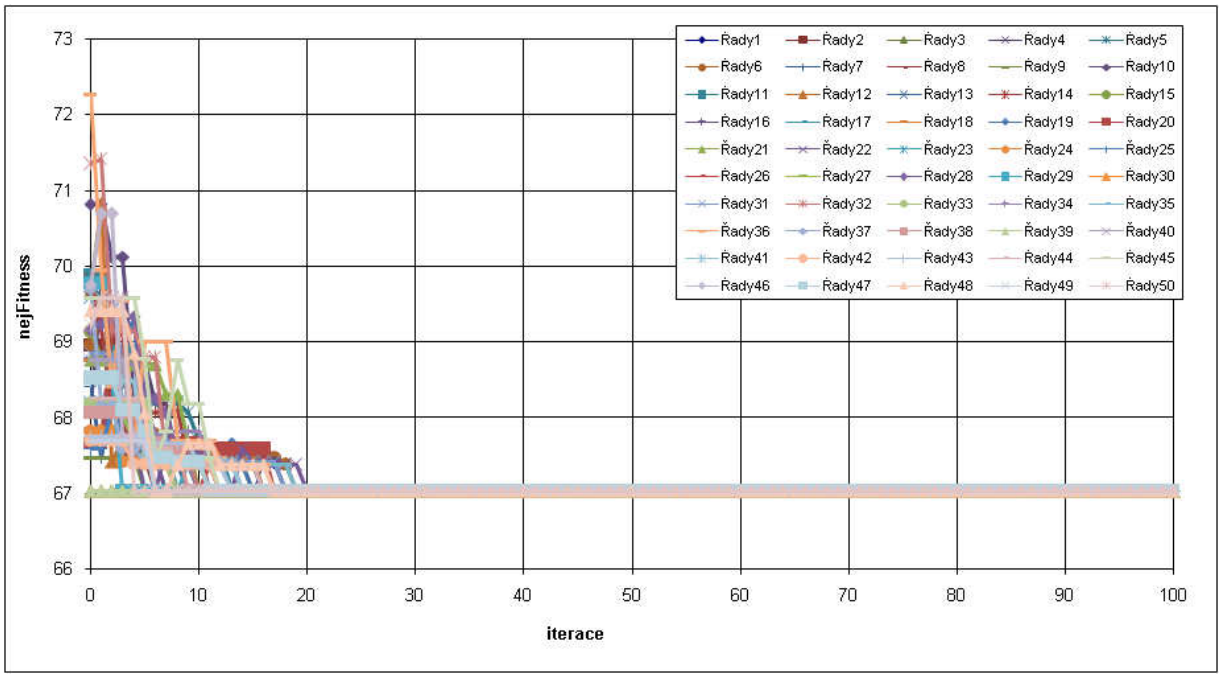
Pro jednotlivé výpočty byly následně měněny vstupní hodnoty v krocích:

- selekce tT = 1, 2, 3, 4, 8, 16

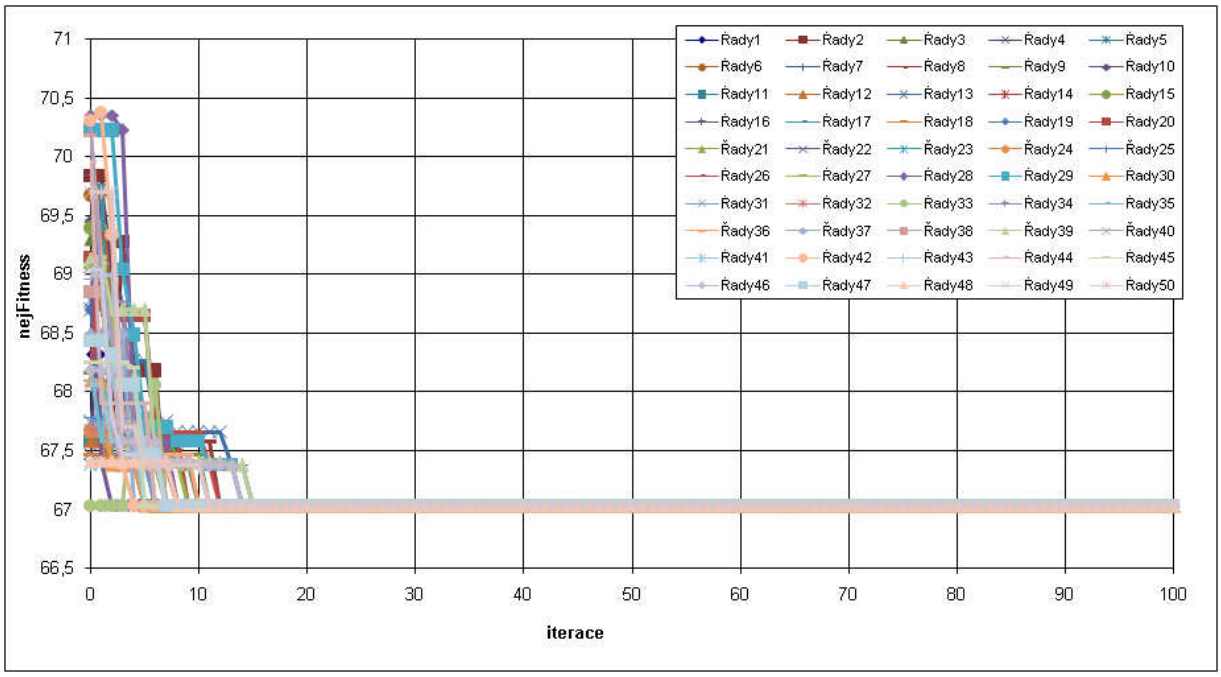
Následující grafy ukazují průběh hodnoty nejlepší fitness funkce. Výpočet byl kvůli statistickému posouzení proveden vždy 50x.



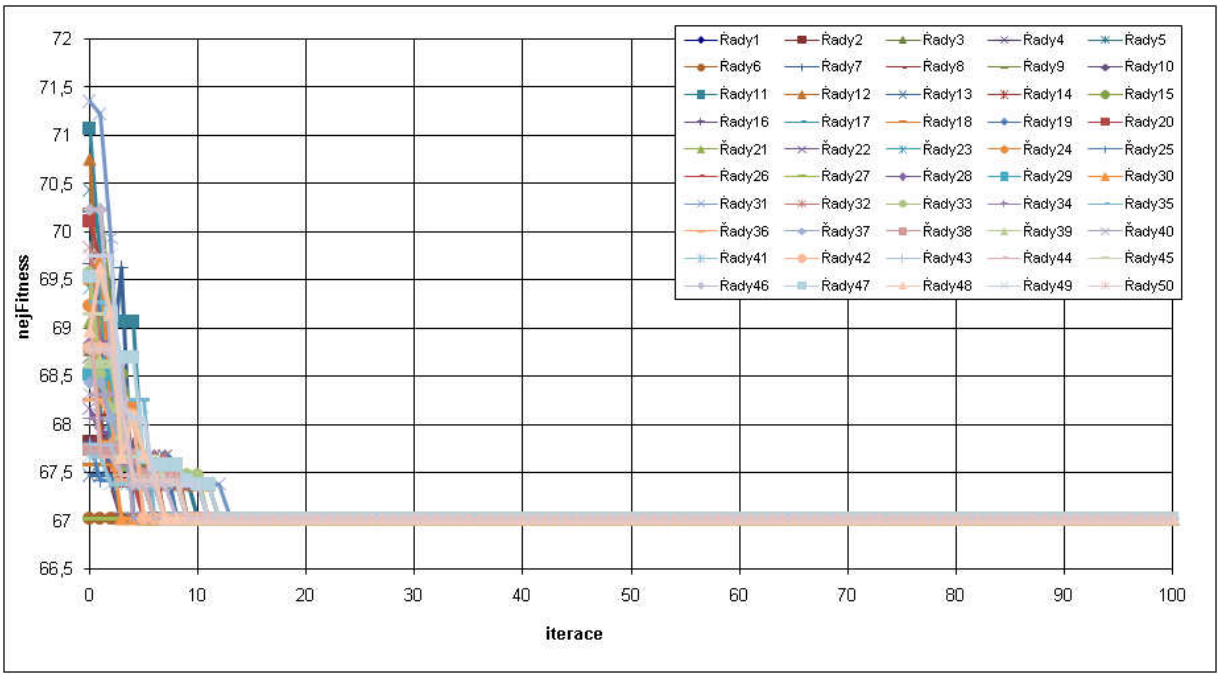
Graf 13: Vstup Pm=1/30, pc=100, N=500, tT=1



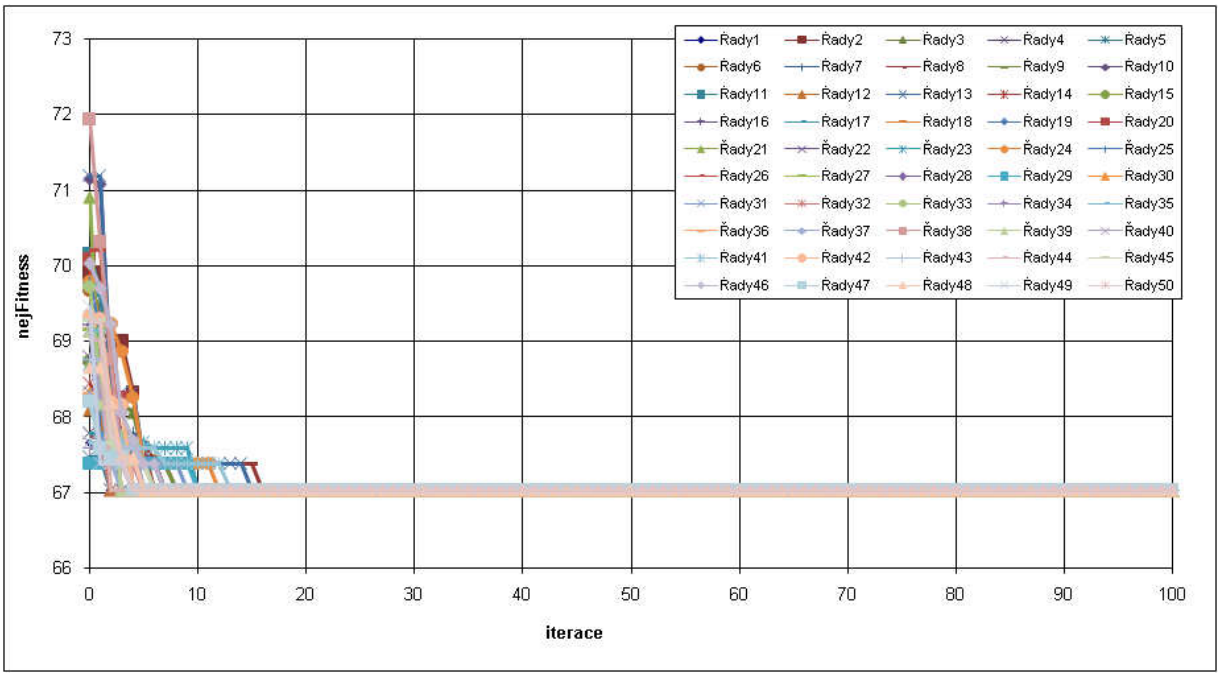
Graf 14: Vstup Pm=1/30, pc=100, N=500, tT=2



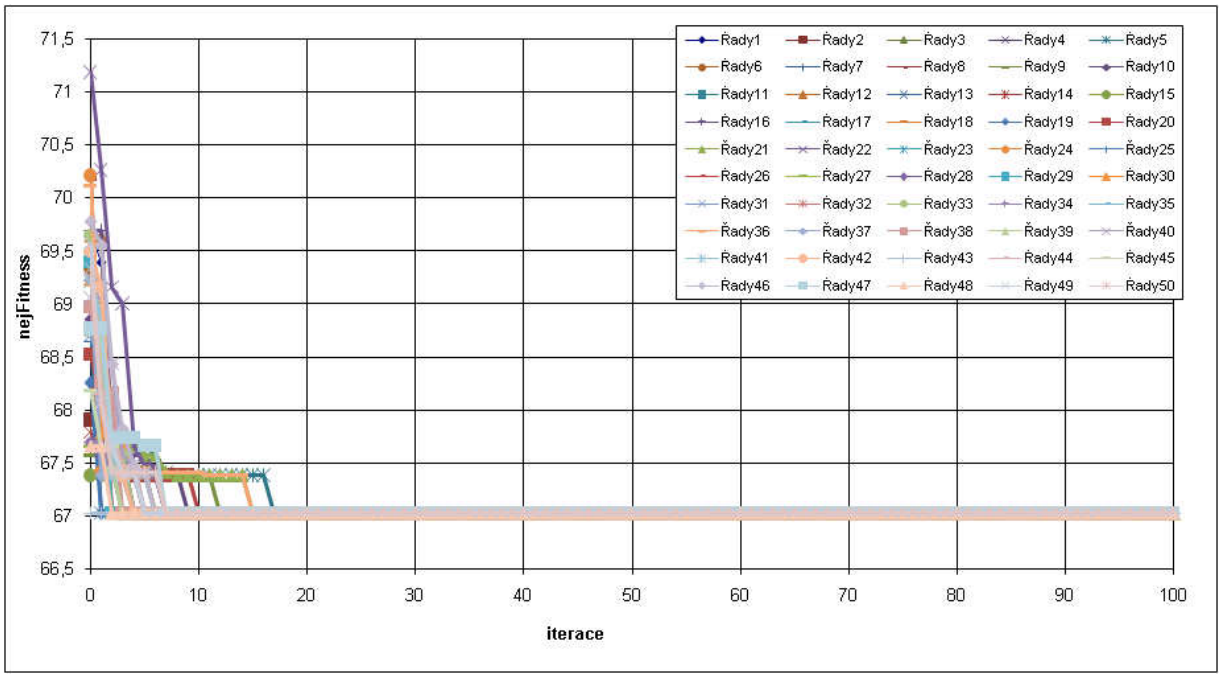
Graf 15: Vstup Pm=1/30, pc=100, N=500, tT=3



Graf 16: Vstup Pm=1/30, pc=100, N=500, tT=4



Graf 17: Vstup Pm=1/30, pc=100, N=500, tT=8



Graf 18: Vstup Pm=1/30, pc=100, N=500, tT=16

Prokázalo se, že zvýšení počtu jedinců v turnajovém souboji v dané úloze má za následek konvergenci již po několika málo cyklech. Pro zajímavost byl proveden i výpočet s jedním jedincem v turnaji, což pochopitelně k zdárnému vyřešení úlohy nevede. Je vhodné ponechat počet jedinců v turnaji na hodnotě tT=2.

7.5 VYHODNOCENÍ

7.5.1 Numerické výsledky

Návrh skutečného týmu pro ověření v praxi se realizoval na základě předchozího zkoumání vhodných vstupních hodnot. Konkrétné se jednalo o tyto hodnoty:

- velikost populace N = 500
- selekce tT = 2
- pravděpodobnost mutace Pm = 1/30
- počet iterací pc = 100.

Veškeré výpočty byly kvůli statistickému porovnání provedeny 50x. Výběr prvních 20-ti iterací je uveden v následujících tabulkách. Více iterací není uváděno vzhledem k tomu, že stagnují na již zobrazených hodnotách. Pro následující tabulky platí, že čím více je barva světlejší, tím je nižší (lepší) hodnota fitness funkce.

Tab. 8: Hodnoty nejlepší fitness funkce z 50-ti výpočtů

výp./it.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	68	68	68	68	68	68	68	68	67	67	67	67	67	67	67	67	67	67	67	67	67
2	69	69	69	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
3	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
4	68	70	69	69	69	69	68	68	67	67	67	67	67	67	67	67	67	67	67	67	67
5	70	70	70	68	68	68	68	68	68	68	68	67	67	67	67	67	67	67	67	67	67
6	69	69	69	69	68	68	68	68	68	68	68	68	67	67	67	67	67	67	67	67	67
7	68	67	68	68	68	68	68	67	67	67	67	67	67	67	67	67	67	67	67	67	67
8	69	69	69	68	68	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
9	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
10	71	71	70	70	68	68	68	68	67	67	67	67	67	67	67	67	67	67	67	67	67
11	70	70	69	69	69	68	67	67	68	68	68	68	68	67	67	67	67	67	67	67	67
12	70	71	67	68	68	68	68	68	68	68	68	68	67	67	67	67	67	67	67	67	67
13	70	70	70	69	68	68	68	68	67	67	67	67	67	67	67	67	67	67	67	67	67
14	70	69	68	68	68	68	68	68	68	67	67	67	67	67	67	67	67	67	67	67	67
15	69	69	69	69	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
16	69	69	69	69	69	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
17	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
18	69	69	69	69	68	68	68	68	68	68	68	68	67	67	67	67	67	67	67	67	67
19	68	68	69	69	69	68	68	68	68	68	67	67	67	68	67	67	67	67	67	67	67
20	68	68	69	69	68	68	68	68	68	68	68	68	68	68	68	68	68	67	67	67	67
21	69	69	69	69	69	69	69	68	68	68	67	67	67	67	67	67	67	67	67	67	67
22	68	68	68	68	68	67	67	68	68	68	68	67	67	67	68	67	67	67	67	67	67
23	68	68	68	68	68	68	68	68	67	67	67	67	67	67	67	67	67	67	67	67	67
24	68	68	68	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
25	68	68	68	68	68	68	68	68	68	67	67	67	67	67	67	67	67	67	67	67	67
26	68	68	68	68	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
27	68	68	68	68	68	68	68	68	68	67	67	67	67	67	67	67	67	67	67	67	67
28	69	69	69	69	69	68	68	68	67	67	67	67	67	67	67	67	67	67	67	67	67
29	70	70	69	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
30	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
31	69	69	69	68	68	68	68	68	67	67	67	67	67	67	67	67	67	67	67	67	67

32	71	71	69	69	69	69	69	67	67	67	67	67	67	67	67	67	67	67	67	67	67
33	68	68	68	68	68	68	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
34	69	70	70	69	69	68	68	68	68	68	68	67	67	67	67	67	67	67	67	67	67
35	69	69	69	69	68	68	68	67	67	67	67	67	67	67	67	67	67	67	67	67	67
36	72	70	68	70	68	69	69	69	68	68	68	68	67	67	67	67	67	67	67	67	67
37	70	68	68	68	68	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
38	68	68	68	68	68	68	68	68	68	67	67	67	67	67	67	67	67	67	67	67	67
39	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
40	69	69	69	69	68	68	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
41	68	68	68	68	68	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
42	68	68	68	68	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
43	68	68	68	68	68	68	68	68	68	68	67	67	67	67	67	67	67	67	67	67	67
44	68	68	68	68	68	68	68	68	67	67	67	67	67	67	67	67	67	67	67	67	67
45	70	70	70	70	70	69	67	68	69	68	68	67	67	67	67	67	67	67	67	67	67
46	70	71	71	68	68	68	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
47	69	69	69	68	68	68	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
48	69	69	69	69	69	68	67	67	67	68	68	68	67	67	67	67	67	67	67	67	67
49	68	68	68	68	68	68	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
50	69	70	70	70	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67

Tab. 9: Hodnoty rozptylu fitness funkce z 50-ti výpočtů

výp\it.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	31	26	19	14	11	9	8	6	4	4	4	2	3	3	3	4	4	3	3	4	3
2	30	22	17	12	10	9	8	7	6	5	4	4	4	3	3	4	4	5	4	4	4
3	30	22	15	10	8	7	7	5	6	5	6	6	3	4	3	3	3	2	3	3	3
4	29	21	15	13	10	8	7	5	4	3	4	2	4	3	4	3	3	3	3	3	3
5	33	20	15	11	10	8	7	5	4	3	3	4	2	2	3	2	2	4	4	4	4
6	30	20	13	9	7	6	5	4	5	4	3	4	3	3	4	4	4	3	4	2	2
7	30	19	16	14	12	10	9	8	6	4	4	4	3	2	4	3	3	4	3	2	4
8	31	21	15	12	7	5	5	5	5	4	3	3	3	3	4	3	3	3	4	4	2
9	28	18	13	11	10	9	8	7	6	5	6	3	3	3	2	4	3	2	4	3	3
10	30	19	11	7	6	4	5	5	5	5	5	5	4	4	4	3	3	3	4	2	3
11	30	20	16	12	8	8	6	5	5	5	5	3	3	3	2	4	3	3	3	3	4
12	33	20	15	10	8	6	4	4	5	4	4	4	4	3	3	2	2	3	2	3	3
13	28	18	12	10	8	7	6	5	5	5	4	4	4	4	2	2	3	2	4	3	3
14	30	21	16	12	8	7	5	6	4	3	3	3	4	3	4	3	4	3	3	3	3
15	31	22	17	14	9	8	7	4	4	3	4	4	3	3	2	3	2	3	4	2	4
16	30	21	16	13	10	8	6	6	5	3	4	4	3	5	4	4	3	3	3	3	4
17	32	22	16	11	9	6	6	5	6	6	5	5	4	3	4	4	2	3	5	4	3
18	31	21	16	12	10	8	7	5	3	4	3	2	4	3	3	3	3	2	2	3	3
19	32	21	14	11	9	8	6	6	3	4	3	4	4	3	4	3	4	4	4	3	4
20	32	23	16	10	8	5	5	6	5	4	4	3	3	3	3	4	3	3	2	3	5
21	29	18	13	11	9	8	7	5	5	6	4	3	3	3	4	3	3	3	3	2	3
22	34	25	19	16	11	9	6	5	4	3	3	3	3	3	3	2	3	3	2	3	3
23	32	25	18	14	13	11	8	5	4	4	4	3	3	3	4	3	4	3	3	3	2
24	32	22	15	13	12	10	8	5	5	3	3	3	3	3	4	4	4	3	3	3	3
25	31	22	15	12	8	7	7	6	5	4	3	3	3	4	3	3	2	3	3	3	4
26	31	23	17	14	11	10	8	7	4	3	4	4	4	3	3	3	3	3	4	3	3
27	28	19	14	11	10	8	8	7	5	5	3	4	3	2	2	2	2	2	3	3	3
28	29	20	14	13	10	7	4	4	4	3	4	4	3	2	4	3	3	4	3	3	2
29	29	19	14	10	9	7	5	5	4	4	5	4	3	4	3	4	5	4	3	3	2
30	32	23	17	12	10	8	8	6	4	4	4	3	3	3	3	5	4	3	4	3	4
31	33	23	18	14	11	8	7	5	4	4	4	4	2	3	2	4	2	4	2	4	2
32	29	19	13	10	8	7	5	5	4	5	3	4	4	3	4	4	4	3	3	4	4

33	31	23	15	12	9	8	5	6	4	4	4	4	3	2	4	4	3	4	3	4	4
34	30	18	13	10	7	6	6	7	4	4	4	4	4	3	3	3	3	2	2	4	4
35	33	23	16	12	8	8	4	4	5	4	4	3	4	3	3	3	3	3	3	3	3
36	26	17	12	8	7	6	5	4	4	5	4	3	3	4	4	3	4	2	3	2	2
37	32	24	18	14	10	6	4	4	4	4	3	2	3	4	3	3	3	4	3	4	4
38	31	22	13	10	9	8	7	7	6	5	5	5	4	2	2	2	3	2	3	3	3
39	33	21	17	13	10	7	8	7	6	5	5	5	5	4	3	3	3	3	3	4	4
40	29	19	14	13	11	10	8	6	6	4	3	4	4	3	2	3	4	4	4	3	2
41	32	19	15	14	13	8	6	5	4	4	2	3	3	3	2	3	3	3	4	3	2
42	31	20	14	10	9	8	8	6	6	5	4	3	2	3	3	3	4	3	3	3	3
43	30	21	16	14	12	10	8	6	6	4	3	3	2	2	4	3	2	3	4	3	3
44	30	21	16	13	10	9	9	7	6	5	4	3	3	3	4	2	3	3	3	2	4
45	30	20	13	10	9	7	5	5	5	4	3	3	3	2	3	3	3	4	4	4	3
46	29	19	14	9	7	6	5	4	4	5	6	6	5	4	4	5	3	4	3	3	3
47	29	21	15	12	10	10	8	6	4	4	5	3	4	4	3	3	3	3	3	3	5
48	29	20	15	12	9	5	5	4	4	4	3	4	4	3	4	4	4	3	2	3	3
49	32	20	13	13	10	7	6	5	4	4	5	4	4	4	3	3	4	3	3	4	4
50	32	22	16	10	8	6	6	6	5	4	4	4	4	5	4	4	3	5	3	5	3

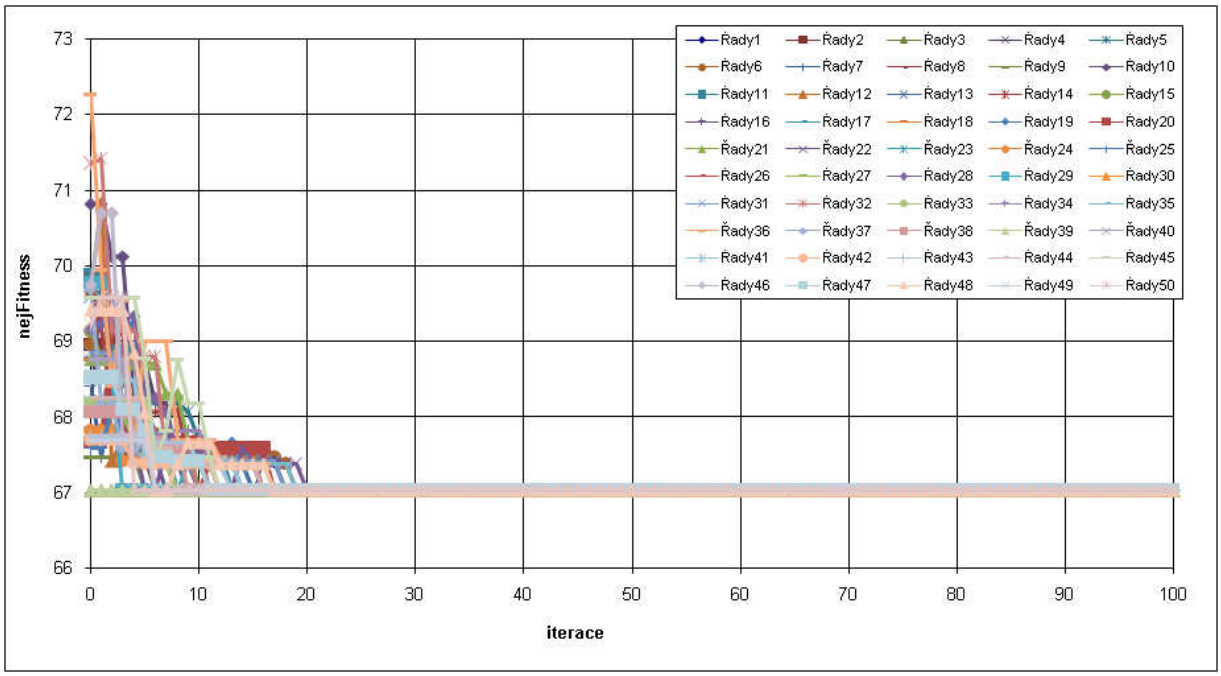
Tab. 10: Hodnoty průměrné fitness funkce z 50-ti výpočtů

výp\it.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	84	81	78	76	75	73	72	71	70	69	69	69	68	68	68	68	68	68	68	68	68
2	84	81	78	76	75	73	72	71	70	70	69	69	68	68	68	68	68	68	68	68	68
3	84	81	78	76	74	73	72	71	70	70	69	69	68	68	68	68	68	67	68	68	68
4	84	81	78	76	75	73	72	71	70	70	70	69	69	69	69	69	68	68	68	68	68
5	84	81	79	77	76	74	73	72	71	71	70	70	69	69	69	69	69	69	69	68	68
6	83	80	78	76	75	74	73	72	71	70	70	70	69	69	69	69	69	68	68	68	68
7	84	81	79	77	75	74	72	71	70	69	69	69	68	68	68	68	68	68	68	68	68
8	84	81	78	76	75	73	73	72	71	70	69	69	69	68	68	68	68	68	68	68	68
9	84	81	78	77	75	74	72	71	71	70	69	68	68	68	68	68	68	67	68	68	68
10	84	80	78	76	75	74	73	73	72	71	70	70	69	69	69	68	68	68	68	68	68
11	84	81	79	77	75	74	73	72	71	71	70	70	69	69	69	69	68	68	68	68	68
12	84	81	78	76	74	73	72	72	71	70	70	69	69	69	68	68	68	68	68	68	68
13	83	81	79	77	75	74	73	72	72	71	70	70	69	69	68	68	68	68	68	68	68
14	84	81	78	76	75	73	72	72	71	70	70	70	70	69	69	69	69	68	68	68	68
15	83	80	77	75	73	72	71	70	70	69	69	69	69	68	68	68	68	68	68	68	68
16	84	81	79	77	75	74	73	72	71	70	70	69	69	69	69	68	68	68	68	68	68
17	84	81	78	77	75	73	73	72	71	70	69	69	68	68	68	68	68	68	68	68	68
18	83	80	78	76	74	73	72	71	70	70	69	69	69	69	68	68	68	68	68	68	68
19	84	81	78	77	75	74	73	72	71	71	70	70	69	69	69	69	69	68	68	68	68
20	84	81	78	77	75	74	73	72	71	71	70	70	69	69	69	69	68	68	68	68	68
21	83	81	79	77	75	74	73	72	71	71	70	70	69	69	69	69	68	68	68	68	68
22	84	81	78	76	74	72	71	70	70	69	69	69	68	68	68	68	68	68	68	68	68
23	84	81	78	76	74	72	71	70	69	69	69	69	69	68	68	68	68	68	68	68	68
24	83	81	78	76	74	73	71	70	69	69	68	68	68	68	68	68	68	68	68	68	68
25	83	80	78	76	75	73	72	71	70	69	69	69	69	69	68	68	68	68	68	68	68
26	84	81	79	76	75	73	72	71	70	69	69	69	68	68	68	68	68	68	68	68	68
27	84	81	78	76	75	74	72	71	70	69	69	69	68	68	68	68	68	68	68	68	68
28	83	81	78	76	75	73	72	72	71	71	70	70	69	69	69	69	68	68	68	68	68
29	84	81	79	77	75	74	73	72	71	71	70	70	69	69	68	68	68	68	68	68	68
30	84	81	78	76	74	73	72	71	70	69	69	68	68	68	68	68	68	68	68	68	68
31	83	80	78	76	74	72	71	70	70	69	69	69	68	68	68	68	68	68	68	68	68
32	84	81	79	76	75	74	73	72	71	71	70	70	70	69	69	69	69	68	68	68	68
33	84	81	78	76	75	73	72	71	70	70	69	69	69	68	68	68	68	68	68	68	68

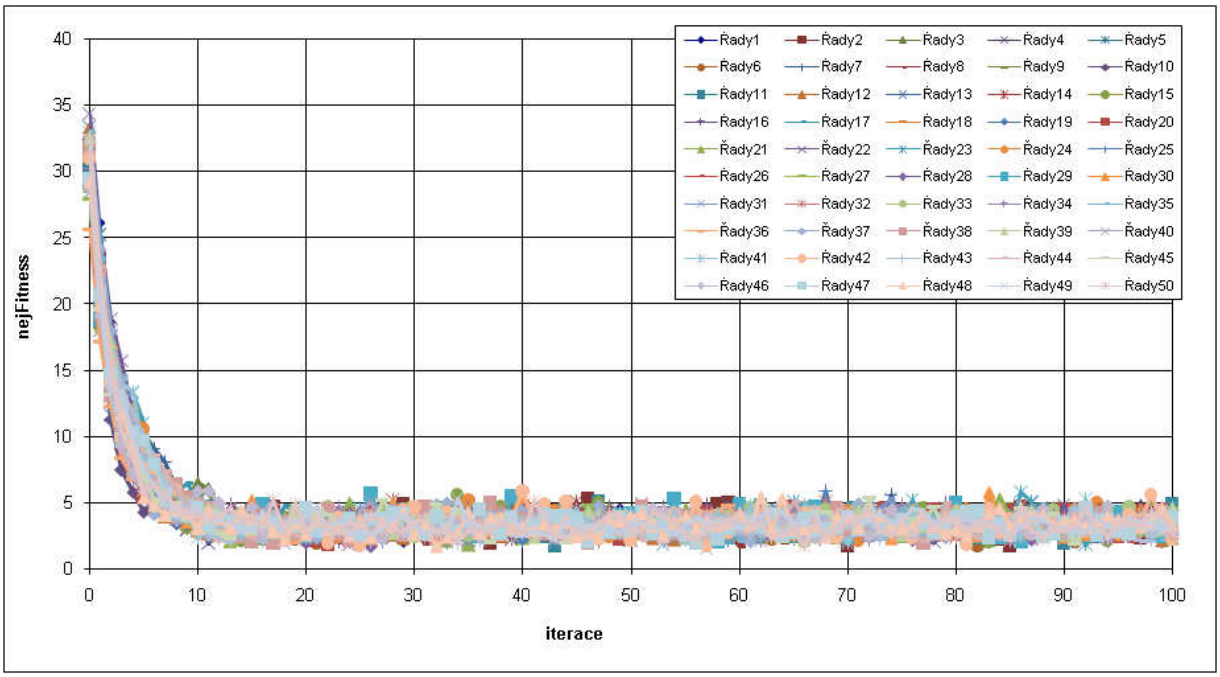
34	84	81	79	77	76	75	74	73	72	71	71	70	70	69	69	69	69	68	68	68	68
35	84	81	78	76	75	73	72	71	71	70	69	69	69	68	68	68	68	68	68	68	68
36	84	81	78	77	75	74	73	72	71	71	71	70	70	70	70	69	69	68	69	68	68
37	84	81	78	75	74	72	71	70	70	69	69	68	68	68	68	68	68	68	68	68	68
38	84	81	78	76	75	74	73	72	71	70	69	69	69	68	68	68	68	68	68	68	68
39	83	80	78	76	74	73	72	71	70	69	69	68	68	68	68	68	68	68	68	68	68
40	84	81	78	77	75	74	72	71	71	70	69	69	69	68	68	68	68	68	68	68	68
41	84	80	78	76	74	72	71	70	70	69	69	69	69	68	68	68	68	68	68	68	68
42	83	80	78	76	75	73	72	71	71	70	69	69	68	68	68	68	68	68	68	68	68
43	84	81	79	77	75	73	72	71	70	69	69	68	68	68	68	68	68	68	68	68	68
44	84	81	79	77	75	73	72	71	70	70	69	69	68	68	68	68	68	68	68	68	68
45	84	81	78	77	75	74	73	72	71	71	70	70	70	70	70	69	69	69	69	68	68
46	83	81	79	77	75	74	74	73	72	71	71	70	69	69	68	68	68	68	68	68	68
47	84	81	78	76	75	73	72	71	70	69	69	69	69	68	68	68	68	68	68	68	68
48	84	81	78	76	75	73	73	72	71	71	70	70	69	69	69	69	69	68	68	68	68
49	84	80	78	76	75	73	72	71	70	70	70	69	69	68	68	68	68	68	68	68	68
50	83	80	78	76	74	73	72	71	70	70	70	69	69	69	68	68	68	68	68	68	68

7.5.2 Grafické výsledky

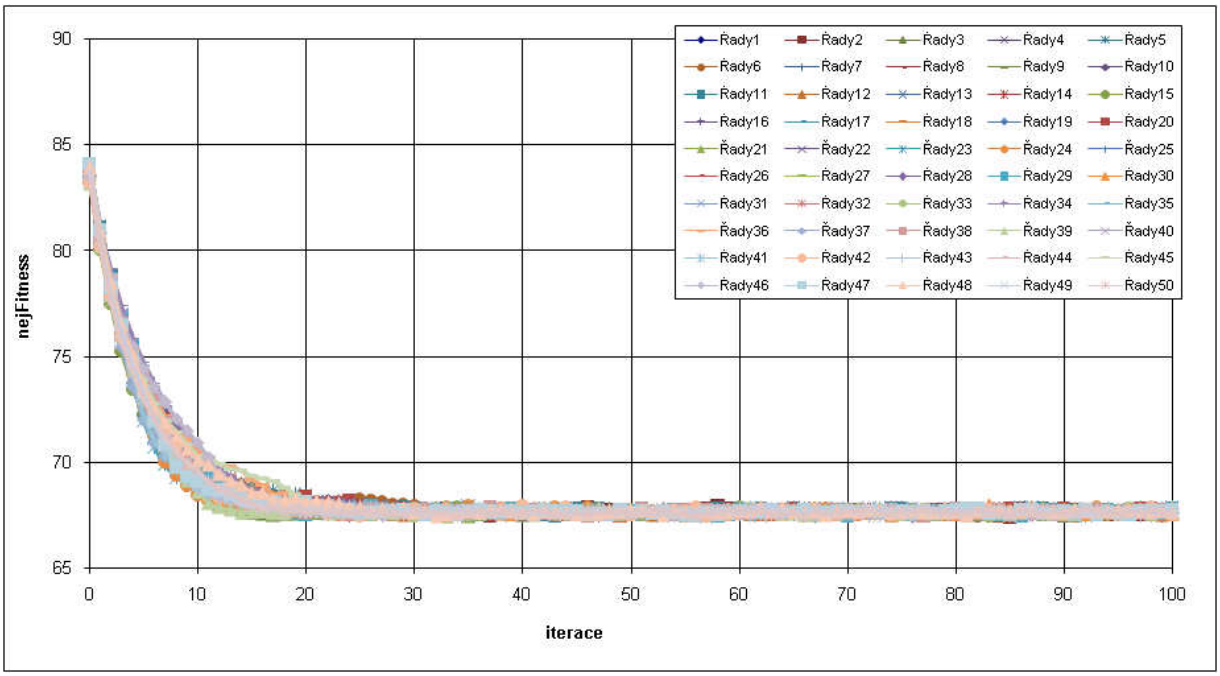
Výsledné hodnoty zobrazené grafickou formou jsou na následujících grafech:



Graf 19: Průběh nejlepší fitness funkce z 50-ti výpočtů



Graf 20: Průběh rozptylu fitness funkce z 50-ti výpočtů



Graf 21: Průběh průměrné fitness funkce z 50-ti výpočtů

7.5.3 Dekódování výsledků

Z dílčích uložených textových souborů s výsledky pro každou skupinu vyplývá, že do výsledného týmu byli vybráni následující zaměstnanci. Výsledky jsou v podobě číselných kódů, do nichž byl na začátku každý zaměstnanec zakódován. Je proto nutné podívat se nakonec do tabulky č. 6 a podle číselných kódů si zpětně najít jména příslušných vybraných zaměstnanců. V tabulce č. 11 jsou zobrazeni již pouze vybraní zaměstnanci, kteří tvoří optimální tým.

Tab. 11: Hotový tým z vybraných zaměstnanců

Název skupiny dle organizačního členění	**Jméno a příjmení**	**1: Angličtina**	**2: Znalosti PC**	**3: Odbornost v oboru**	**4: Řidičský průkaz sk. B**	**5: Primární týmová role dle Belbina**	**6: Sekundární týmová role dle Belbina**	**7: Subjektivní hodnocení**	**Výsledný kód**
Logistika	Milan Jandák	2	1	1	1	1	3	2	**2111132**
	Michal Toman	3	2	2	1	6	5	1	**3221651**
Obchod	Jiří Borovec	2	1	2	1	1	2	2	**2121122**
	Alena Malá	2	3	2	1	2	4	1	**2321241**
Personalistika	Pavla Šplíchová	5	3	2	1	5	3	3	**5321533**
Projekce	Barbora Zubrová	3	2	2	2	2	6	1	**3222261**
	Romana Švermová	2	1	2	1	8	4	2	**2121842**
	Kamila Nováková	3	2	1	2	8	6	2	**3212862**
	Jana Vesecká	3	3	3	1	5	6	3	**3331563**
Stavbyvedoucí	Aleš Bodlák	2	2	1	1	4	3	3	**2211433**
	Martin Ondráček	3	3	2	1	7	4	2	**3321742**
	Kamil Černý	4	2	2	1	5	6	1	**4221561**
	Roman Podlužný	4	3	1	1	7	8	2	**4311782**
Účetní	Anna Kočí	3	3	1	1	4	7	3	**3311473**

7.5.4 Zastoupení týmových rolí dle Belbina

Z následující tabulky č. 9 je patrné, že zaměstnanci, navržení do popisovaného týmu, disponují potřebnou různorodostí osobních charakteristik a v týmu se tak vyskytují všechny role definované Belbinem. Podle Belbinovy teorie týmových rolí má tento tým šanci na bezproblémové fungování, což se v praxi potvrdilo.

Tab. 12: Zastoupení týmových rolí dle Belbina ve výsledném týmu

Název role	**Počet výskytů primárně**	**Počet výskytů sekundárně**	**Počet výskytů celkem**
1 = Inovátor	2	0	2
2 = Vyhledávač zdrojů	2	1	3
3 = Koordinátor	0	3	3
4 = Usměrňovač	2	3	5
5 = Monitor vyhodnocovač	3	1	4
6 = Týmový pracovník	1	4	5
7 = Realizátor	2	1	3
8 = Kompletovač finišer	2	1	3

8 ZÁVĚR

Navržený způsob optimalizace tvorby týmů pomocí genetických algoritmů prokázal schopnost efektivně a rychle řešit zadaný problém. Vypočítané sestavy osob, které spolu mají vytvořit tým, se v praxi ukázaly jako fungující a takto sestavené týmy skutečně úspěšně plnily své úkoly. Jelikož se řešená problematika nachází v oblasti jednorázových, objektivně neopakovatelných projektů, považuje se za úspěšný takový tým, který dle očekávání vedoucích pracovníků splní zadané úkoly kvalitně a v termínu. Není dost dobře možné vrátit se v reálném životě zpět a vyzkoušet, jak by danou zakázku (projekt či úkol) splnil jiný tým v tom stejném čase, prostředí a okolních vlivech. Je tudíž vhodné vycházet i ze zkušeností a porovnat genetickým algoritmem navržený tým s představou vedoucího pracovníka.

V rámci testování programu byly rovněž zpětně vytvořeny modely již existujících týmů ve více firmách. V rámci konzultací s vedoucími pracovníky potom byly diskutovány rozdíly mezi modely a skutečností a tím byly rovněž validovány výsledky výpočtů, jelikož vymodelované týmy byly z pohledu těchto vedoucích pracovníků životaschopné..

Zhodnocení splnění cílů práce:

- matice hodnotících kritérií – byla navržena obecně použitelná matice pro zaznamenání vstupních osobních i technických parametrů jednotlivých osob. Rozměr matice je možné měnit dle konkrétní řešené úlohy a to včetně individuálního zvolení počtu osobních i technických parametrů.
- fitness funkce – byla navržena fitness funkce (FT), která v sobě zahrnuje tři dílčí fitness funkce osobních vlastností (FOV), synergického efektu (FSE) a znalostí a zkušeností (FZZ). Pomocí vah je možné upřednostnit či potlačit kteroukoliv z těchto dílčích funkcí dle požadavků konkrétní řešené úlohy.
- vhodné vstupní parametry genetického algoritmu – testováním citlivosti navrženého programu na různé hodnoty vstupních parametrů genetického algoritmu byly nalezeny následující hodnoty, při kterých lze získat optimální výsledky v reálném čase: velikost populace N = 500, selekce tT = 2, pravděpodobnost mutace Pm = 1/30 a počet iterací pc = 100.
- způsob zakódování jedinců – z matic hodnotících kritérií, které byly vyplněny pro všechny uvažované osoby, byly sestaveny číselné kódy pro každého člověka, pod kterým daný člověk vstupuje do optimalizačního programu.
- způsob zadání požadavků na výsledný tým – realizováno pomocí matice požadovaných kvalit (známek) ve výsledném týmu pro každou profesní skupinu zvlášť.
- váhová matice významnosti dílčích vlastností – v rámci definice požadavků na výsledný tým je možné váhovým způsobem upřednostnit či potlačit kteroukoliv konkrétní vlastnost pro každou profesní skupinu zvlášť dle požadavků řešené úlohy.
- vlastní algoritmus pro testování této hypotézy – byl vytvořen optimalizační program v jazyku Java (vývojové prostředí NetBeans 5.0 IDE). Výsledné matice byly dále zpracovány v tabulkovém procesoru MS Excel za účelem vytvoření grafů a tabulek v barevných škálách z důvodů snadné optické kontroly výsledků výpočtů.

- způsob interpretace výsledků – matice číselných kódů, která vykazovala nejlepší (nejnižší) hodnotu fitness funkce, byla podle kódovací tabulky zpětně převedena na jména konkrétních osob, kterým dané číselné kódy patřily a vyhodnocena. Tímto byl získán optimální výsledný tým.

Vytvořený optimalizační program je experimentálního charakteru (pro účely disertační práce). K rutinnímu využití v podnikové praxi by bylo zapotřebí vytvořit verzi programu s grafickým uživatelským rozhraním a umožnit uživateli snadněji měnit nastavení různých parametrů. Nicméně je možné optimalizační program plnohodnotně používat i ve stávající formě, kde je k dispozici méně komfortní uživatelské rozhraní v textové podobě. Navržené řešení je detailně popsáno (včetně celého algoritmu a nastavení) a stává se tak východiskem pro další vědecko-výzkumnou činnost.

9 LITERATURA

[1] SEKAJ, I. Riešenie problémov pomocou genetických algoritmov. *Automatizace*, září 2004, roč. 47, č. 9, s. 552-555. ISSN 0005-125X.

[2] SLÁMA, L. *Genetický algoritmus a jeho využití pro řešení identifikačních a optimalizačních úloh inženýrské mechaniky*. 1. vyd. Brno: Vysoké učení technické, 2000. 31 s. ISBN 80-214-1773-0.

[3] HUBER, A. *Emocionální inteligence*. 1. vyd. Praha: ZEMS, 2005. 90 s. ISBN 80-903305-6-8.

[4] DAWKINS, R. *Sobecký gen*. 2. vyd. Praha: Mladá fronta, 1998. 320 s. ISBN 80-204-0730-8.

[5] SEKAJ, I. Genetické algoritmy pri návrhu regulátorov a pri statickej optimálizácii procesov. *Automatizace*, září 2004, roč. 47, č. 10, s. 617-620. ISSN 0005-125X.

[6] MAŘÍK, V. - ŠTĚPÁNKOVÁ, O. - LAŽANSKÝ, J. a kol. *Umělá inteligence (3)*. Praha: Academia, 2001. 328 s. ISBN 80-200-0472-6.

[7] ZELINKA, I. *Umělá inteligence v problémech globální optimalizace.* Praha: Ben, 2002. 192 s. ISBN 80-7300-069-5.

[8] BARTSCH, H. J. *Matematické vzorce.* Praha: Mladá Fronta, 1996. 831 s. ISBN 80-204-0607-7.

[9] OŠMERA, P. *Genetické algoritmy a jejich aplikace.* [Habilitační práce]. Brno. Vysoké učení technické v Brně, 2001. 108 s.

[10] HAYES, N. *Psychologie týmové práce – strategie efektivního vedení týmu.* 1. vyd. Praha: Portál, s.r.o., 2005. 189 s. ISBN 80-7178-983-6.

[11] KOLAJOVÁ, L. *Týmová spolupráce.* 1. vyd. Praha: Grada Publishing a.s., 2006. 105 s. ISBN 80-24717-64-6.

[12] BAY, R. *Účinné vedení týmů.* 1. vyd. Praha: Grada Publishing, spol. s r.o., 2000. 152 s. ISBN 80-247-9068-8.

[13] KLAPKA, J., PIŇOS, P. Decision support system for multicriterial R&D and information systems projects selection. *European Journal of Operational Research 140*, 2002, p. 434-446.

[14] BELBIN, M. *Management Teams.* 2nd ed. Oxford: Elsevier, 2004. 201 p. ISBN 0-7506-5910-6.

[15] BELBIN, M. *Team Roles at Work.* 1st ed. Oxford: Elsevier, 2003. 141 p. ISBN 0-7506-2675-5.

[16] HLAOITTINUN, O., BONJOUR, E., DULMET, M. A multidisciplinary team building method based on competency modelling in design project management. *International Journal of Management Science and Engineering Management*, 2008, Vol. 3, No. 3, p 163-175. ISSN 1750-9653.

Elektronické zdroje informací

[17] ČERNÝ, T. *Demonstrační program GATSP.exe* [online]. Bakalářská práce. Brno: Masarykova Universita, 2004. [cit. 2006-05-03]. Dostupné z <http://www.cba.muni.cz>.

[18] Hestley a.s. *Co umožňuje technologie genetických algoritmů* [online], 2005. [cit. 2006-04-21]. Dostupné z <http://www.hestley.com>.

[19] ŽIŽKA, J: *Evoluční výpočty – Genetické algoritmy* [online], 2005. [cit. 2006-05-03]. Dostupné z < http://www.cba.muni.cz >.

[20] Computer Science Department at Boston University. *Parallel Genetic Algorithms* [online], 2005. [cit. 2006-02-16]. Dostupné z <http://cs-pub.bu.edu>.

[21] The Genetic Programming Notebook. *The Genetic Programming Tutorial* [online], 2005. [cit. 2006-01-25]. Dostupné z <http://www.geneticprogramming.com>.

[22] OŠMERA, P. *Neuronové sítě* [online], Brno: VUT v Brně, FSI, ÚAI, 2004. [cit. 2004-10-11]. Dostupné z <http://www.fme.vutbr.cz>.

[23] JOHNO. *Minimalizácia CSS genetickým algoritmom* [online], 2004. [cit. 2005-08-20]. Dostupné z <http://johno.jsmf.net>.

[24] Hestley a.s. *Analýza pracovníků metodou genetických algoritmů* [online], 2005. [cit. 2006-04-21]. Dostupné z <http://www.analyzy.cz>.

[25] Hestley a.s. *Moderní a tradiční matematické metody analýzy dat* [online], 2005. [cit. 2006-04-21]. Dostupné z <http://www.analyzy.cz>.

[26] MAREŠ, M. *Design pro formuli 1 vyvíjí i genetický algoritmus* [online], 2004. [cit. 2005-12-07]. Dostupné z <http://ihned.cz>.

[27] WIKIPEDIA. *Hlavní strana – Rozptyl (statistika) – Wikipedie, otevřená encyklopedie* [online]. [citováno 7. 10. 2006]. Dostupné z URL <http://cs.wikipedia.org/wiki/Rozptyl_%28statistika%29>

[28] BELBIN – týmové role. *Popis týmových rolí* [online], 2008. [cit. 2008-03-05]. Dostupné z <http://www.belbin.cz>.

[29] TEAM.CZ, s.r.o. *Belbin Team Roles* [online], 2004. [cit. 2007-11-10]. Dostupné z <http://www.teamtech.cz>.

[30] Týmové role. *Týmové role* [online], 1999. [cit. 2007-04-08]. Dostupné z <http://lukov.brontosaurus.cz>.

10 POUŽITÉ ZKRATKY

Zkratka	Význam zkratky anglicky	Význam zkratky česky
CAD	Computer Aided Design	Počítačová podpora konstruování
FOV		Dílčí fitness funkce osobních vlastností
FSE		Dílčí fitness funkce synergického efektu
FZZ		Dílčí fitness funkce znalostí a zkušeností
GA	Genetic Algorithm	Genetický algoritmus

11 SEZNAM PŘÍLOH

Příloha č. 1 – Dotazník pro zjištění rolí v týmu dle Dr. Belbina

JAK VIDÍM SVOU ROLI V TÝMU

Tento dotazník má celkem sedm sekcí označených římskými číslicemi. V každé sekci tohoto dotazníku zakroužkujte ty výroky, které vás nejlépe vystihují. Můžete zakroužkovat jeden, dva nebo více výroků. Zakroužkované výroky poté ohodnoťte bodovým hodnocením tak, že mezi ně rozdělíte vždy deset bodů v každé sekci.

I. Čím, jak věřím, mohu být v týmu prospěšný(á):

a) Myslím, že si dokážu rychle všimnout nových příležitostí a včas jich využít.

b) Mohu dobře spolupracovat s velmi širokým okruhem lidí.

c) Velmi snadno a přirozeně přicházím na nové myšlenky a nápady.

d) Dokážu vyhecovat lidi k činnosti kdykoliv zjistím, že mohu něčím cenným přispět ke skupinovým cílům.

e) Moje schopnost dotahovat věci do konce vyplývá z mé osobní výkonnosti.

f) Dokážu čelit dočasné neoblíbenosti, jestliže to nakonec vede k dobrým výsledkům.

g) Rychle vycítím, co se má dělat v situaci, kterou znám.

h) Dovedu bez předsudků a zaujatostí najít rozumné důvody pro změnu zaměření činnosti (resp. pro alternativní průběh nějaké akce).

II. Kdybych měl(a) nedostatky v týmové práci, byly by to nejspíš:

a) Necítím se dobře, pokud pracovní schůzka nemá jasnou strukturu a není dobře řízená.

b) Mám tendenci být příliš velkorysý k lidem, kteří zastávají opodstatněné stanovisko, jemuž nebyla věnována náležitá pozornost.

c) Mám tendenci mluvit příliš mnoho, když se skupina dostane k novým myšlenkám.

d) Můj objektivní náhled mi neumožňuje sdílet nadšení ostatních.

e) Někdy se jevím jako příliš energický a autoritářský, když je potřeba něco dodělat.

f) Je pro mě těžké být v popředí nebo vystupovat ve vedoucí roli, snad proto, že jsem příliš citlivý na atmosféru ve skupině.

g) Stává se mi, že se tak ponořím do svých myšlenek, že ztratím ponětí o tom, co se děje.

h) Druzí mi někdy vyčítají, že se příliš starám o podružné detaily a lámu si hlavu nad možnými nezdary.

III. Když spolupracuji na nějakém projektu s jinými lidmi:

a) Mám schopnost lidi ovlivnit, aniž bych je k něčemu nutil.

b) Moje bdělost umožňuje předcházet omylům a chybám z nepozornosti.

c) Jsem připraven tlačit ostatní do činnosti, aby se na setkání neztrácel čas a zřetel na hlavní cíl.

d) Dá se počítat s tím, že přispěji něčím originálním.

e) Jsem vždycky připraven hájit dobrý návrh ve společném zájmu.

f) Jsem blázen do nových myšlenek a posledních vývojových novinek.

g) Věřím, že ostatní oceňují mou schopnost chladného úsudku.

h) Je na mně spolehnutí, že dohlédnu na to, aby se udělalo, co je třeba.

IV. Můj charakteristický přístup ke skupinové práci je, že:

a) Mám zájem lépe poznat své kolegy.

b) Nezdráhám se odmítnout názory druhých a zastávat sám menšinové stanovisko.

c) Obvykle najdu řadu argumentů vyvracejících nesmyslné návrhy.

d) Dokážu uvést věci do chodu, když je třeba začít uskutečňovat plán.

e) Mám tendenci vyhýbat se obvyklým věcem a přicházet s něčím nečekaným.

f) Snažím se vnést náznak dokonalosti do každé týmové práce, na níž se podílím.

g) Dokážu využít kontaktů vně samotného týmu.

h) I když se zajímám o všechny názory, neváhám se rozhodnout, jakmile je nutno nějaké rozhodnutí učinit.

V. Práce mě těší, protože:

a) Baví mě analyzovat situace a zvažovat všechny možné volby.

b) Zajímá mě nacházet praktická řešení problémů.

c) Rád cítím, že podporuji dobré pracovní vztahy.

d) Mohu uplatnit silný vliv na rozhodnutí.

e) Mám příležitost setkávat se s lidmi, kteří mi mohou poskytnout novou zkušenost.

f) Dokážu sjednotit názory různých lidí a vést je ke společné žádoucí činnosti.

g) Cítím se ve svém živlu, když se mohu plně věnovat nějakému úkolu.

h) Rád mám věci, které rozšiřují moji představivost.

VI. Kdybych nečekaně dostal obtížný úkol, který je nutno splnit v omezeném čase a s neznámými lidmi:

a) Sedl bych si někam do kouta, přemýšlel jak se dostat ze slepé uličky a snažil se ujasnit další postup.

b) Byl bych ochoten pracovat s člověkem, který projevuje nejpozitivnější přístup, bez ohledu na to, že může být hůře snesitelný.

c) Hledal bych nějaký způsob, jak úkol zjednodušit tím, že bych si zjistil, čím mohou různí jednotlivci nejlépe přispět.

d) Můj přirozený cit pro povinnost by pomohl zajistit, že dodržíme harmonogram.

e) Věřím, že bych zůstal klidný a udržel si svou schopnost jasně myslet.

f) Držel bych se stále konečného cíle navzdory tlakům.

g) Byl bych ochoten ujmout se vedení, kdybych cítil, že se skupina nehýbá z místa.

h) Zahájil bych diskuzi se záměrem stimulovat nové myšlenky a uvést věci do pohybu.

VII. V souvislosti s problémy, které mi přináší práce ve skupině:

a) Mám sklon projevovat netrpělivost s těmi, kdo zdržují postup.

b) Ostatní mě mohou kritizovat za to, že jsem příliš analytický a nepříliš intuitivní.

c) Moje potřeba ujistit se, že práce je udělána dobře, může být překážkou postupu.

d) Snadno se začnu nudit a spoléhám na někoho z týmu, že mě vyburcuje.

e) Je pro mě obtížné začít, dokud nejsou jasné cíle.

f) Někdy se mi nedaří vysvětlovat a objasňovat složité myšlenky, které mě napadají.

g) Jsem si vědom toho, že požaduji od ostatních věci, které sám nedovedu.

h) Váhám se postavit na odpor, když se setkám se skutečnou opozicí.

VYHODNOCENÍ

Přepište bodové hodnocení z testu do následující tabulky. Napravo sečtěte bodová hodnocení za jednotlivé řádky.

	I.	II.	III.	IV.	V.	VI.	VII.	Body celkem
Koordinátor	d)	b)	a)	h)	f)	c)	g)	
Usměrňovač	f)	e)	c)	b)	d)	g)	a)	
Inovátor	c)	g)	d)	e)	h)	a)	f)	
Monitor vyhodnocovač	h)	d)	g)	c)	a)	e)	b)	
Realizátor	g)	a)	h)	d)	b)	f)	e)	
Vyhledávač zdrojů	a)	c)	f)	g)	e)	h)	d)	
Týmový pracovník	b)	f)	e)	a)	c)	b)	h)	
Kompletovač finišer	e)	h)	b)	f)	g)	d)	c)	

[30]

Příloha č. 2 – Zdrojový kód programu

```
//***** OPTIMALIZACE PRACOVNÍHO TÝMU *****
//**************************************
import java.util.*;
import java.io.*;
class optiTym {
    public static void main(String arg[]) throws IOException {
        Random rand = new Random();


        //*** 1. DATABÁZE ZAMĚSTNANCŮ ***
        //zaměstnanci jsou rozdělení do pracovních skupin
        //1=Logistika, 2=Obchod, 3=Personalistika, 4=Projekce, 5=Stavbyvedoucí,
6=Účetní
        //* 1=LOGISTIKA *
        int logistika[][]={
            {2,1,1,1,1,3,2},
            {4,4,3,1,8,2,3},
            {3,2,2,1,6,5,1}
        };
        //* 2=OBCHOD*
        int obchod[][]={
            {2,1,2,1,1,2,2},
            {1,1,1,1,1,3,1},
            {2,3,2,1,2,4,1}
        };
        //* 3=PERSONALISTIKA *
        int personalistika[][]={
            {3,2,1,1,6,7,1},
            {5,3,2,1,5,3,3}
        };
        //* 4=PROJEKCE *
        int projekce[][]={
            {2,1,1,1,8,3,1},
            {3,2,2,2,2,6,1},
            {3,3,3,1,5,6,3},
            {1,1,1,1,1,3,1},
            {2,1,2,1,8,4,2},
            {3,2,1,2,8,6,2}
        };
        //* 5=STAVBYVEDOUCÍ *
        int stavbyvedouci[][]={
            {4,2,2,1,5,6,1},
            {5,5,2,1,7,6,3},
            {4,3,1,1,7,8,2},
            {3,2,1,1,1,5,1},
            {2,2,1,1,4,3,3},
            {3,3,2,1,7,4,2},
            {5,4,2,1,8,6,2}
        };
        //* 6=ÚČETNÍ *
        int ucetni[][]={
            {1,1,2,1,8,1,1},
            {2,2,2,2,2,6,2},
            {3,3,1,1,4,7,3}
        };

        //*** presun kodu k ukladani dat ***
        File PFS = new File("C:\\CESTA\\prumernaFitnessStatistika.txt");
        FileWriter vi = new FileWriter(PFS);
        File RFS = new File("C:\\CESTA\\rozptylFitnessStatistika.txt");
        FileWriter vii = new FileWriter(RFS);
        File NFS = new File("C:\\CESTA\\nejlepsiFitnessStatistika.txt ");
        FileWriter viii = new FileWriter(NFS);

        //***** VNĚJŠÍ CYKLUS PROGRAMU *****
        int povy = 50;
```

```
            for (int povyp=0;povyp<povy;povyp++) {

            //*** 2. SLOŽENÍ A VELIKOST PRACOVNÍHO TÝMU ***
            //Číslo v 1D poli tym[] = počet zaměstnanců pracovní skupiny.
            //Součet prvků 1D pole tym[] = počet zaměstnanců týmu.
            int tym[]={2,2,1,4,4,1};
            //sečtení prvků 1D pole tym[]
            int sum=0;
            for (int i=0; i<tym.length; i++) {
                sum += tym[i];
            }
            //vytvoření 1D pole tymExpand[], počet prvků 1D pole = velikost
pracovního týmu
            int tymExpand[]=new int[sum];
            //přiřazení hodnot poli tymExpand[], hodnota prvků označuje skupinu
zaměstnance, tj. 1-6
            int itr=0;
            int ltr=0;
            for (int j=0;j<sum;j++) {
                tymExpand[j]=itr+1;
                ltr++;
                if((tym[itr]-ltr)==0) {
                    itr++;
                    ltr=0;
                }
            }
            //***** zacatek vypisu 1
            //System.out.println("** Vypis pole tymExpand **");
            //vypis1DpoleINT (tymExpand);
            //System.out.println("\n");
            //***** konec vypisu 1


            //*** 3. PARAMETRY GENETICKÉHO ALGORITMU (GA) ***
            //velikost populace (počet jedinců v generaci)
            int velPop = 700;
            //převrácená hodnota určuje pravděpodobnost mutace Pm
            int Pm = 10;


            //*** 4. INICIALIZACE PRACOVNÍHO TÝMU ***
            //vytvoření 3D polí, zde bude uložena náhodně generována populace
            int inicializaceL[][][]=new int[tym[0]][logistika[0].length][velPop];
            int inicializaceO[][][]=new int[tym[1]][obchod[0].length][velPop];
            int inicializacePE[][][]=new
int[tym[2]][personalistika[0].length][velPop];
            int inicializacePR[][][]=new int[tym[3]][projekce[0].length][velPop];
            int inicializaceS[][][]=new int[tym[4]][stavbyvedouci[0].length][velPop];
            int inicializaceU[][][]=new int[tym[5]][ucetni[0].length][velPop];
            //naplnění 3D polí inicializace pomocí 3. metody

inicializaceL=plneniPoleInicializace(inicializaceL,logistika,tym[0],velPop);
            inicializaceO=plneniPoleInicializace(inicializaceO,obchod,tym[1],velPop);

inicializacePE=plneniPoleInicializace(inicializacePE,personalistika,tym[2],velPop);

inicializacePR=plneniPoleInicializace(inicializacePR,projekce,tym[3],velPop);

inicializaceS=plneniPoleInicializace(inicializaceS,stavbyvedouci,tym[4],velPop);
            inicializaceU=plneniPoleInicializace(inicializaceU,ucetni,tym[5],velPop);
            //***** zacatek vypisu 2
            //System.out.println("** Vypis pole inicializaceL **");
            //vypis3DpoleINT (inicializaceL);
            //System.out.println();
            //System.out.println("** Vypis pole inicializaceO **");
            //vypis3DpoleINT (inicializaceO);
            //System.out.println();
            //System.out.println("** Vypis pole inicializacePE **");
```

```
            //vypis3DpoleINT (inicializacePE);
            //System.out.println();
            //System.out.println("** Vypis pole inicializacePR **");
            //vypis3DpoleINT (inicializacePR);
            //System.out.println();
            //System.out.println("** Vypis pole inicializaceS **");
            //vypis3DpoleINT (inicializaceS);
            //System.out.println();
            //System.out.println("** Vypis pole inicializaceU **");
            //vypis3DpoleINT (inicializaceU);
            //System.out.println();
            //***** konec vypisu 2


            //*** 5. URČENÍ FITNESS FUNKCE POPULACE ***
            //určení počtu osobních vlastností u pracovních skupin
            int POV[] = {3,3,3,3,3,3};
            //stanovení vah u dílčích fitness funkcí pracovních skupin
            double
vahy[][]={{2.0,1.0,1.0},{1.0,1.0,1.0},{1.0,1.0,1.0},{1.0,1.0,1.0},{1.0,1.0,1.0},{1.0
,1.0,2.0}};
            //vytvoření 2D polí, kde budou uloženy hodnoty dílčích fitness funkcí
            double FZZ[][]=new double[velPop][6];
            double FSE[][]=new double[velPop][6];
            double FOV[][]=new double[velPop][6];
            //zadání požadovaných známek u pracovních skupin
            int pozadovaneL[]={3,2,1,1,8,3,2};
            int pozadovaneO[]={1,1,1,1,1,2,1};
            int pozadovanePE[]={3,2,2,1,6,3,1};
            int pozadovanePR[]={2,1,1,1,8,4,2};
            int pozadovaneS[]={3,2,2,1,7,5,2};
            int pozadovaneU[]={3,2,1,1,8,7,1};
            //vytvoření 2D polí, ve kterých budou uloženy součty známek pracovních
skupin
            int SZZZ1[][]=new int[velPop][logistika[0].length];
            int SZZZ2[][]=new int[velPop][obchod[0].length];
            int SZZZ3[][]=new int[velPop][personalistika[0].length];
            int SZZZ4[][]=new int[velPop][projekce[0].length];
            int SZZZ5[][]=new int[velPop][stavbyvedouci[0].length];
            int SZZZ6[][]=new int[velPop][ucetni[0].length];
            //využití 2. metody ke stanovení součtu známek u pracovních skupin
            SZZZ1=souctyZnamekSkupina(SZZZ1,inicializaceL,velPop);
            SZZZ2=souctyZnamekSkupina(SZZZ2,inicializaceO,velPop);
            SZZZ3=souctyZnamekSkupina(SZZZ3,inicializacePE,velPop);
            SZZZ4=souctyZnamekSkupina(SZZZ4,inicializacePR,velPop);
            SZZZ5=souctyZnamekSkupina(SZZZ5,inicializaceS,velPop);
            SZZZ6=souctyZnamekSkupina(SZZZ6,inicializaceU,velPop);
            //***** zacatek vypisu 3
            //System.out.println("** Vypis pole SZZZ1 **");
            //vypis2DpoleINT (SZZZ1);
            //System.out.println();
            //System.out.println("** Vypis pole SZZZ2 **");
            //vypis2DpoleINT (SZZZ2);
            //System.out.println();
            //System.out.println("** Vypis pole SZZZ3 **");
            //vypis2DpoleINT (SZZZ3);
            //System.out.println();
            //System.out.println("** Vypis pole SZZZ4 **");
            //vypis2DpoleINT (SZZZ4);
            //System.out.println();
            //System.out.println("** Vypis pole SZZZ5 **");
            //vypis2DpoleINT (SZZZ5);
            //System.out.println();
            //System.out.println("** Vypis pole SZZZ6 **");
            //vypis2DpoleINT (SZZZ6);
            //System.out.println();
            //***** konec vypisu 3
            //* určení dílčích fitness funkcí FZZ, 4. metoda *
```

```
FZZ=urceniFunkceFZZ(FZZ,SZZZ1,inicializaceL,logistika,POV[0],pozadovaneL,velPop,0);

FZZ=urceniFunkceFZZ(FZZ,SZZZ2,inicializaceO,obchod,POV[1],pozadovaneO,velPop,1);

FZZ=urceniFunkceFZZ(FZZ,SZZZ3,inicializacePE,personalistika,POV[2],pozadovanePE,velP
op,2);

FZZ=urceniFunkceFZZ(FZZ,SZZZ4,inicializacePR,projekce,POV[3],pozadovanePR,velPop,3);

FZZ=urceniFunkceFZZ(FZZ,SZZZ5,inicializaceS,stavbyvedouci,POV[4],pozadovaneS,velPop,
4);

FZZ=urceniFunkceFZZ(FZZ,SZZZ6,inicializaceU,ucetni,POV[5],pozadovaneU,velPop,5);
          //***** zacatek vypisu 4
          //System.out.println("** Vypis pole FZZ **");
          //vypis2DpoleDOUBLE (FZZ);
          //System.out.println();
          //***** konec vypisu 4
          //* určení dílčích fitness funkcí FSE, 11. metoda *
          //vytvoření 3D pomocných polí pracovních skupin, zde budou některé pozice
z 3D polí inicializace jiné
          int pomocneL[][][]=new int[tym[0]][logistika[0].length-POV[0]][velPop];
          int pomocneO[][][]=new int[tym[1]][obchod[0].length-POV[1]][velPop];
          int pomocnePE[][][]=new int[tym[2]][personalistika[0].length-
POV[2]][velPop];
          int pomocnePR[][][]=new int[tym[3]][projekce[0].length-POV[3]][velPop];
          int pomocneS[][][]=new int[tym[4]][stavbyvedouci[0].length-
POV[4]][velPop];
          int pomocneU[][][]=new int[tym[5]][ucetni[0].length-POV[5]][velPop];
          //procentuální rozložení činností v pracovních skupinách
          double procentaL[] = {0.1, 0.2, 0.3, 0.15, 0.1, 0.1, 0.05};
          double procentaO[] = {0.15, 0.2, 0.2, 0.2, 0.1, 0.1, 0.05};
          double procentaPE[] = {0.1, 0.3, 0.3, 0.05, 0.1, 0.1, 0.05};
          double procentaPR[] = {0.5, 0.2, 0.15, 0.15, 0.1, 0.1, 0.05};
          double procentaS[] = {0.05, 0.05, 0.35, 0.3, 0.1, 0.1, 0.05};
          double procentaU[] = {0.1, 0.2, 0.4, 0.05, 0.1, 0.1, 0.05};
          //naplnění 3D pomocných polí dle určitého klíče z 3D polí inicializace,
5. metoda

pomocneL=plneniPomocPoli(velPop,inicializaceL,POV[0],pomocneL,pozadovaneL);

pomocneO=plneniPomocPoli(velPop,inicializaceO,POV[1],pomocneO,pozadovaneO);

pomocnePE=plneniPomocPoli(velPop,inicializacePE,POV[2],pomocnePE,pozadovanePE);

pomocnePR=plneniPomocPoli(velPop,inicializacePR,POV[3],pomocnePR,pozadovanePR);

pomocneS=plneniPomocPoli(velPop,inicializaceS,POV[4],pomocneS,pozadovaneS);

pomocneU=plneniPomocPoli(velPop,inicializaceU,POV[5],pomocneU,pozadovaneU);
          //***** zacatek vypisu 5
          //System.out.println("** Vypis pole pomocneL **");
          //vypis3DpoleINT (pomocneL);
          //System.out.println();
          //System.out.println("** Vypis pole pomocneO **");
          //vypis3DpoleINT (pomocneO);
          //System.out.println();
          //System.out.println("** Vypis pole pomocnePE **");
          //vypis3DpoleINT (pomocnePE);
          //System.out.println();
          //System.out.println("** Vypis pole pomocnePR **");
          //vypis3DpoleINT (pomocnePR);
          //System.out.println();
          //System.out.println("** Vypis pole pomocneS **");
          //vypis3DpoleINT (pomocneS);
          //System.out.println();
          //System.out.println("** Vypis pole pomocneU **");
```

```
//vypis3DpoleINT (pomocneU);
//System.out.println();
//***** konec vypisu 5
//stanovení počtu požadovaných známek z procentuálního rozložení činností
double PPZ1[] = new double[inicializaceL[0].length-POV[0]];
double PPZ2[] = new double[inicializaceO[0].length-POV[1]];
double PPZ3[] = new double[inicializacePE[0].length-POV[2]];
double PPZ4[] = new double[inicializacePR[0].length-POV[3]];
double PPZ5[] = new double[inicializaceS[0].length-POV[4]];
double PPZ6[] = new double[inicializaceU[0].length-POV[5]];
//přiřazení hodnot do polí PPZ1-6
PPZ1=prirazeniHodnot(inicializaceL,velPop,POV[0],PPZ1,procentaL);
PPZ2=prirazeniHodnot(inicializaceO,velPop,POV[1],PPZ2,procentaO);
PPZ3=prirazeniHodnot(inicializacePE,velPop,POV[2],PPZ3,procentaPE);
PPZ4=prirazeniHodnot(inicializacePR,velPop,POV[3],PPZ4,procentaPR);
PPZ5=prirazeniHodnot(inicializaceS,velPop,POV[4],PPZ5,procentaS);
PPZ6=prirazeniHodnot(inicializaceU,velPop,POV[5],PPZ6,procentaU);
//***** zacatek vypisu 6
//System.out.println("** Vypis pole PPZ1 **");
//vypis1DpoleDOUBLE (PPZ1);
//System.out.println("\n");
//System.out.println("** Vypis pole PPZ2 **");
//vypis1DpoleDOUBLE (PPZ2);
//System.out.println("\n");
//System.out.println("** Vypis pole PPZ3 **");
//vypis1DpoleDOUBLE (PPZ3);
//System.out.println("\n");
//System.out.println("** Vypis pole PPZ4 **");
//vypis1DpoleDOUBLE (PPZ4);
//System.out.println("\n");
//System.out.println("** Vypis pole PPZ5 **");
//vypis1DpoleDOUBLE (PPZ5);
//System.out.println("\n");
//System.out.println("** Vypis pole PPZ6 **");
//vypis1DpoleDOUBLE (PPZ6);
//System.out.println("\n");
//***** konec vypisu 6
//stanovení skutečného počtu požadovaných známek
int SPPZ1[][]=new int[velPop][inicializaceL[0].length-POV[0]];
int SPPZ2[][]=new int[velPop][inicializaceO[0].length-POV[1]];
int SPPZ3[][]=new int[velPop][inicializacePE[0].length-POV[2]];
int SPPZ4[][]=new int[velPop][inicializacePR[0].length-POV[3]];
int SPPZ5[][]=new int[velPop][inicializaceS[0].length-POV[4]];
int SPPZ6[][]=new int[velPop][inicializaceU[0].length-POV[5]];
//stanovení skutečného počtu požadovaných známek
SPPZ1=skutPocetZnamek(velPop,inicializaceL,POV[0],pomocneL,SPPZ1);
SPPZ2=skutPocetZnamek(velPop,inicializaceO,POV[1],pomocneO,SPPZ2);
SPPZ3=skutPocetZnamek(velPop,inicializacePE,POV[2],pomocnePE,SPPZ3);
SPPZ4=skutPocetZnamek(velPop,inicializacePR,POV[3],pomocnePR,SPPZ4);
SPPZ5=skutPocetZnamek(velPop,inicializaceS,POV[4],pomocneS,SPPZ5);
SPPZ6=skutPocetZnamek(velPop,inicializaceU,POV[5],pomocneU,SPPZ6);
//***** zacatek vypisu 7
//System.out.println("** Vypis pole SPPZ1 **");
//vypis2DpoleINT (SPPZ1);
//System.out.println();
//System.out.println("** Vypis pole SPPZ2 **");
//vypis2DpoleINT (SPPZ2);
//System.out.println();
//System.out.println("** Vypis pole SPPZ3 **");
//vypis2DpoleINT (SPPZ3);
//System.out.println();
//System.out.println("** Vypis pole SPPZ4 **");
//vypis2DpoleINT (SPPZ4);
//System.out.println();
//System.out.println("** Vypis pole SPPZ5 **");
//vypis2DpoleINT (SPPZ5);
//System.out.println();
//System.out.println("** Vypis pole SPPZ6 **");
```

```
            //vypis2DpoleINT (SPPZ6);
            //System.out.println();
            //***** konec vypisu 7
            //stanovení součtu prvků náhradních polí v řadách
            double sumRadaL[][]=new double[pomocneL.length][velPop];
            double sumRadaO[][]=new double[pomocneO.length][velPop];
            double sumRadaPE[][]=new double[pomocnePE.length][velPop];
            double sumRadaPR[][]=new double[pomocnePR.length][velPop];
            double sumRadaS[][]=new double[pomocneS.length][velPop];
            double sumRadaU[][]=new double[pomocneU.length][velPop];
            //stanovení součtu prvků náhradních polí v řadách
            sumRadaL=souctyPrvkuNahPole(velPop,pomocneL,sumRadaL);
            sumRadaO=souctyPrvkuNahPole(velPop,pomocneO,sumRadaO);
            sumRadaPE=souctyPrvkuNahPole(velPop,pomocnePE,sumRadaPE);
            sumRadaPR=souctyPrvkuNahPole(velPop,pomocnePR,sumRadaPR);
            sumRadaS=souctyPrvkuNahPole(velPop,pomocneS,sumRadaS);
            sumRadaU=souctyPrvkuNahPole(velPop,pomocneU,sumRadaU);
            //***** zacatek vypisu 8
            //System.out.println("** Vypis pole sumRadaL **");
            //vypis2DpoleDOUBLE (sumRadaL);
            //System.out.println();
            //System.out.println("** Vypis pole sumRadaO **");
            //vypis2DpoleDOUBLE (sumRadaO);
            //System.out.println();
            //System.out.println("** Vypis pole sumRadaPE **");
            //vypis2DpoleDOUBLE (sumRadaPE);
            //System.out.println();
            //System.out.println("** Vypis pole sumRadaPR **");
            //vypis2DpoleDOUBLE (sumRadaPR);
            //System.out.println();
            //System.out.println("** Vypis pole sumRadaS **");
            //vypis2DpoleDOUBLE (sumRadaS);
            //System.out.println();
            //System.out.println("** Vypis pole sumRadaU **");
            //vypis2DpoleDOUBLE (sumRadaU);
            //System.out.println();
            //***** konec vypisu 8
            //stanovení průměrných hodnot součtů u jednotlivých skupin
            double stredHodn[][]=new double[6][velPop];
            double pocetZamL=pomocneL.length;
            double pocetZamO=pomocneO.length;
            double pocetZamPE=pomocnePE.length;
            double pocetZamPR=pomocnePR.length;
            double pocetZamS=pomocneS.length;
            double pocetZamU=pomocneU.length;

stredHodn=prumHodnotaSoucet(sumRadaL,velPop,pomocneL,stredHodn,pocetZamL,0);

stredHodn=prumHodnotaSoucet(sumRadaO,velPop,pomocneO,stredHodn,pocetZamO,1);

stredHodn=prumHodnotaSoucet(sumRadaPE,velPop,pomocnePE,stredHodn,pocetZamPE,2);

stredHodn=prumHodnotaSoucet(sumRadaPR,velPop,pomocnePR,stredHodn,pocetZamPR,3);

stredHodn=prumHodnotaSoucet(sumRadaS,velPop,pomocneS,stredHodn,pocetZamS,4);

stredHodn=prumHodnotaSoucet(sumRadaU,velPop,pomocneU,stredHodn,pocetZamU,5);
            //***** zacatek vypisu 9
            //System.out.println("** Vypis pole stredHodn **");
            //vypis2DpoleDOUBLE (stredHodn);
            //System.out.println();
            //***** konec vypisu 9
            //pomocná pole delta[][] - součty odchylek od středních hodnot
            double sumDelta[][]=new double[velPop][6];

sumDelta=souctyOdchylekOdStredHodnot(velPop,pomocneL,sumRadaL,stredHodn,sumDelta,0);

sumDelta=souctyOdchylekOdStredHodnot(velPop,pomocneO,sumRadaO,stredHodn,sumDelta,1);
```

```
sumDelta=souctyOdchylekOdStredHodnot(velPop,pomocnePE,sumRadaPE,stredHodn,sumDelta,2
);

sumDelta=souctyOdchylekOdStredHodnot(velPop,pomocnePR,sumRadaPR,stredHodn,sumDelta,3
);

sumDelta=souctyOdchylekOdStredHodnot(velPop,pomocneS,sumRadaS,stredHodn,sumDelta,4);

sumDelta=souctyOdchylekOdStredHodnot(velPop,pomocneU,sumRadaU,stredHodn,sumDelta,5);
          //***** zacatek vypisu 10
          //System.out.println("** Vypis pole sumDelta **");
          //vypis2DpoleDOUBLE (sumDelta);
          //System.out.println();
          //***** konec vypisu 10
          //stanovení dílčích fitness funkcí FSE
          double vaha1=1.0;
          double vaha2=1.0;

FSE=dilciFitnessFSE(velPop,pomocneL,vaha1,vaha2,sumDelta,PPZ1,SPPZ1,FSE,0);

FSE=dilciFitnessFSE(velPop,pomocneO,vaha1,vaha2,sumDelta,PPZ2,SPPZ2,FSE,1);

FSE=dilciFitnessFSE(velPop,pomocnePE,vaha1,vaha2,sumDelta,PPZ3,SPPZ3,FSE,2);

FSE=dilciFitnessFSE(velPop,pomocnePR,vaha1,vaha2,sumDelta,PPZ4,SPPZ4,FSE,3);

FSE=dilciFitnessFSE(velPop,pomocneS,vaha1,vaha2,sumDelta,PPZ5,SPPZ5,FSE,4);

FSE=dilciFitnessFSE(velPop,pomocneU,vaha1,vaha2,sumDelta,PPZ6,SPPZ6,FSE,5);
          //***** zacatek vypisu 11
          //System.out.println("** Vypis pole FSE **");
          //vypis2DpoleDOUBLE (FSE);
          //System.out.println();
          //***** konec vypisu 11
          //*** stanovení FOV u pracovních skupin ***
          //definice polí, kde budou uloženy průměrné známky osobních vlastností
          double prumVlastnostL[][]=new double[velPop][POV[0]];
          double prumVlastnostO[][]=new double[velPop][POV[1]];
          double prumVlastnostPE[][]=new double[velPop][POV[2]];
          double prumVlastnostPR[][]=new double[velPop][POV[3]];
          double prumVlastnostS[][]=new double[velPop][POV[4]];
          double prumVlastnostU[][]=new double[velPop][POV[5]];
          //určení středních hodnot známek osobních vlastností
          prumVlastnostL=prumVlastnostFOV(velPop, POV, inicializaceL,
prumVlastnostL,0);
          prumVlastnostO=prumVlastnostFOV(velPop, POV, inicializaceO,
prumVlastnostO,1);
          prumVlastnostPE=prumVlastnostFOV(velPop, POV, inicializacePE,
prumVlastnostPE,2);
          prumVlastnostPR=prumVlastnostFOV(velPop, POV, inicializacePR,
prumVlastnostPR,3);
          prumVlastnostS=prumVlastnostFOV(velPop, POV, inicializaceS,
prumVlastnostS,4);
          prumVlastnostU=prumVlastnostFOV(velPop, POV, inicializaceU,
prumVlastnostU,5);
          //***** zacatek vypisu 12
          //System.out.println("** Vypis pole prumVlastnostL **");
          //vypis2DpoleDOUBLE (prumVlastnostL);
          //System.out.println();
          //System.out.println("** Vypis pole prumVlastnostO **");
          //vypis2DpoleDOUBLE (prumVlastnostO);
          //System.out.println();
          //System.out.println("** Vypis pole prumVlastnostPE **");
          //vypis2DpoleDOUBLE (prumVlastnostPE);
          //System.out.println();
          //System.out.println("** Vypis pole prumVlastnostPR **");
          //vypis2DpoleDOUBLE (prumVlastnostPR);
```

```
            //System.out.println();
            //System.out.println("** Vypis pole prumVlastnostS **");
            //vypis2DpoleDOUBLE (prumVlastnostS);
            //System.out.println();
            //System.out.println("** Vypis pole prumVlastnostU **");
            //vypis2DpoleDOUBLE (prumVlastnostU);
            //System.out.println();
            //***** konec vypisu 12
            //fitness FOV
            FOV=dilciFitnessFOV(velPop,POV,FOV,prumVlastnostL,pozadovaneL,0);
            FOV=dilciFitnessFOV(velPop,POV,FOV,prumVlastnostO,pozadovaneO,1);
            FOV=dilciFitnessFOV(velPop,POV,FOV,prumVlastnostPE,pozadovanePE,2);
            FOV=dilciFitnessFOV(velPop,POV,FOV,prumVlastnostPR,pozadovanePR,3);
            FOV=dilciFitnessFOV(velPop,POV,FOV,prumVlastnostS,pozadovaneS,4);
            FOV=dilciFitnessFOV(velPop,POV,FOV,prumVlastnostU,pozadovaneU,5);
            //***** zacatek vypisu 13
            //System.out.println("** Vypis pole FOV **");
            //vypis2DpoleDOUBLE (FOV);
            //System.out.println();
            //***** konec vypisu 13
            //dílčí fitness funkce FZZ, FSE a FOV roznásobené váhami
            double vahyFZZ[][]=new double[velPop][6];
            double vahyFSE[][]=new double[velPop][6];
            double vahyFOV[][]=new double[velPop][6];
            vahyFZZ=roznasobeniVahamiFZZ (velPop,tym,vahyFZZ,vahy,FZZ);
            vahyFSE=roznasobeniVahamiFSE (velPop,tym,vahyFSE,vahy,FSE);
            vahyFOV=roznasobeniVahamiFOV (velPop,tym,vahyFOV,vahy,FOV);
            //***** zacatek vypisu 14
            //System.out.println("** Vypis pole vahyFZZ **");
            //vypis2DpoleDOUBLE (vahyFZZ);
            //System.out.println();
            //System.out.println("** Vypis pole vahyFSE **");
            //vypis2DpoleDOUBLE (vahyFSE);
            //System.out.println();
            //System.out.println("** Vypis pole vahyFOV **");
            //vypis2DpoleDOUBLE (vahyFOV);
            //System.out.println();
            //***** konec vypisu 14
            //součet dílčích funkcí fitness roznásobených váhami
            double sumaDilciFitness[][]=new double[velPop][3];
            sumaDilciFitness=sumaDilciFitness
(velPop,tym,sumaDilciFitness,vahyFZZ,vahyFSE,vahyFOV);
            //***** zacatek vypisu 15
            //System.out.println("** Vypis pole sumaDilciFitness **");
            //vypis2DpoleDOUBLE (sumaDilciFitness);
            //System.out.println();
            //***** konec vypisu 15
            //* stanoveni fitness funkce pracovniho tymu *
            double FT[]=new double[velPop];
            FT=fitnessFunkceTymu (FT,velPop,sumaDilciFitness);
            //***** zacatek vypisu 16
            //System.out.println("** Vypis pole FT **");
            //vypis1DpoleDOUBLE (FT);
            //System.out.println("\n");
            //***** konec vypisu 16


            //*** 6. VYHODNOCENÍ POPULACE ***
            //1. prům. fitness populace, 2. rozptyl populace, 3. kód nejlepší
populace
            //1. průměrná fitness populace
            double prumFitnessPopulace=0;
            prumFitnessPopulace=prumFitnessPopulace (prumFitnessPopulace,velPop,FT);
            //***** zacatek vypisu 16b
            //System.out.println("** Vypis prumFitnessPopulace **");
            //System.out.println(prumFitnessPopulace);
            //***** konec vypisu 16b
            //2. rozptyl populace
```

```
            double rozptyl=0;
            rozptyl=rozptylPopulace (rozptyl, velPop, prumFitnessPopulace,FT);
            //***** zacatek vypisu 16c
            //System.out.println("** Vypis rozptyl **");
            //System.out.println(rozptyl);
            //***** konec vypisu 16c
            //3. nejlepší fitness v populaci
            double setrideneFT[]=new double[velPop];
            double nejFT = setrideneFT[0];  //minimum FT je nejlepsi
            nejFT=nejFitnessPopulace (setrideneFT, FT, nejFT);
            //***** zacatek vypisu 16d
            //System.out.println("** Vypis nejFT **");
            //System.out.println(nejFT);
            //***** konec vypisu 16d
            //4. vyhledani nejlepsiho jedince dle nej FT a ulozeni jeho kodu do pole
nejJedinec[]
            //vytvoření 2D polí castNejJedince[][], zde bude uložen kód nejlepšího
jedince z populace
            int castNejJedinceL[][]=new int[tym[0]][logistika[0].length];
            int castNejJedinceO[][]=new int[tym[1]][obchod[0].length];
            int castNejJedincePE[][]=new int[tym[2]][personalistika[0].length];
            int castNejJedincePR[][]=new int[tym[3]][projekce[0].length];
            int castNejJedinceS[][]=new int[tym[4]][stavbyvedouci[0].length];
            int castNejJedinceU[][]=new int[tym[5]][ucetni[0].length];
            //určení pořadí nejlepšího jedince
            int poradiNejJedince=0;
            castNejJedinceL=ulozeniNejJedince
(velPop,castNejJedinceL,inicializaceL,poradiNejJedince,nejFT,FT);
            castNejJedinceO=ulozeniNejJedince
(velPop,castNejJedinceO,inicializaceO,poradiNejJedince,nejFT,FT);
            castNejJedincePE=ulozeniNejJedince
(velPop,castNejJedincePE,inicializacePE,poradiNejJedince,nejFT,FT);
            castNejJedincePR=ulozeniNejJedince
(velPop,castNejJedincePR,inicializacePR,poradiNejJedince,nejFT,FT);
            castNejJedinceS=ulozeniNejJedince
(velPop,castNejJedinceS,inicializaceS,poradiNejJedince,nejFT,FT);
            castNejJedinceU=ulozeniNejJedince
(velPop,castNejJedinceU,inicializaceU,poradiNejJedince,nejFT,FT);
            //***** zacatek vypisu 17
            //System.out.println("** Vypis pole castNejJedinceL **");
            //vypis2DpoleINT (castNejJedinceL);
            //System.out.println();
            //System.out.println("** Vypis pole castNejJedinceO **");
            //vypis2DpoleINT (castNejJedinceO);
            //System.out.println();
            //System.out.println("** Vypis pole castNejJedincePE **");
            //vypis2DpoleINT (castNejJedincePE);
            //System.out.println();
            //System.out.println("** Vypis pole castNejJedincePR **");
            //vypis2DpoleINT (castNejJedincePR);
            //System.out.println();
            //System.out.println("** Vypis pole castNejJedinceS **");
            //vypis2DpoleINT (castNejJedinceS);
            //System.out.println();
            //System.out.println("** Vypis pole castNejJedinceU **");
            //vypis2DpoleINT (castNejJedinceU);
            //System.out.println();
            //***** konec vypisu 17


            //*** 7. SOUHRNÉ VYHODNOCENÍ POPULACE ***
            //uložení průměrné fitness, rozptylu a nej fitness u populace do 2D pole
            int pocetIteraci=100;
            double souhrneVysledky[][]=new double [pocetIteraci+1][3];
            souhrneVysledky[0][0]=prumFitnessPopulace;
            souhrneVysledky[0][1]=rozptyl;
            souhrneVysledky[0][2]=nejFT;
            //*** SOUHRNE VYSLEDKY STATISTIKY ***
```

```
          double souhrneVysledkyGlobal[][][]=new double [pocetIteraci+1][3][povy];
          //3D pole, kde budou uloženy kódy nejlepších jedinců
          int nejJedinciL[][][]=new
int[tym[0]][logistika[0].length][pocetIteraci+1];
          int nejJedinciO[][][]=new int[tym[1]][obchod[0].length][pocetIteraci+1];
          int nejJedinciPE[][][]=new
int[tym[2]][personalistika[0].length][pocetIteraci+1];
          int nejJedinciPR[][][]=new
int[tym[3]][projekce[0].length][pocetIteraci+1];
          int nejJedinciS[][][]=new
int[tym[4]][stavbyvedouci[0].length][pocetIteraci+1];
          int nejJedinciU[][][]=new int[tym[5]][ucetni[0].length][pocetIteraci+1];
          //uložení kódů nejlepších jedinců
          for (int i=0;i<tym[0];i++) {
              for (int j=0;j<logistika[0].length;j++) {
                  nejJedinciL[i][j][0]=castNejJedinceL[i][j];
              }
          }
          for (int i=0;i<tym[1];i++) {
              for (int j=0;j<obchod[0].length;j++) {
                  nejJedinciO[i][j][0]=castNejJedinceO[i][j];
              }
          }
          for (int i=0;i<tym[2];i++) {
              for (int j=0;j<personalistika[0].length;j++) {
                  nejJedinciPE[i][j][0]=castNejJedincePE[i][j];
              }
          }
          for (int i=0;i<tym[3];i++) {
              for (int j=0;j<projekce[0].length;j++) {
                  nejJedinciPR[i][j][0]=castNejJedincePR[i][j];
              }
          }
          for (int i=0;i<tym[4];i++) {
              for (int j=0;j<stavbyvedouci[0].length;j++) {
                  nejJedinciS[i][j][0]=castNejJedinceS[i][j];
              }
          }
          for (int i=0;i<tym[5];i++) {
              for (int j=0;j<ucetni[0].length;j++) {
                  nejJedinciU[i][j][0]=castNejJedinceU[i][j];
              }
          }
          //***** zacatek vypisu 18
          //System.out.println("** Vypis pole nejJedinciL **");
          //vypis3DpoleINT (nejJedinciL);
          //System.out.println();
          //System.out.println("** Vypis pole nejJedinciO **");
          //vypis3DpoleINT (nejJedinciO);
          //System.out.println();
          //System.out.println("** Vypis pole nejJedinciPE **");
          //vypis3DpoleINT (nejJedinciPE);
          //System.out.println();
          //System.out.println("** Vypis pole nejJedinciPR **");
          //vypis3DpoleINT (nejJedinciPR);
          //System.out.println();
          //System.out.println("** Vypis pole nejJedinciS **");
          //vypis3DpoleINT (nejJedinciS);
          //System.out.println();
          //System.out.println("** Vypis pole nejJedinciU **");
          //vypis3DpoleINT (nejJedinciU);
          //System.out.println();
          //***** konec vypisu 18
          int mutaceL[][][]=new int[tym[0]][logistika[0].length][velPop];
          int mutaceO[][][]=new int[tym[1]][obchod[0].length][velPop];
          int mutacePE[][][]=new int[tym[2]][personalistika[0].length][velPop];
          int mutacePR[][][]=new int[tym[3]][projekce[0].length][velPop];
          int mutaceS[][][]=new int[tym[4]][stavbyvedouci[0].length][velPop];
```

```
          int mutaceU[][][]=new int[tym[5]][ucetni[0].length][velPop];
          int selektovanaPopulaceL[][][]=new
int[tym[0]][logistika[0].length][velPop];
          int selektovanaPopulaceO[][][]=new int[tym[1]][obchod[0].length][velPop];
          int selektovanaPopulacePE[][][]=new
int[tym[2]][personalistika[0].length][velPop];
          int selektovanaPopulacePR[][][]=new
int[tym[3]][projekce[0].length][velPop];
          int selektovanaPopulaceS[][][]=new
int[tym[4]][stavbyvedouci[0].length][velPop];
          int selektovanaPopulaceU[][][]=new int[tym[5]][ucetni[0].length][velPop];


              //**********************************
              //***** VSTUP DO CYKLU PROGRAMU *****
              //**********************************
              //počet cyklů, které program provede určuje proměnná pocetIteraci
              for (int pc=0;pc<pocetIteraci;pc++) {
              //System.out.println("\n"+"***** Cyklus cislo ***** " +(c+1)+"\n");


              //*** 7. SELEKCE ***
              //** v programu je implementovan turnajovy mechanismus
              //* počet jedinců vstupujících do turnaje
              int jedinciVturnaji=2;
              //* pomocne pole (do jednoho turnaje nemůže jít nikdo 2x)
              int pomPolePocitadlo[]=new int[velPop];
              //* náhodné losování jedinců do turnaje a vyplneni soutezniho pole
              int soutezniPole[][]=new int[velPop][jedinciVturnaji];
              for (int i=0;i<velPop;i++) {
                  Arrays.fill(pomPolePocitadlo,1); //naplneni pomocneho pole
jednickami
                  for (int j=0;j<jedinciVturnaji;j++) {
                      int losovaniJedincu=rand.nextInt(velPop);
                      if (pomPolePocitadlo[losovaniJedincu]==1) { //pokud neni
jedinec jiz v turnaji, bude vybran
                      soutezniPole[i][j]=losovaniJedincu;
                      pomPolePocitadlo[losovaniJedincu]=0;
                      }
                      else
                          j--;
                  }
              }
              //***** zacatek vypisu 19
              //System.out.println("** Vypis pole soutezniPole **");
              //vypis2DpoleINT (soutezniPole) ;
              //System.out.println();
              //***** konec vypisu 19
              //* výběr soutěžících z jednoho turnaje, jejich setřídění, výběr
nejlepšího do pole postupující *
              int postupujici[]=new int[velPop];
              double soutezniPoleFitness[][]=new double[velPop][jedinciVturnaji];
              //přiřazení odpovídajících fitness dle soutěžících do 2D pole
soutezniPoleFitness[][]
              for (int i=0;i<velPop;i++) {
                  for (int j=0;j<jedinciVturnaji;j++) {
                      soutezniPoleFitness[i][j]=FT[soutezniPole[i][j]];
                  }
              }
              //***** zacatek vypisu 20
              //System.out.println("** Vypis pole soutezniPoleFitness **");
              //vypis2DpoleDOUBLE (soutezniPoleFitness);
              //System.out.println();
              //***** konec vypisu 20
              //výběr soutěžících z jednoho turnaje, jejich setřídění a stanovení
postupujícího
              double tridiciPole[]=new double[jedinciVturnaji];
              for (int i=0;i<velPop;i++) {
```

```
                    for (int j=0;j<jedinciVturnaji;j++) {
                        tridiciPole[j]=soutezniPoleFitness[i][j];
                    }
                    Arrays.sort(tridiciPole);  //setrideni pole
                    for (int j=0;j<jedinciVturnaji;j++) {
                        if (tridiciPole[0]==soutezniPoleFitness[i][j]) {
                            postupujici[i]=soutezniPole[i][j];
                        }
                    }
                }
                //***** zacatek vypisu 21
                //System.out.println("** Vypis pole postupujici **");
                //vypis1DpoleINT (postupujici);
                //System.out.println();
                //***** konec vypisu 21
                //***** zacatek vypisu 22
                //System.out.println("** Vypis pole tridiciPole **");
                //vypis1DpoleDOUBLE (tridiciPole);
                //System.out.println();
                //***** konec vypisu 22
                //vytvoření selektované populace
                //naplnění 3D polí selektovanaPopulace[][][] pomocí 14. metody

selektovanaPopulaceL=selektovaniPopulace(velPop,logistika,tym[0],selektovanaPopulace
L,inicializaceL,postupujici,mutaceL,pc);

selektovanaPopulaceO=selektovaniPopulace(velPop,obchod,tym[1],selektovanaPopulaceO,i
nicializaceO,postupujici,mutaceO,pc);

selektovanaPopulacePE=selektovaniPopulace(velPop,personalistika,tym[2],selektovanaPo
pulacePE,inicializacePE,postupujici,mutacePE,pc);

selektovanaPopulacePR=selektovaniPopulace(velPop,projekce,tym[3],selektovanaPopulace
PR,inicializacePR,postupujici,mutacePR,pc);

selektovanaPopulaceS=selektovaniPopulace(velPop,stavbyvedouci,tym[4],selektovanaPopu
laceS,inicializaceS,postupujici,mutaceS,pc);

selektovanaPopulaceU=selektovaniPopulace(velPop,ucetni,tym[5],selektovanaPopulaceU,i
nicializaceU,postupujici,mutaceU,pc);
                //***** zacatek vypisu 23
                //System.out.println("** Vypis pole selektovanaPopulaceL **");
                //vypis3DpoleINT (selektovanaPopulaceL);
                //System.out.println();
                //System.out.println("** Vypis pole selektovanaPopulaceO **");
                //vypis3DpoleINT (selektovanaPopulaceO);
                //System.out.println();
                //System.out.println("** Vypis pole selektovanaPopulacePE **");
                //vypis3DpoleINT (selektovanaPopulacePE);
                //System.out.println();
                //System.out.println("** Vypis pole selektovanaPopulacePR **");
                //vypis3DpoleINT (selektovanaPopulacePR);
                //System.out.println();
                //System.out.println("** Vypis pole selektovanaPopulaceS **");
                //vypis3DpoleINT (selektovanaPopulaceS);
                //System.out.println();
                //System.out.println("** Vypis pole selektovanaPopulaceU **");
                //vypis3DpoleINT (selektovanaPopulaceU);
                //System.out.println();
                //***** konec vypisu 23

                //*** 8. MUTACE ***
                //pocet zamestnancu v pracovnim tymu
                mutaceL=mutovaniPopulace(velPop, logistika, selektovanaPopulaceL,
Pm,tym[0]);
                mutaceO=mutovaniPopulace(velPop, obchod, selektovanaPopulaceO, Pm,
tym[1]);
```

```
                mutacePE=mutovaniPopulace(velPop, personalistika,
selektovanaPopulacePE, Pm,tym[2]);
                mutacePR=mutovaniPopulace(velPop, projekce, selektovanaPopulacePR,
Pm,tym[3]);
                mutaceS=mutovaniPopulace(velPop, stavbyvedouci, selektovanaPopulaceS,
Pm,tym[4]);
                mutaceU=mutovaniPopulace(velPop, ucetni, selektovanaPopulaceU,
Pm,tym[5]);
                //***** zacetek vypisu 24
                //System.out.println("** Vypis pole mutaceL **");
                //vypis3DpoleINT (mutaceL);
                //System.out.println();
                //System.out.println("** Vypis pole mutaceO **");
                //vypis3DpoleINT (mutaceO);
                //System.out.println();
                //System.out.println("** Vypis pole mutacePE **");
                //vypis3DpoleINT (mutacePE);
                //System.out.println();
                //System.out.println("** Vypis pole mutacePR **");
                //vypis3DpoleINT (mutacePR);
                //System.out.println();
                //System.out.println("** Vypis pole mutaceS **");
                //vypis3DpoleINT (mutaceS);
                //System.out.println();
                //System.out.println("** Vypis pole mutaceU **");
                //vypis3DpoleINT (mutaceU);
                //System.out.println();
                //***** konec vypisu 24


                //*** 9. URČENÍ FITNESS FUNKCE ZMĚNĚNÉ POPULACE ***
                //využití 2. metody ke stanovení součtu známek u pracovních skupin,
2. metoda
                SZZZ1=souctyZnamekSkupina(SZZZ1,mutaceL,velPop);
                SZZZ2=souctyZnamekSkupina(SZZZ2,mutaceO,velPop);
                SZZZ3=souctyZnamekSkupina(SZZZ3,mutacePE,velPop);
                SZZZ4=souctyZnamekSkupina(SZZZ4,mutacePR,velPop);
                SZZZ5=souctyZnamekSkupina(SZZZ5,mutaceS,velPop);
                SZZZ6=souctyZnamekSkupina(SZZZ6,mutaceU,velPop);
                //***** zacatek vypisu 25
                //System.out.println("** Vypis pole SZZZ1 **");
                //vypis2DpoleINT (SZZZ1);
                //System.out.println();
                //System.out.println("** Vypis pole SZZZ2 **");
                //vypis2DpoleINT (SZZZ2);
                //System.out.println();
                //System.out.println("** Vypis pole SZZZ3 **");
                //vypis2DpoleINT (SZZZ3);
                //System.out.println();
                //System.out.println("** Vypis pole SZZZ4 **");
                //vypis2DpoleINT (SZZZ4);
                //System.out.println();
                //System.out.println("** Vypis pole SZZZ5 **");
                //vypis2DpoleINT (SZZZ5);
                //System.out.println();
                //System.out.println("** Vypis pole SZZZ6 **");
                //vypis2DpoleINT (SZZZ6);
                //System.out.println();
                //***** konec vypisu 25
                //* určení dílčích fitness funkcí FZZ, 4. metoda *
                //vynulovani pole FZZ
                for (int q=0;q<FZZ[0].length;q++) {
                    for (int w=0;w<FZZ.length;w++) {
                        FZZ[w][q]=0.0;
                    }
                }

FZZ=urceniFunkceFZZ(FZZ,SZZZ1,mutaceL,logistika,POV[0],pozadovaneL,velPop,0);
```

```
FZZ=urceniFunkceFZZ(FZZ,SZZZ2,mutaceO,obchod,POV[1],pozadovaneO,velPop,1);

FZZ=urceniFunkceFZZ(FZZ,SZZZ3,mutacePE,personalistika,POV[2],pozadovanePE,velPop,2);

FZZ=urceniFunkceFZZ(FZZ,SZZZ4,mutacePR,projekce,POV[3],pozadovanePR,velPop,3);

FZZ=urceniFunkceFZZ(FZZ,SZZZ5,mutaceS,stavbyvedouci,POV[4],pozadovaneS,velPop,4);

FZZ=urceniFunkceFZZ(FZZ,SZZZ6,mutaceU,ucetni,POV[5],pozadovaneU,velPop,5);
               //***** zacatek vypisu 26
               //System.out.println("** Vypis pole FZZ **");
               //vypis2DpoleDOUBLE (FZZ);
               //System.out.println();
               //***** konec vypisu 26
               //* určení dílčích fitness funkcí FSE, 11. metoda *
               //naplnění 3D pomocných polí dle určitého klíče z 3D polí
inicializace, 5. metoda
               //vynulovani pole pomocne
               for (int i=0;i<pomocneL.length;i++) {
                   for (int j=0;j<pomocneL[0].length;j++) {
                       for (int k=0;k<pomocneL[0][0].length;k++) {
                           pomocneL[i][j][k]=0;
                       }
                   }
               }
               for (int i=0;i<pomocneO.length;i++) {
                   for (int j=0;j<pomocneO[0].length;j++) {
                       for (int k=0;k<pomocneO[0][0].length;k++) {
                           pomocneO[i][j][k]=0;
                       }
                   }
               }
               for (int i=0;i<pomocnePE.length;i++) {
                   for (int j=0;j<pomocnePE[0].length;j++) {
                       for (int k=0;k<pomocnePE[0][0].length;k++) {
                           pomocnePE[i][j][k]=0;
                       }
                   }
               }
               for (int i=0;i<pomocnePR.length;i++) {
                   for (int j=0;j<pomocnePR[0].length;j++) {
                       for (int k=0;k<pomocnePR[0][0].length;k++) {
                           pomocnePR[i][j][k]=0;
                       }
                   }
               }
               for (int i=0;i<pomocneS.length;i++) {
                   for (int j=0;j<pomocneS[0].length;j++) {
                       for (int k=0;k<pomocneS[0][0].length;k++) {
                           pomocneS[i][j][k]=0;
                       }
                   }
               }
               for (int i=0;i<pomocneU.length;i++) {
                   for (int j=0;j<pomocneU[0].length;j++) {
                       for (int k=0;k<pomocneU[0][0].length;k++) {
                           pomocneU[i][j][k]=0;
                       }
                   }
               }
               pomocneL=plneniPomocPoli(velPop,mutaceL,POV[0],pomocneL,pozadovaneL);
               pomocneO=plneniPomocPoli(velPop,mutaceO,POV[1],pomocneO,pozadovaneO);

pomocnePE=plneniPomocPoli(velPop,mutacePE,POV[2],pomocnePE,pozadovanePE);

pomocnePR=plneniPomocPoli(velPop,mutacePR,POV[3],pomocnePR,pozadovanePR);
               pomocneS=plneniPomocPoli(velPop,mutaceS,POV[4],pomocneS,pozadovaneS);
```

```
pomocneU=plneniPomocPoli(velPop,mutaceU,POV[5],pomocneU,pozadovaneU);
//***** zacatek vypisu 27
//System.out.println("** Vypis pole pomocneL **");
//vypis3DpoleINT (pomocneL);
//System.out.println();
//System.out.println("** Vypis pole pomocneO **");
//vypis3DpoleINT (pomocneO);
//System.out.println();
//System.out.println("** Vypis pole pomocnePE **");
//vypis3DpoleINT (pomocnePE);
//System.out.println();
//System.out.println("** Vypis pole pomocnePR **");
//vypis3DpoleINT (pomocnePR);
//System.out.println();
//System.out.println("** Vypis pole pomocneS **");
//vypis3DpoleINT (pomocneS);
//System.out.println();
//System.out.println("** Vypis pole pomocneU **");
//vypis3DpoleINT (pomocneU);
//System.out.println();
//***** konec vypisu 27
//nulovani poli
for (int i=0;i<SPPZ1.length;i++) {
    for (int j=0;j<SPPZ1[0].length;j++) {
        SPPZ1[i][j]=0;
    }
}
for (int i=0;i<SPPZ2.length;i++) {
    for (int j=0;j<SPPZ2[0].length;j++) {
        SPPZ2[i][j]=0;
    }
}
for (int i=0;i<SPPZ3.length;i++) {
    for (int j=0;j<SPPZ3[0].length;j++) {
        SPPZ3[i][j]=0;
    }
}
for (int i=0;i<SPPZ4.length;i++) {
    for (int j=0;j<SPPZ4[0].length;j++) {
        SPPZ4[i][j]=0;
    }
}
for (int i=0;i<SPPZ5.length;i++) {
    for (int j=0;j<SPPZ5[0].length;j++) {
        SPPZ5[i][j]=0;
    }
}
for (int i=0;i<SPPZ6.length;i++) {
    for (int j=0;j<SPPZ6[0].length;j++) {
        SPPZ6[i][j]=0;
    }
}
//stanovení skutečného počtu požadovaných známek, 7. metoda
SPPZ1=skutPocetZnamek(velPop,mutaceL,POV[0],pomocneL,SPPZ1);
SPPZ2=skutPocetZnamek(velPop,mutaceO,POV[1],pomocneO,SPPZ2);
SPPZ3=skutPocetZnamek(velPop,mutacePE,POV[2],pomocnePE,SPPZ3);
SPPZ4=skutPocetZnamek(velPop,mutacePR,POV[3],pomocnePR,SPPZ4);
SPPZ5=skutPocetZnamek(velPop,mutaceS,POV[4],pomocneS,SPPZ5);
SPPZ6=skutPocetZnamek(velPop,mutaceU,POV[5],pomocneU,SPPZ6);
//***** zacatek vypisu 29
//System.out.println("** Vypis pole SPPZ1 **");
//vypis2DpoleINT (SPPZ1);
//System.out.println();
//System.out.println("** Vypis pole SPPZ2 **");
//vypis2DpoleINT (SPPZ2);
//System.out.println();
//System.out.println("** Vypis pole SPPZ3 **");
//vypis2DpoleINT (SPPZ3);
```

```
//System.out.println();
//System.out.println("** Vypis pole SPPZ4 **");
//vypis2DpoleINT (SPPZ4);
//System.out.println();
//System.out.println("** Vypis pole SPPZ5 **");
//vypis2DpoleINT (SPPZ5);
//System.out.println();
//System.out.println("** Vypis pole SPPZ6 **");
//vypis2DpoleINT (SPPZ6);
//System.out.println();
//***** konec vypisu 29
//stanovení součtu prvků náhradních polí v řadách, 8. metoda
//nulovani pole
for (int i=0;i<sumRadaL.length;i++) {
    for (int j=0;j<sumRadaL[0].length;j++) {
        sumRadaL[i][j]=0;
    }
}
for (int i=0;i<sumRadaO.length;i++) {
    for (int j=0;j<sumRadaO[0].length;j++) {
        sumRadaO[i][j]=0;
    }
}
for (int i=0;i<sumRadaPE.length;i++) {
    for (int j=0;j<sumRadaPE[0].length;j++) {
        sumRadaPE[i][j]=0;
    }
}
for (int i=0;i<sumRadaPR.length;i++) {
    for (int j=0;j<sumRadaPR[0].length;j++) {
        sumRadaPR[i][j]=0;
    }
}
for (int i=0;i<sumRadaS.length;i++) {
    for (int j=0;j<sumRadaS[0].length;j++) {
        sumRadaS[i][j]=0;
    }
}
for (int i=0;i<sumRadaU.length;i++) {
    for (int j=0;j<sumRadaU[0].length;j++) {
        sumRadaU[i][j]=0;
    }
}
sumRadaL=souctyPrvkuNahPole(velPop,pomocneL,sumRadaL);
sumRadaO=souctyPrvkuNahPole(velPop,pomocneO,sumRadaO);
sumRadaPE=souctyPrvkuNahPole(velPop,pomocnePE,sumRadaPE);
sumRadaPR=souctyPrvkuNahPole(velPop,pomocnePR,sumRadaPR);
sumRadaS=souctyPrvkuNahPole(velPop,pomocneS,sumRadaS);
sumRadaU=souctyPrvkuNahPole(velPop,pomocneU,sumRadaU);
//***** zacatek vypisu 30
//System.out.println("** Vypis pole sumRadaL **");
//vypis2DpoleDOUBLE (sumRadaL);
//System.out.println();
//System.out.println("** Vypis pole sumRadaO **");
//vypis2DpoleDOUBLE (sumRadaO);
//System.out.println();
//System.out.println("** Vypis pole sumRadaPE **");
//vypis2DpoleDOUBLE (sumRadaPE);
//System.out.println();
//System.out.println("** Vypis pole sumRadaPR **");
//vypis2DpoleDOUBLE (sumRadaPR);
//System.out.println();
//System.out.println("** Vypis pole sumRadaS **");
//vypis2DpoleDOUBLE (sumRadaS);
//System.out.println();
//System.out.println("** Vypis pole sumRadaU **");
//vypis2DpoleDOUBLE (sumRadaU);
//System.out.println();
```

```
                //***** konec vypisu 30
                //stanovení průměrných hodnot součtů u jednotlivých skupin, 9. metoda
                //nulování pole stredHodn
                for (int i=0;i<stredHodn.length;i++) {
                    for (int j=0;j<stredHodn[0].length;j++) {
                        stredHodn[i][j]=0;
                    }
                }

stredHodn=prumHodnotaSoucet(sumRadaL,velPop,pomocneL,stredHodn,pocetZamL,0);

stredHodn=prumHodnotaSoucet(sumRadaO,velPop,pomocneO,stredHodn,pocetZamO,1);

stredHodn=prumHodnotaSoucet(sumRadaPE,velPop,pomocnePE,stredHodn,pocetZamPE,2);

stredHodn=prumHodnotaSoucet(sumRadaPR,velPop,pomocnePR,stredHodn,pocetZamPR,3);

stredHodn=prumHodnotaSoucet(sumRadaS,velPop,pomocneS,stredHodn,pocetZamS,4);

stredHodn=prumHodnotaSoucet(sumRadaU,velPop,pomocneU,stredHodn,pocetZamU,5);
                //***** zacatek vypisu 31
                //System.out.println("** Vypis pole stredHodn **");
                //vypis2DpoleDOUBLE (stredHodn);
                //System.out.println();
                //***** konec vypisu 31
                //pomocná pole delta[][] - součty odchylek od středních hodnot, 10.
metoda
                //vynulovani pole
                for (int i=0;i<sumDelta.length;i++) {
                    for (int j=0;j<sumDelta[0].length;j++) {
                        sumDelta[i][j]=0;
                    }
                }

sumDelta=souctyOdchylekOdStredHodnot(velPop,pomocneL,sumRadaL,stredHodn,sumDelta,0);

sumDelta=souctyOdchylekOdStredHodnot(velPop,pomocneO,sumRadaO,stredHodn,sumDelta,1);

sumDelta=souctyOdchylekOdStredHodnot(velPop,pomocnePE,sumRadaPE,stredHodn,sumDelta,2
);

sumDelta=souctyOdchylekOdStredHodnot(velPop,pomocnePR,sumRadaPR,stredHodn,sumDelta,3
);

sumDelta=souctyOdchylekOdStredHodnot(velPop,pomocneS,sumRadaS,stredHodn,sumDelta,4);

sumDelta=souctyOdchylekOdStredHodnot(velPop,pomocneU,sumRadaU,stredHodn,sumDelta,5);
                //***** zacatek vypisu 32
                //System.out.println("** Vypis pole sumDelta **");
                //vypis2DpoleDOUBLE (sumDelta);
                //System.out.println();
                //***** konec vypisu 32
                //stanovení dílčích fitness funkcí FSE, 11. metoda
                //nulování pole
                for (int i=0;i<FSE.length;i++) {
                    for (int j=0;j<FSE[0].length;j++) {
                        FSE[i][j]=0;
                    }
                }

FSE=dilciFitnessFSE(velPop,pomocneL,vaha1,vaha2,sumDelta,PPZ1,SPPZ1,FSE,0);

FSE=dilciFitnessFSE(velPop,pomocneO,vaha1,vaha2,sumDelta,PPZ2,SPPZ2,FSE,1);

FSE=dilciFitnessFSE(velPop,pomocnePE,vaha1,vaha2,sumDelta,PPZ3,SPPZ3,FSE,2);

FSE=dilciFitnessFSE(velPop,pomocnePR,vaha1,vaha2,sumDelta,PPZ4,SPPZ4,FSE,3);
```

```
FSE=dilciFitnessFSE(velPop,pomocneS,vaha1,vaha2,sumDelta,PPZ5,SPPZ5,FSE,4);

FSE=dilciFitnessFSE(velPop,pomocneU,vaha1,vaha2,sumDelta,PPZ6,SPPZ6,FSE,5);
               //***** zacatek vypisu 33
               //System.out.println("** Vypis pole FSE **");
               //vypis2DpoleDOUBLE (FSE);
               //System.out.println();
               //***** konec vypisu 33
               //* stanovení FOV u pracovních skupin, 12. metoda *
               //určení středních hodnot známek osobních vlastností
               //nulovani pole
               for (int i=0;i<prumVlastnostL.length;i++) {
                   for (int j=0;j<prumVlastnostL[0].length;j++) {
                       prumVlastnostL[i][j]=0;
                   }
               }
               for (int i=0;i<prumVlastnostO.length;i++) {
                   for (int j=0;j<prumVlastnostO[0].length;j++) {
                       prumVlastnostO[i][j]=0;
                   }
               }
               for (int i=0;i<prumVlastnostPE.length;i++) {
                   for (int j=0;j<prumVlastnostPE[0].length;j++) {
                       prumVlastnostPE[i][j]=0;
                   }
               }
               for (int i=0;i<prumVlastnostPR.length;i++) {
                   for (int j=0;j<prumVlastnostPR[0].length;j++) {
                       prumVlastnostPR[i][j]=0;
                   }
               }
               for (int i=0;i<prumVlastnostS.length;i++) {
                   for (int j=0;j<prumVlastnostS[0].length;j++) {
                       prumVlastnostS[i][j]=0;
                   }
               }
               for (int i=0;i<prumVlastnostU.length;i++) {
                   for (int j=0;j<prumVlastnostU[0].length;j++) {
                       prumVlastnostU[i][j]=0;
                   }
               }
               prumVlastnostL=prumVlastnostFOV(velPop, POV, mutaceL,
prumVlastnostL,0);
               prumVlastnostO=prumVlastnostFOV(velPop, POV, mutaceO,
prumVlastnostO,1);
               prumVlastnostPE=prumVlastnostFOV(velPop, POV, mutacePE,
prumVlastnostPE,2);
               prumVlastnostPR=prumVlastnostFOV(velPop, POV, mutacePR,
prumVlastnostPR,3);
               prumVlastnostS=prumVlastnostFOV(velPop, POV, mutaceS,
prumVlastnostS,4);
               prumVlastnostU=prumVlastnostFOV(velPop, POV, mutaceU,
prumVlastnostU,5);
               //***** zacatek vypisu 34
               //System.out.println("** Vypis pole prumVlastnostL **");
               //vypis2DpoleDOUBLE (prumVlastnostL);
               //System.out.println();
               //System.out.println("** Vypis pole prumVlastnostO **");
               //vypis2DpoleDOUBLE (prumVlastnostO);
               //System.out.println();
               //System.out.println("** Vypis pole prumVlastnostPE **");
               //vypis2DpoleDOUBLE (prumVlastnostPE);
               //System.out.println();
               //System.out.println("** Vypis pole prumVlastnostPR **");
               //vypis2DpoleDOUBLE (prumVlastnostPR);
               //System.out.println();
               //System.out.println("** Vypis pole prumVlastnostS **");
```

```
                //vypis2DpoleDOUBLE (prumVlastnostS);
                //System.out.println();
                //System.out.println("** Vypis pole prumVlastnostU **");
                //vypis2DpoleDOUBLE (prumVlastnostU);
                //System.out.println();
                //***** konec vypisu 34
                //fitness FOV, 13. metoda
                //nulovani pole
                for (int i=0;i<FOV.length;i++) {
                    for (int j=0;j<FOV[0].length;j++) {
                        FOV[i][j]=0;
                    }
                }
                FOV=dilciFitnessFOV(velPop,POV,FOV,prumVlastnostL,pozadovaneL,0);
                FOV=dilciFitnessFOV(velPop,POV,FOV,prumVlastnostO,pozadovaneO,1);
                FOV=dilciFitnessFOV(velPop,POV,FOV,prumVlastnostPE,pozadovanePE,2);
                FOV=dilciFitnessFOV(velPop,POV,FOV,prumVlastnostPR,pozadovanePR,3);
                FOV=dilciFitnessFOV(velPop,POV,FOV,prumVlastnostS,pozadovaneS,4);
                FOV=dilciFitnessFOV(velPop,POV,FOV,prumVlastnostU,pozadovaneU,5);
                //***** zacatek vypisu 35
                //System.out.println("** Vypis pole FOV **");
                //vypis2DpoleDOUBLE (FOV);
                //System.out.println();
                //***** konec vypisu 35
                //dílčí fitness funkce FZZ, FSE a FOV roznásobené váhami
                vahyFZZ=roznasobeniVahamiFZZ (velPop,tym,vahyFZZ,vahy,FZZ);
                vahyFSE=roznasobeniVahamiFSE (velPop,tym,vahyFSE,vahy,FSE);
                vahyFOV=roznasobeniVahamiFOV (velPop,tym,vahyFOV,vahy,FOV);
                //***** zacatek vypisu 36
                //System.out.println("** Vypis pole vahyFZZ **");
                //vypis2DpoleDOUBLE (vahyFZZ);
                //System.out.println();
                //System.out.println("** Vypis pole vahyFSE **");
                //vypis2DpoleDOUBLE (vahyFSE);
                //System.out.println();
                //System.out.println("** Vypis pole vahyFOV **");
                //vypis2DpoleDOUBLE (vahyFOV);
                //System.out.println();
                //***** konec vypisu 36
                //součet dílčích funkcí fitness roznásobených váhami
                //nulovani pole
                for (int i=0;i<sumaDilciFitness.length;i++) {
                    for (int j=0;j<sumaDilciFitness[0].length;j++) {
                        sumaDilciFitness[i][j]=0.0;
                    }
                }
                sumaDilciFitness=sumaDilciFitness
(velPop,tym,sumaDilciFitness,vahyFZZ,vahyFSE,vahyFOV);
                //***** zacatek vypisu 37
                //System.out.println("** Vypis pole sumaDilciFitness **");
                //vypis2DpoleDOUBLE (sumaDilciFitness);
                //System.out.println();
                //***** konec vypisu 37


                //*** 10. STANOVENÍ FITNESS FUNKCE PRACOVNÍHO TÝMU ***
                FT=fitnessFunkceTymu (FT,velPop,sumaDilciFitness);
                //***** zacatek vypisu
                //System.out.println("** Vypis pole FT - fitness funkce tymu **");
                //vypis1DpoleDOUBLE(FT);
                //System.out.println();
                //***** konec vypisu


                //*** 11. VYHODNOCENÍ POPULACE ***
                //1. prům. fitness populace, 2. rozptyl populace, 3. kód nejlepší
populace
                //1. průměrná fitness populace
```

```
                //nulovani pole
                prumFitnessPopulace=0.0;
                prumFitnessPopulace=prumFitnessPopulace
(prumFitnessPopulace,velPop,FT);
                //***** zacatek vypisu 37b
                //System.out.println("Vypis prumFitnessPopulace");
                //System.out.println(prumFitnessPopulace);
                //***** konec vypisu 37b
                //2. rozptyl populace
                rozptyl=0.0;
                rozptyl=rozptylPopulace (rozptyl, velPop, prumFitnessPopulace,FT);
                //***** zacatek vypisu 37c
                //System.out.println("Vypis rozptyl");
                //System.out.println(rozptyl);
                //***** konec vypisu 37c
                //3. nejlepší fitness v populaci
                nejFT=nejFitnessPopulace (setrideneFT, FT, nejFT);
                //***** zacatek vypisu 37d
                //System.out.println("Vypis nejFT");
                //System.out.println(nejFT);
                //***** konec vypisu 37d
                //4. vyhledani nejlepsiho jedince dle nej FT a ulozeni jeho kodu do
pole nejJedinec[]
                //vytvoření 2D polí castNejJedince[][], zde bude uložen kód
nejlepšího jedince z populace
                //určení pořadí nejlepšího jedince
                castNejJedinceL=ulozeniNejJedince
(velPop,castNejJedinceL,mutaceL,poradiNejJedince,nejFT,FT);
                castNejJedinceO=ulozeniNejJedince
(velPop,castNejJedinceO,mutaceO,poradiNejJedince,nejFT,FT);
                castNejJedincePE=ulozeniNejJedince
(velPop,castNejJedincePE,mutacePE,poradiNejJedince,nejFT,FT);
                castNejJedincePR=ulozeniNejJedince
(velPop,castNejJedincePR,mutacePR,poradiNejJedince,nejFT,FT);
                castNejJedinceS=ulozeniNejJedince
(velPop,castNejJedinceS,mutaceS,poradiNejJedince,nejFT,FT);
                castNejJedinceU=ulozeniNejJedince
(velPop,castNejJedinceU,mutaceU,poradiNejJedince,nejFT,FT);
                //uložení kódů nejlepších jedinců
                for (int i=0;i<tym[0];i++) {
                for (int j=0;j<logistika[0].length;j++) {
                    nejJedinciL[i][j][pc+1]=castNejJedinceL[i][j];
                    }
                }
                for (int i=0;i<tym[1];i++) {
                for (int j=0;j<obchod[0].length;j++) {
                    nejJedinciO[i][j][pc+1]=castNejJedinceO[i][j];
                    }
                }
                for (int i=0;i<tym[2];i++) {
                for (int j=0;j<personalistika[0].length;j++) {
                    nejJedinciPE[i][j][pc+1]=castNejJedincePE[i][j];
                    }
                }
                for (int i=0;i<tym[3];i++) {
                for (int j=0;j<projekce[0].length;j++) {
                    nejJedinciPR[i][j][pc+1]=castNejJedincePR[i][j];
                    }
                }
                for (int i=0;i<tym[4];i++) {
                for (int j=0;j<stavbyvedouci[0].length;j++) {
                    nejJedinciS[i][j][pc+1]=castNejJedinceS[i][j];
                    }
                }
                for (int i=0;i<tym[5];i++) {
                for (int j=0;j<ucetni[0].length;j++) {
                    nejJedinciU[i][j][pc+1]=castNejJedinceU[i][j];
                    }
```

```
                }


                //*** 12. SOUHRNÉ VYHODNOCENÍ POPULACE ***
                //uložení průměrné fitness, rozptylu a nej fitness u populace do 2D
pole

souhrneVysledky=souhrneVyhodnoceni(souhrneVysledky,pocetIteraci,prumFitnessPopulace,
rozptyl,nejFT,pc);
                }
                //****** KONEC CYKLU *****


                //výpis souhrných výsledků na obrazovku
                //System.out.println("\n" + "Výpis souhrných výsledků - 1.
sl.=prumFitness, 2. sl.=rozptyl, 3. sl.=nejFitness" +"\n");
                //for (int i=0;i<pocetIteraci+1;i++) {
                    //for (int j=0;j<3;j++) {
                        //System.out.print(souhrneVysledky[i][j] + "\t");
                    //}
                    //System.out.println();
                //}

                //*** ukladani souhrnych vysledku ***
                for (int i=0;i<povy;i++) {
                for (int j=0;j<pocetIteraci+1;j++) {
                    for (int k=0;k<3;k++) {
                        souhrneVysledkyGlobal[j][k][i]=souhrneVysledky[j][k];
                    }
                }
                }


                //*** 13. ULOŽENÍ VYPOČÍTANÝCH DAT A JEDINCŮ DO SOUBORU ***
                File fwJm = new File("C:\\CESTA\\statistikaOptiPracTym.txt");
                FileWriter fw = new FileWriter(fwJm);
                String separator = System.getProperty("line.separator");
                for (int i=0;i<souhrneVysledky.length;i++) {
                    for (int j=0;j<souhrneVysledky[0].length;j++) {
                        fw.write(String.valueOf(souhrneVysledky[i][j]));
                        fw.write("\t");
                    }
                    fw.write(separator);
                }
                if (povyp==povy-1) {
                fw.close();
                }
                //*** NOVÁ STATISTIKA - průměrná fitness ***
                for (int i=0;i<souhrneVysledky.length;i++) {
                        vi.write(String.valueOf(souhrneVysledkyGlobal[i][0][povy-
1]));
                        vi.write("\t");
                }
                vi.write("\t");
                vi.write(separator);
                if (povyp==povy-1) {
                vi.close();
                }

                //*** NOVÁ STATISTIKA - rozptyl ***
                for (int i=0;i<souhrneVysledky.length;i++) {
                        vii.write(String.valueOf(souhrneVysledky[i][1]));
                        vii.write("\t");
                }
                vii.write("\t");
                vii.write(separator);
                if (povyp==povy-1) {
                vii.close();
```

```
}

//*** NOVÁ STATISTIKA - nejFitness ***
for (int i=0;i<souhrneVysledky.length;i++) {
        viii.write(String.valueOf(souhrneVysledky[i][2]));
        viii.write("\t");
}
viii.write("\t");
viii.write(separator);
if (povyp==povy-1) {
viii.close();
}

//* ulozeni nejlepsich logistiku *
File nejL = new File("C:\\CESTA\\nejLogistika.txt");
FileWriter l = new FileWriter(nejL);
for (int i=0;i<pocetIteraci+1;i++) {
    for (int k=0;k<tym[0];k++) {
        for (int j=0;j<logistika[0].length;j++) {
            l.write(String.valueOf(nejJedinciL[k][j][i]));
            l.write("\t");
        }
        l.write(separator);
    }
    l.write(separator);
}
if (povyp==povy-1) {
l.close();
}
//* ulozeni nejlepsich obchodniku *
File nejO = new File("C:\\CESTA\\nejObchod.txt");
FileWriter o = new FileWriter(nejO);
for (int i=0;i<pocetIteraci+1;i++) {
    for (int k=0;k<tym[1];k++) {
        for (int j=0;j<obchod[0].length;j++) {
            o.write(String.valueOf(nejJedinciO[k][j][i]));
            o.write("\t");
        }
        o.write(separator);
    }
    o.write(separator);
}
if (povyp==povy-1) {
o.close();
}
//* ulozeni nejlepsich personalistu *
File nejPE = new File("C:\\CESTA\\nejPErsonalistika.txt");
FileWriter pe = new FileWriter(nejPE);
for (int i=0;i<pocetIteraci+1;i++) {
    for (int k=0;k<tym[2];k++) {
        for (int j=0;j<personalistika[0].length;j++) {
            pe.write(String.valueOf(nejJedinciPE[k][j][i]));
            pe.write("\t");
        }
        pe.write(separator);
    }
    pe.write(separator);
}
if (povyp==povy-1) {
pe.close();
}
//* ulozeni nejlepsich projektantu *
File nejPR = new File("C:\\CESTA\\nejPRojekce.txt");
FileWriter pr = new FileWriter(nejPR);
for (int i=0;i<pocetIteraci+1;i++) {
    for (int k=0;k<tym[3];k++) {
        for (int j=0;j<projekce[0].length;j++) {
            pr.write(String.valueOf(nejJedinciPR[k][j][i]));
```

```
                        pr.write("\t");
                    }
                    pr.write(separator);
                }
                pr.write(separator);
            }
            if (povyp==povy-1) {
            pr.close();
            }
            //* ulozeni nejlepsich stavbyvedoucich *
            File nejS = new File("C:\\CESTA\\nejStavbyvedouci.txt");
            FileWriter s = new FileWriter(nejS);
            for (int i=0;i<pocetIteraci+1;i++) {
                for (int k=0;k<tym[4];k++) {
                    for (int j=0;j<stavbyvedouci[0].length;j++) {
                        s.write(String.valueOf(nejJedinciS[k][j][i]));
                        s.write("\t");
                    }
                    s.write(separator);
                }
                s.write(separator);
            }
            if (povyp==povy-1) {
            s.close();
            }
            //* ulozeni nejlepsich ucetnich *
            File nejU = new File("C:\\CESTA\\nejUcetni.txt");
            FileWriter u = new FileWriter(nejU);
            for (int i=0;i<pocetIteraci+1;i++) {
                for (int k=0;k<tym[5];k++) {
                    for (int j=0;j<ucetni[0].length;j++) {
                        u.write(String.valueOf(nejJedinciU[k][j][i]));
                        u.write("\t");
                    }
                    u.write(separator);
                }
                u.write(separator);
            }
            if (povyp==povy-1) {
            u.close();
            }

        }
        } //ukonceni vnejsiho cyklu



        //*************************************
        //*** METODY OPTIMALIZAČNÍHO PROGRAMU ***
        //*************************************
            //** 1. METODA **
            //naplní jedničkami 1D pole vstupující do metody
            public static int[] plneniJednickami(int[] prvkyPole) {
                Arrays.fill(prvkyPole,1);
            return prvkyPole;
            }
            //** 2. METODA **
            //sečte známky určité znalosti, zkušenosti nebo osobní vlastnosti
            public static int[][]souctyZnamekSkupina(int
SZZZ[][],int[][][]prvkyPole,int velPop) {
                //dulezete - vynulovat predem pole
                for (int h=0;h<SZZZ[0].length;h++) {
                    for (int g=0;g<SZZZ.length;g++) {
                        SZZZ[g][h]=0;
                    }
                }
                for (int k=0; k<velPop;k++) {
                    for (int j=0;j<prvkyPole[0].length;j++) {
```

```
                    for (int i=0;i<prvkyPole.length;i++) {
                        SZZZ[k][j]+=prvkyPole[i][j][k];
                    }
                }
            }
            return SZZZ;
        }
        //** 3. METODA **
        //plnění 3D polí inicializaceL-U známkami náhodně zvolených zaměstnanců
        public static int[][][]plneniPoleInicializace(int [][][]inicializace,
int[][]zamestnanci,int konstanta, int velPop) {
            int pom[]=new int[zamestnanci.length];
            Random rand = new Random();
            for (int l=0;l<velPop;l++) {
                pom=plneniJednickami(pom); //volání 1. metody
                for (int j=0;j<konstanta;) {
                     int nah=rand.nextInt(pom.length);
                     if (pom[nah]==1) {
                        pom[nah]=0;
                        for (int k=0;k<zamestnanci[0].length;k++) {
                            inicializace[j][k][l]=zamestnanci[nah][k];
                        }
                        j++;
                    }
                }
            }
            return inicializace;
        }
        //** 4. METODA **
        //určení dílčích fitness funkcí FZZ
        public static double[][]urceniFunkceFZZ(double FZZ[][],int
[][]SZZZ,int[][][] prvkyPole,int[][]zamestnanci, int vlastnosti, int[]pozadavek,int
velPop,int skupina) {
            for (int l=0;l<velPop;l++) {
                for (int j=0;j<zamestnanci[0].length-vlastnosti;j++) {
                FZZ[l][skupina]+=Math.abs(prvkyPole.length*pozadavek[j]-
SZZZ[l][j]);
            }
            }
            return FZZ;
        }
        //** 5. METODA **
        //naplnění 3D pomocných polí podle klíče z 3D polí inicializace
        public static int[][][]plneniPomocPoli(int velPop, int
inicializace[][][],int POV, int pomocne[][][], int pozadovane[]) {
            for (int l=0;l<velPop;l++) {
                for (int j=0; j<inicializace.length;j++) {
                    for (int k=0;k<inicializace[0].length-POV;k++) {
                        if (inicializace[j][k][l]==pozadovane[k]) {
                        pomocne[j][k][l]=1;
                        }
                        if (inicializace[j][k][l]<pozadovane[k]) {
                        pomocne[j][k][l]=1;
                        }
                        if (inicializace[j][k][l]>pozadovane[k]) {
                        pomocne[j][k][l]=0;
                    }
                    }
                }
            }
            return pomocne;
        }
        //** 6. METODA **
        //přiřazení hodnot do polí PPZ1-6
        public static double[]prirazeniHodnot(int inicializace[][][],int velPop,
int POV, double PPZ[], double procenta[]) {
            for (int k=0;k<inicializace[0].length-POV;k++) {
            PPZ[k]=procenta[k]*inicializace.length;
```

```
            }
                return PPZ;
            }
            //** 7. METODA **
            //stanovení skutečného počtu požadovaných známek
            public static int[][]skutPocetZnamek(int velPop,int
inicializace[][][],int POV,int pomocne[][][],int SPPZ[][]) {
                //vynulovani pole
                for (int a=0;a<SPPZ[0].length;a++) {
                    for (int e=0;e<SPPZ.length;e++) {
                        SPPZ[e][a]=0;
                    }
                }
                for (int l=0;l<velPop;l++) {
                    for (int k=0;k<inicializace[0].length-POV;k++) {
                        int m=0;
                        for (int j=0;j<inicializace.length;j++) {
                            if (pomocne[j][k][l]!=0) {
                                m++;
                                SPPZ[l][k]=m;
                            }
                        }
                    }
                }
                return SPPZ;
            }
            //** 8. METODA **
            //stanovení součtu prvků náhradních polí v řadách
            public static double[][]souctyPrvkuNahPole(int velPop,int
pomocne[][][],double sumRada[][]) {
                //vynulovani pole
                for (int o=0;o<sumRada[0].length;o++) {
                    for (int p=0;p<sumRada.length;p++) {
                        sumRada[p][o]=0;
                    }
                }
                for (int l=0;l<velPop;l++) {
                    for (int j=0;j<pomocne.length;j++) {
                        for (int k=0;k<pomocne[0].length;k++) {
                            sumRada[j][l]+=pomocne[j][k][l];
                        }
                    }
                }
                return sumRada;
            }
            //** 9. METODA **
            //stanovení průměrných hodnot součtů u jednotlivých skupin
            public static double[][] prumHodnotaSoucet(double sumRada[][],int
velPop,int pomocne[][][],double stredHodn[][],double  pocetZam,int cislo) {
                for (int l=0;l<velPop;l++) {
                    for (int j=0;j<pomocne.length;j++) {
                        stredHodn[cislo][l]+=(1.0/pocetZam)*sumRada[j][l];
                    }
                }
                return stredHodn;
            }
            //** 10. METODA **
            //součty odchylek od středních hodnot
            public static double[][] souctyOdchylekOdStredHodnot(int velPop, int
pomocne[][][],double sumRada[][],double stredHodn[][],double sumDelta[][],int cislo)
{
                for (int l=0;l<velPop;l++) {
                    for (int j=0;j<pomocne.length;j++) {
                        sumDelta[l][cislo]+=Math.abs(sumRada[j][l]-
stredHodn[cislo][l]);
                    }
                }
                return sumDelta;
```

```
        }
        //** 11. METODA **
        //stanovení dílčích fitness funkcí FSE
        public static double[][] dilciFitnessFSE(int velPop,int
pomocne[][][],double vaha1,double vaha2,double sumDelta[][],double PPZ[],int
SPPZ[][],double FSE[][],int cislo) {
            for (int l=0; l<velPop;l++) {
                    for (int k=0;k<pomocne[0].length;k++) {
                        FSE[l][cislo]+=vaha1*(Math.abs(PPZ[k]-
SPPZ[l][k]))+vaha2*sumDelta[l][cislo];
                    }
            }
            return FSE;
        }
        //** 12. METODA **
        //*** stanovení FOV u pracovních skupin ***
        public static double[][] prumVlastnostFOV(int velPop, int POV[], int
inicializace[][][], double prumVlastnost[][],int cislo) {
            for (int l=0;l<velPop;l++) {
                for (int k=0;k<POV[cislo];k++)  {
                    for (int j=0;j<inicializace.length;j++) {

prumVlastnost[l][k]+=(1.0/inicializace.length)*inicializace[j][inicializace[0].lengt
h-POV[cislo]+k][l];
                    }
                }
            }
            return prumVlastnost;
        }
        //** 13. METODA **
        //fitness FOV
        public static double[][] dilciFitnessFOV(int velPop,int POV[],double
FOV[][],double prumVlastnost[][],int pozadovane[],int cislo) {
            for (int l=0;l<velPop;l++) {
                for (int k=0;k<POV[cislo];k++) {
                    FOV[l][cislo]+=Math.abs(prumVlastnost[l][k]-
pozadovane[pozadovane.length-POV[cislo]+k]);
                }
            }
            return FOV;
        }
        //** 14. METODA **
        //tato metoda slouží k provedeni selekce populace
        public static int[][][] selektovaniPopulace(int velPop,int
zamestnanci[][],int tym,int selektovanaPopulace[][][],int inicializace[][][],int
postupujici[],int mutovanaPopulace[][][],int c) {
            for (int i=0;i<velPop;i++) {
                for (int j=0;j<zamestnanci[0].length;j++) {
                    for (int k=0;k<tym;k++) {
                        if (c==0) {

selektovanaPopulace[k][j][i]=inicializace[k][j][postupujici[i]];
                        }
                        else {

selektovanaPopulace[k][j][i]=mutovanaPopulace[k][j][postupujici[i]];
                        }
                    }
                }
            }
            return selektovanaPopulace;
        }
        //** 15. METODA **
        //tato metoda slouzi k provedeni mutace populace
        public static int[][][] mutovaniPopulace(int velPop, int [][]zamestnanci,
int [][][]selektovanaPopulace,int Pm,int tymKonst) {
            //System.out.println("vstup do mutace");
            Random rand = new Random();
```

```
                int mutace[][][]=new int[tymKonst][zamestnanci[0].length][velPop];
                //kopirovani selektovane populace do poli mutace
                for (int i=0;i<velPop;i++) {
                    for (int j=0;j<tymKonst;j++) {
                        for (int k=0;k<zamestnanci[0].length;k++) {
                            mutace[j][k][i]=selektovanaPopulace[j][k][i];
                        }
                    }
                }
                int vylosovani[]=new int[velPop];
                //plneni pole vylosovani nulami
                for (int j=0;j<velPop;j++) {
                    vylosovani[j]=0;
                }
                //zamestnanci skupiny, kteri budou predmetem zmen dle zvolene
pravdepodobnosti
                for (int j=0;j<velPop;j++) {
                int nah=rand.nextInt(Pm);
                    if (nah==1) {
                        vylosovani[j]=1;
                    }
                }
                //vlastni mutace (mutovany jedinec = znovu inicializovany jedinec)
                int pom[]=new int[zamestnanci.length];
                for (int l=0;l<velPop;l++) {
                    //podminka, aby nemutoval vse, ale jen vylosovaneho jedince
                    if (vylosovani[l]==1) {
                    pom=plneniJednickami(pom); //volani 1. metody
                    for (int j=0;j<mutace.length;) {
                         int nah=rand.nextInt(pom.length);
                         if (pom[nah]==1) {
                            pom[nah]=0;
                            for (int k=0;k<zamestnanci[0].length;k++) {
                                mutace[j][k][l]=zamestnanci[nah][k];
                            }
                            j++;
                        }
                    }
                } //podm
                }
                return mutace;
            }
            //** 16. METODA **
            //tato metoda slouží ke stanovení průměrné fitness populace
            //je použita před vstupem do cyklu a na konci cyklu po provedení mutace
            public static double prumFitnessPopulace (double prumFitnessPopulace, int
velPop, double FT[]) {
                for (int i=0;i<velPop;i++) {
                    prumFitnessPopulace+=(FT[i])/velPop;
                }
            return prumFitnessPopulace;
            }
            //** 17. METODA **
            //tato metoda slouží ke stanovení rozptylu populace
            //je použita před vstupem do cyklu a na konci cyklu po provedení mutace
            public static double rozptylPopulace (double rozptyl, int velPop, double
prumFitnessPopulace, double FT[]) {
                for (int i=0;i<velPop;i++) {
                    rozptyl+=(1.0/velPop)*(Math.pow((FT[i]-prumFitnessPopulace),2));
                }
                return rozptyl;
            }
            //** 18. METODA **
            //tato metoda slouží ke stanovení nejlepší fitness v populaci
            //je použita před vstupem do cyklu a na konci cyklu po provedení mutace
            public static double nejFitnessPopulace (double []setrideneFT, double
[]FT, double nejFT) {
```

```
            setrideneFT = (double[]) FT.clone();  //převzetí prvků z pole FT
(nutná konverze, aby nevrátil typ object)
            Arrays.sort(setrideneFT);
            nejFT = setrideneFT[0];  //minimum FT je nejlepsi
            return nejFT;
        }
        //** 19. METODA **
        //tato metoda slouží k uložení kódu nejlepšího jedince
        //je použita před vstupem do cyklu a na konci cyklu po provedení mutace
        public static int[][] ulozeniNejJedince (int velPop,int
[][]castNejJedince,int inicializace[][][],int poradiNejJedince,double nejFT,double
FT[]) {
            //určení pořadí nejlepšího jedince
        for (int i=0;i<velPop;i++) {
            if (nejFT==FT[i]) {
                poradiNejJedince=i;
            }
        }
        //načtení nejlepšího jedince do 2D polí castNejJedince[][]
        for (int i=0;i<inicializace.length;i++) {
            for (int j=0;j<inicializace[0].length;j++) {
                castNejJedince[i][j]=inicializace[i][j][poradiNejJedince];
            }
        }
        return castNejJedince;
        }
        //** 20. METODA **
        //tato metoda provede souhrne ulozeni vypocitanych vysledku
        public static double[][] souhrneVyhodnoceni(double[][]
souhrneVysledky,int pocetIteraci,double prumFitnessPopulace,double rozptyl,double
nejFT,int c) {
                souhrneVysledky[c+1][0]=prumFitnessPopulace;
                souhrneVysledky[c+1][1]=rozptyl;
                souhrneVysledky[c+1][2]=nejFT;
            return souhrneVysledky;
        }
        //** 21. METODA **
        //vypis 1D pole typu INT na obrazovku
        static void vypis1DpoleINT (int []pole) {
            for (int i=0;i<pole.length;i++) {
                System.out.print(pole[i]+ "\t");
            }
        }
        //** 22. METODA **
        //vypis 2D pole typu INT na obrazovku
        static void vypis2DpoleINT (int [][]pole) {
            for (int i=0;i<pole.length;i++) {
                for (int j=0;j<pole[0].length;j++) {
                    System.out.print(pole[i][j] +"\t");
                }
                System.out.println("\n");
            }
        }
        //** 23. METODA **
        //vypis 3D pole typu INT na obrazovku
        static void vypis3DpoleINT (int [][][]pole) {
            for (int k=0;k<pole[0][0].length;k++) {
                for (int i=0;i<pole.length;i++) {
                    for (int j=0;j<pole[0].length;j++) {
                        System.out.print(pole[i][j][k] +"\t");
                    }
                    System.out.println();
                }
                System.out.println();
            }
        }
        //** 24. METODA **
        //vypis 1D pole typu DOUBLE na obrazovku
```

```
            static void vypis1DpoleDOUBLE (double []pole) {
                for (int i=0;i<pole.length;i++) {
                    System.out.print(pole[i]+ "\t");
                }
            }
            //** 25. METODA **
            //vypis 2D pole typu DOUBLE na obrazovku
            static void vypis2DpoleDOUBLE (double [][]pole) {
                for (int i=0;i<pole.length;i++) {
                    for (int j=0;j<pole[0].length;j++) {
                        System.out.print(pole[i][j] +"\t");
                    }
                    System.out.println("\n");
                }
            }
            //** 26. METODA **
            //vypis 3D pole typu DOUBLE na obrazovku
            static void vypis3DpoleDOUBLE (double [][][]pole) {
                for (int k=0;k<pole[0][0].length;k++) {
                    for (int i=0;i<pole.length;i++) {
                        for (int j=0;j<pole[0].length;j++) {
                            System.out.print(pole[i][j][k] +"\t");
                        }
                        System.out.println();
                    }
                    System.out.println();
                }
            }
            //** 27. METODA **
            //dílčí fitness funkce FZZ roznásobené váhami
            public static double[][] roznasobeniVahamiFZZ (int velPop,int
tym[],double vahyFZZ[][],double vahy[][],double FZZ[][]) {
                for (int l=0;l<velPop;l++) {
                    for (int i=0;i<tym.length;i++) {
                        vahyFZZ[l][i]=vahy[i][0]*FZZ[l][i];
                    }
                }
                return vahyFZZ;
            }
            //** 28. METODA **
            //dílčí fitness funkce FSE roznásobené váhami
            public static double[][] roznasobeniVahamiFSE (int velPop,int
tym[],double vahyFSE[][],double vahy[][],double FSE[][]) {
                for (int l=0;l<velPop;l++) {
                    for (int i=0;i<tym.length;i++) {
                        vahyFSE[l][i]=vahy[i][1]*FSE[l][i];
                    }
                }
                return vahyFSE;
            }
            //** 29. METODA **
            //dílčí fitness funkce FOV roznásobené váhami
            public static double[][] roznasobeniVahamiFOV (int velPop,int
tym[],double vahyFOV[][],double vahy[][],double FOV[][]) {
                for (int l=0;l<velPop;l++) {
                    for (int i=0;i<tym.length;i++) {
                        vahyFOV[l][i]=vahy[i][2]*FOV[l][i];
                    }
                }
                return vahyFOV;
            }
            //** 30. METODA **
            //součet dílčích funkcí fitness roznásobených váhami
            public static double[][] sumaDilciFitness (int velPop,int tym[],double
sumaDilciFitness[][],double vahyFZZ[][],double vahyFSE[][],double vahyFOV[][]) {
                for (int l=0;l<velPop;l++) {
                        for (int k=0;k<tym.length;k++) {
                            sumaDilciFitness[l][0]+=vahyFZZ[l][k];
```

```
                        sumaDilciFitness[l][1]+=vahyFSE[l][k];
                        sumaDilciFitness[l][2]+=vahyFOV[l][k];
                    }
                }
                return sumaDilciFitness;
            }
            //** 31. METODA **
            //* stanoveni fitness funkce pracovniho tymu *
            public static double[] fitnessFunkceTymu (double FT[],int velPop,double
sumaDilciFitness[][]) {
                for (int l=0;l<velPop;l++) {

FT[l]=Math.sqrt(Math.pow(sumaDilciFitness[l][0],2.0)+Math.pow(sumaDilciFitness[l][1]
,2.0)+Math.pow(sumaDilciFitness[l][2],2.0));
                }
                return FT;
            }
    }
```