UNIVERZITA PARDUBICE
FAKULTA EKONOMICKO - SPRÁVNÍ
ÚSTAV SYSTÉMOVÉHO INŽENÝRSTVÍ A INFORMATIKY

GENETICKÉ ALGORITMY VE ZVOLENÉM SOFTWAROVÉM PROSTŘEDÍ

BAKALÁŘSKÁ PRÁCE

AUTOR PRÁCE: Michala Maděrová

VEDOUCÍ PRÁCE: Ing. Jan Panuš

2007

UNIVERSITY OF PARDUBICE
FACULTY OF ECONOMICS AND ADMINISTRATION
DEPARTMENT OF SYSTEM ENGINEERING AND INFORMATICS

GENETIC ALGORITHM IN THE CHOSEN SOFTWARE BACKGROUND

BACHELOR WORK

AUTHOR: Michala Maděrová
SUPERVISOR: Ing. Jan Panuš

2007

UNIVERZITA PARDUBICE

Fakulta Ekonomicko-správní
Ústav Systémového inženýrství a informatiky
Akademický rok: 2006/2007

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Pro:	**Michala Maděrová**
Studijní program:	**Systémové inženýrství a informatika**
Studijní obor:	**Informatika ve veřejné správě**
Název tématu:	**Genetické algoritmy ve zvoleném softwarovém prostředí**

Zásady pro zpracování:

- Úvod do genetických algoritmů
- Využití genetických algoritmů
- Samotná aplikace využívající genetických algoritmů - v konkrétním prostředí

Seznam odborné literatury:

- KVASNIČKA, V., POSPÍCHAL, J., TIŇO, P. *Evolučné algoritmy*. STU Bratislava 2000. 80-227-1377-5.
- MAŘÍK, V., ŠTĚPÁNKOVÁ, O., LAŽANSKÝ, J. *Umělá inteligence (3)*. Academia, Praha, 2001. 80-200-0472-6.
- OLEJ, V. *Modelovanie ekonomických procesov na báze výpočtovej inteligencie*, Miloš Vognar-M&V, Hradec Králové, 2003. 80-90329-9-1.

Rozsah:	**40-50 stran**
Vedoucí práce:	**Ing. Jan Panuš**
Vedoucí ústavu:	**doc. Ing. Pavel Petr, Ph.D.**
Datum zadání práce:	**1. 10. 2005**
Termín odevzdání práce:	**19. 2. 2007**

Prohlašuji:

Tuto práci jsem vypracovala samostatně. Veškeré literární prameny a informace, které jsem v práci využila, jsou uvedeny v seznamu použité literatury.

Byla jsem seznámena s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně Univerzity Pardubice.

V Pardubicích dne 19.2.2007

Michala Maděrová

Shrnutí

Tato bakalářská práce se zabývá řešením běžných optimalizačních úloh s využitím techniky genetických algoritmů. Tyto algoritmy představují heuristické optimalizační techniky založené na principech přirozené selekce a přírodní genetiky. Abstrahováním těchto přírodních principů vznikl počítačový algoritmus schopný řešit složité optimalizační problémy a na základě svých operátorů poměrně účinně prohledávat prostor všech možných řešení.

Genetický algoritmus tak může představovat cestu pro řešení rozličných matematických i technických úloh. V této práci je ukázán možný přístup k řešení optimalizačních problémů využitím programovacího prostředí MATLAB a jeho toolboxu. Dále je popsána problematika hledání (plánování) nejkratší cesty mezi více místy známá jako Problém obchodního cestujícího.

OBSAH

Úvod

Tato bakalářská práce se zabývá řešením běžných optimalizačních úloh s využitím techniky genetických algoritmů (GA). Tyto algoritmy představují heuristické optimalizační techniky založené na principech přirozené selekce a přírodní genetiky. Abstrahováním těchto přírodních principů vznikl počítačový algoritmus schopný řešit složité optimalizační problémy a na základě svých operátorů poměrně účinně prohledávat prostor všech možných řešení. [16]

Genetický algoritmus tak může představovat cestu pro řešení rozličných matematických i technických úloh. V této práci je ukázán možný přístup k řešení optimalizačních problémů využitím programovacího prostředí MATLAB a jeho toolboxu. Dále je popsána problematika hledání nejkratší cesty mezi více místy známá jako Problém obchodního cestujícího.

Cílem této práce je studium genetických algoritmů obecně, představit jejich využití v podnikové praxi, lékařském a technologickém výzkumu, a samotný návrh genetických algoritmů implementovaný do prostředí MATLABU.

První kapitola se zabývá seznámením s genetickými algoritmy. Popisuje, jak byla biologická Darwinova teorie o vývoji druhů aplikována do informatiky, jakým způsobem ji rozšířila o vývoj „Evolučních výpočetních technik“, jejíž jednou z odnoží jsou genetické algoritmy. Dále se věnuje vývoji této teorie v čase a uvádí příklady typických problémů, které se genetickými algoritmy řeší. Jejich cílem je najít ze souboru možných řešení úlohy řešení dostatečně dobré. Jedná se tedy o optimalizační úlohu. Genetické algoritmy přitom praktikují podobné postupy jako živé organismy, a to především přírodní výběr - lepší řešení přežívají spíše než ta horší.[1]

Kapitola dále definuje základní pojmy: všechna možná řešení nazveme stavovým prostorem, každé jedno řešení jedincem, kvalitativní funkce (fitness) určuje, jak je řešení „dobré“. Soubor jedinců „žijících“ ve stejné době, zpravidla v paměti počítače, nazýváme generací. Selekcí nazýváme výběr skupiny jedinců z generace (dle jistých pravidel), křížením několika (typicky dvou) jedinců z generace vzniká nový jedinec mající některé vlastnosti svých „předků“, mutace je samovolná změna jedince v generaci. Ze současné generace jsou pak jedinci vybraní pro smrt odstraněni a nahrazeni novými jedinci, kteří vzniknou okopírováním kvalitních jedinců nebo křížením. Tak vzniká nová generace, která by při vhodném nastavení všech faktorů měla být kvalitnější než předchozí.[22]

Druhá kapitola nastiňuje, jak lze genetické algoritmy využít v řízení výroby, řešení problémů v podnikové praxi, zemědělství, průmyslu, vědeckém a klinickém výzkumu a dalších oborech. Praktické využití se tedy zaměřuje především na úlohy, kde klasické metody zcela určitě selhávají. Jedná se např. o řízení výrobního procesu. Genetický algoritmus dokáže optimálně navrhnout výrobní linku a zahrnout přitom do řešení spoustu okolních parametrů, jakými jsou např. možnost provedení dané operace na více strojích, libovolné následnosti v technologickém postupu, výpadek některého z pracovišť nebo jednotlivé operace, zahrnutí časových prodlev a nákladů spojených s přechodem od jednoho pracoviště k druhému, definice maximální délky fronty každého pracoviště atd. Podobnou úlohou je sestavení a optimalizace logistických řetězců, opět se zahrnutím mnoha doprovodných parametrů, nebo optimalizace obslužných systémů spočívající v nalezení optimálního počtu jednotlivých kanálů systému, nebo vytvoření různých variant reagujících na různou úroveň přísunu požadavků, které s sebou přináší moderní doba.[5]

Třetí kapitola implementuje algoritmy do praktických úloh prostřednictvím Matlabu. Matlab je integrované prostředí pro vědeckotechnické výpočty, modelování, návrhy algoritmů, simulace, analýzu a prezentaci dat, měření a zpracování signálů, návrhy řídicích a komunikačních systémů.[21] Současně obsahuje Genetic Algorithm Toolbox, který rozšiřuje optimalizační možnosti Matlabu o nástroje pracujícími s genetickými algoritmy. Je využitelný zejména v úlohách, které jsou obtížně řešitelné tradičními optimalizačními metodami, včetně úloh, které nejsou dobře definovatelné, nebo je lze obtížně modelovat matematicky. Kapitola dále ukazuje, jakým způsobem se v Matlabu pracuje s genetickými algoritmy prostřednictvím příkazového řádku, toolboxu a závěrem je návrh řešení konkrétního optimalizačního algoritmu – Problému obchodního cestujícího.

1 Princip fungování genetického algoritmu

Biologie versus informatika

V dějinách lidstva většinou matematika a informatika obohacují ostatní obory svými výsledky, jejíž aplikací na obor vzniká nová hodnota. Málokdy se stává, že obory obohacují matematiku a informatiku. To se právě stalo při genetických algoritmech. Jedna z biologických teorií byla aplikována na informatiku, a tím jsme získali možnost nacházet řešení problémů, kde náš matematický aparát selhával. Touto teorií je Darwinova teorie o vývoji druhů, jež rozšířila informatiku o vývoj „Evolučních výpočetních technik“, jejíž odnoží jsou „Genetické algoritmy“.

Její začátky můžeme položit do 60. let minulého století a jsou spojeny s řešením optimalizačních problémů. Jedním ze zakladatelů této disciplíny je John Holland, který popsal základní vlastnosti těchto algoritmů, a jeho práce je již dnes brána za „klasiku“. Od té doby prošly GA vývojem s množstvím reálných aplikací a tím potvrzením, že se nejedná jen o hříčku, ale o techniku, kterou můžeme získat inteligentní výsledky.[1]

Matrice života v počítačích

Genetické algoritmy mají velice mnoho společného s vývojem života. Oblast genetických algoritmů (GA) je zařazována do evolučních výpočetních technik, kde tvoří také velkou část neuronové sítě. Všechny tyto techniky a jim podobné patří do oboru umělé inteligence.

Jak již název napovídá, genetické algoritmy jsou inspirovány myšlenkami převzatými z biologie a souvisejících věd. Genetický algoritmus může být použit k řešení problému tak, že použije vyjádření problému ve formě chromozomu a snaží se "vypěstovat" optimální řešení za pomoci vyvíjející se populace. Existují problémy, které se dají řešit pomocí genetických algoritmů velmi dobře. Stejně tak je ovšem plno jiných problémů, které je vhodnější řešit jinými způsoby. Genetický algoritmus může však dát řešení i v případech, kde ostatní metody selhaly. Asi nejčastější oblastí aplikací GA je optimalizace a strojové učení. GA byly použity pro řešení problémů z technických oborů, chemie, ekonomie a dalších. Je namístě říci, že GA nejsou zamýšleny jako pomůcka pro simulaci přírody, přestože byly použity pro modelování přírodních pochodů (např. pro studie otázek z genetiky nebo pro modelování ekosystému).[2]

Proč se zabývat GA?

Co tedy genetický algoritmus je? Problém můžeme specifikovat takto: vyber z nějakých přípustných řešení to nejlepší. Takto to zní velmi jednoduše, ovšem pokud si uvědomíme, že u praktických problémů může být počet přípustných řešení velmi rozsáhlý, stává se situace složitější. Tento počet bývá tak velký, že ani při použití toho nejrychlejšího počítače nestihneme všechna přípustná řešení vygenerovat během našeho života, natož z nich vybrat to nejlepší.[2]

Správné řešení této problematiky dále neumíme vypočítat z důvodů:

a) neznáme správné vztahy (např. vzorce)
b) známe postup řešení, ale nejsme jej schopni realizovat.

Např. si vezmeme případ b). Budeme hledat optimální variantu posloupnosti objektů, která bude řešením určitého problému. Postup je znám. Provedeme všechny konfigurace uspořádání a vybereme optimální. Tento postup má však jeden háček a to, že počet uspořádání je faktoriál počtu objektů.

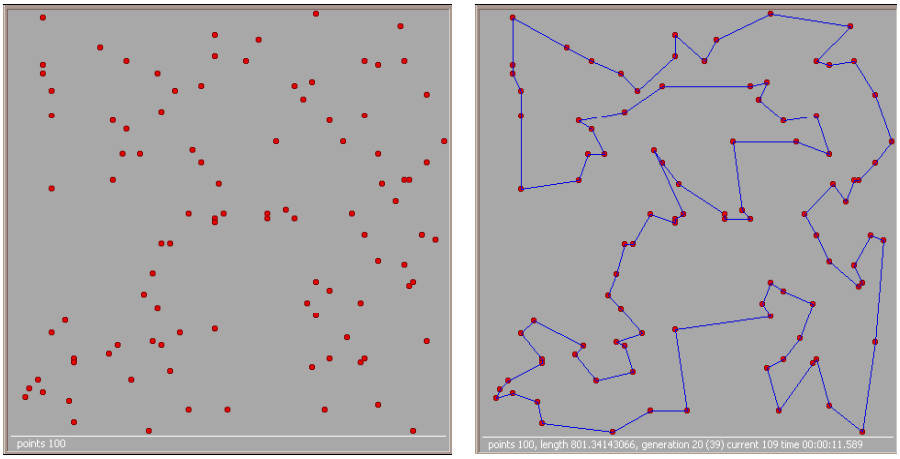
Tabulka 1: Faktoriál počtu objektů

Počet objektů	Počet posloupností
1	1
2	2
3	6
4	24
10	3628800
20	$2.4 * 10^{81}$
30	$2.7 * 10^{32}$
40	$8.2 * 10^{47}$
50	$3.0 * 10^{64}$
60	$8.3 * 10^{81}$

Typické problémy pro genetické algoritmy

Ilustrujme si příklad známého problému obchodního cestujícího: obchodní cestující má navštívit několik měst, ale nechce se mu strávit na cestách moc času. Přesněji tedy musí

navštívit všechna města, vrátit se domů (tj. do města, ze kterého vyjel) a přitom urazit co nejkratší vzdálenost. Problémem je zde tedy nalezení takové posloupnosti měst, kde cesta mezi nimi bude nejkratší možná.

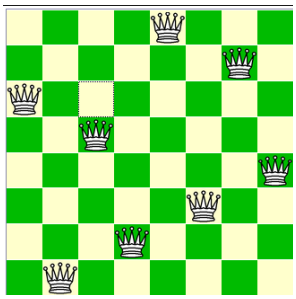


Obrázek 1: Problém obchodního cestujícího, zdroj [6]

Pokud máme nějakou posloupnost, můžeme velmi rychle určit celkovou délku trasy. Zaručeně nejlepší řešení umíme nalézt pouze tak, že probereme všechny možnosti (posloupnosti) a z nich vybereme tu nejlepší. Ohodnocení řešení (vzdálenost) umíme spočítat rychle. Problém je pouze v počtu řešení. Počet všech permutací n měst je n!=1*2*3*.*n. Pro 5 měst dostáváme 120 možností, což je počítačem snadno zvládnutelné. Již pro 20 měst však dostáváme 2432902008176640000 možností, což by počítač ohodnocující miliardu permutací za vteřinu počítal přes 77 let. I jen malým zvýšením tohoto počtu se dostáváme za hranice času předpokládané existence vesmíru.[22]

Musíme se tedy spokojit s tím, že optimální řešení nenalezneme. Člověk je však schopen nalézt dobré řešení v mnohem kratším čase. Není sice zaručené, že je to řešení optimální, ale pro praktické účely může být dostačující. A proč by něco takového nemohl umět i počítač?[2]

Dalším známým teoretickým příkladem pro genetické algoritmy je například problém osmi dam na šachovnici. Úkolem je umístit osm dam na šachovnici 8x8 tak, aby žádná dáma neohrožovala jinou.



Obrázek 2: Problém šachovnice, zdroj [7]

Podobně je konstruován známý problém batohu - máme sadu závaží a batoh a na letišti vám řeknou, že váš batoh musí vážit přesně tolik a tolik. Váš problém je pak určit, která závaží vložit do batohu a která zahodit.

Jak fungují genetické algoritmy

1.1.1 Jedinec a generace jedinců

Jedincem je míněn řetězec nul a jedniček. Pomocí tohoto řetězce je zakódováno místo v prostoru, které určuje možné řešení. Například pro prostor tvořený intervalem celých čísel mohou být jedinci reprezentováni jejich binárním vyjádřením. Později byla jedincem míněna i celá nebo dokonce reálná čísla. Samozřejmě lze vytvořit prakticky libovolnou reprezentaci jedince, tvořenou například vektory, maticemi nebo křivkami. Větší množství jedinců tvoří populaci, na kterou aplikujeme různé genetické operace. Populace jedinců v určitém kroku genetického algoritmu se nazývá generace. Všechny generace neboli všechna možná řešení se nazývají stavový prostor. [4]

1.1.2 Průběh genetického algoritmu

Na začátku, v první generaci, je populace složena z naprosto náhodných členů. V přechodu do nové generace je pro každého jedince spočtena tzv. *fitness* funkce, která vyjadřuje kvalitu řešení reprezentovaného tímto jedincem. Podle této kvality jsou stochasticky vybráni jedinci, kteří jsou modifikováni (pomocí mutací a křížení), čímž vznikne nová populace. Tento postup se iterativně opakuje, čímž se kvalita řešení v populaci postupně vylepšuje. Algoritmus se obvykle zastaví při dosažení postačující kvality řešení, případně po předem dané době.[23]

Vlastní genetický algoritmus můžeme shrnout do sedmi kroků

1) Návrh struktury
2) Inicializace
3) Ohodnocení
4) Selekce
5) Křížení
6) Mutace
7) Reprodukce

Operace 3) až 7) se postupně opakují a tvoří cyklus genetického algoritmu. [4]

1.1.3 Návrh struktury

Je nutné navrhnout strukturu jedince tak, aby se snadno vyjadřovala jeho kvalita a zároveň se dobře provádělo křížení. Pro náš účel jsme si zvolili generaci osmi řetězců délky 8 reprezentovanou nulami a jedničkami.

1.1.4 Inicializace

Inicializací rozumíme počáteční nastavení vektorů tvořících daného jedince, ve valné většině případů nastavujeme tyto vektory na nějaké náhodné hodnoty. Naši první generaci jsme vytvořili pomocí generátoru náhodných čísel a dostali tak zcela náhodně vytvořenou generaci, která vypadá takto:[9]

Tabulka 2: Náhodná generace, zdroj [9]

11110011
01010101
00011101
00111001
01100000
10000011
00000111
11111000

1.1.5 Ohodnocení

Jedná se o výpočet kvality jedince (např. v případě binární reprezentace čísla jeho dekadické vyčíslení). Popřípadě u řešení úloh s omezením "špatným" jedincům kvalitu penalizujeme. Tím se sníží jejich šance dostat se do další populace, popřípadě se zvýší šance na nahrazení novým jedincem.

Necháme naší funkci ***f*** zjistit, jak dobré řetězce jsme dostali:

Tabulka 3: Fitness jedinců, zdroj [9]

f(11110011) = 1,794
f(01010101) = 8,889
f(00011101) = 4,031
f(00111001) = 6,943
f(01100000) = 9,389
f(10000011) = 9,992
f(00000111) = 1,067
f(11111000) = 1,067

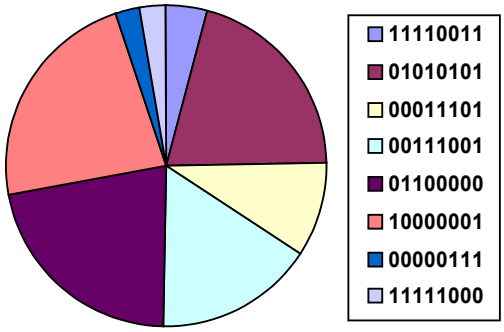
Jak vidíme v tabulce, dostali jsme tři silné řetězce *2*, *5*, *6* a tři slabé řetězce *1*, *7* a *8*. Naším cílem je, aby algoritmus na základě znalostí kvality řetězců vytvořil novou generaci, na jejímž složení se budou více podílet silnější řetězce (v našem případě *2*, *5*, *6*). Menší nebo vůbec žádný podíl na tvorbě generace potom budou mít slabší řetězce (*1*, *7*, *8*). Celý mechanismus tvorby nové generace se děje ve třech krocích. Prvním z nich je selekce.[9]

1.1.5.1 Selekce

Při selekci se do nové generace kopírují řetězce ze staré generace. Čím je řetězec lepší, tím větší je pravděpodobnost, že bude do nové generace okopírován. Jedním ze základních způsobů, jak je selekce prováděna, je výběr pomocí vážené rulety. Váženou ruletu si můžete představit jako klasickou ruletu pouze s jedním podstatným rozdílem: U klasické rulety máme pro každé číslo, které může být vylosováno, stejně velkou část kruhu, indikující výběr daného čísla. U vážené rulety je pravděpodobnost výběru určena kvalitou řetězce. Pravděpodobnost, že bude řetězec vybrán, se spočítá následovně: Nejdříve sečteme kvalitu všech jedinců v populaci. Pravděpodobnost pro daný řetězec potom získáme tak, že jeho kvalitu podělíme získaným součtem. Rozložení pravděpodobností na ruletě v našem příkladě můžete shlédnout v následující tabulce a grafu.

Tabulka 4: Výpočet hodnot pro váženou ruletu, zdroj [9]

Řetězec	*f*	*pravděpodobnost (%)*
11110011	1,794	4,16
01010101	8,889	20,59
00011101	4,031	9,34
00111001	6,943	16,08
01100000	9,389	21,75
10000011	9,992	23,14
00000111	1,067	2,47
11111000	1,067	2,47
průměrná kvalita řetězce	5,397	



Graf 1: Vážená ruleta, zdroj [9]

Představme si tedy, že ruletu *8*krát roztočíme, a získáme tak novou generaci. Řekněme, že námi získaná generace bude vypadat následovně:

Tabulka 5: Nová generace jedinců, zdroj [9]

Řetězec	*F*
00111001	6,943
01010101	8,889
01010101	8,889
10000011	9,992
01100000	9,389
10000011	9,992
00000111	1,067
01100000	9,389
Průměrná kvalita	8,069

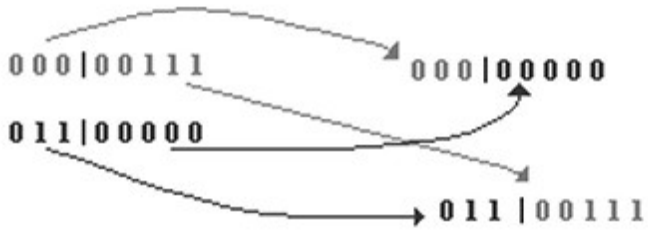
Kdybychom však pro vývoj generace používali pouze opakovaně prováděnou reprodukci, degenerovala by generace takovým způsobem, že by se v ní po čase objevovalo několik nejlepších řetězců z počáteční generace. Nedosáhli bychom tak v žádném případě toho, že by se v nových generacích objevovali řetězce, které by byly lepší, než řetězce v počáteční generaci. Za účelem zlepšování kvality řetězců jsou zde zbývající dva operátory.[9]

1.1.5.2 Křížení

Tato operace spočívá ve výměně informací mezi řetězci. Celý mechanismus křížení začíná tak, že se z nově vylosované populace náhodně vyberou dva řetězce. Tyto dva řetězce lze chápat jako dva rodiče. Jejich potomci budou potom obsahovat genetické informace svých rodičů. Každý z potomků bude přitom obsahovat z každého předka pouze část informace.

Jelikož v řetězci délky *8* existuje sedm míst, kde jej lze rozdělit na dvě části, zvolíme nejdříve místo, kde k rozdělení dojde. To učiníme náhodnou volbou čísla v rozmezí od jedné do sedmi.

Padne-li nám například číslo *3*, znamená to, že první potomek bude mít první tři pozice v řetězci shodné se svým prvním rodičem. Zbývajících *5* pozic bude doplněno informacemi ze shodných pozic druhého rodiče. Přesně naopak je tomu u druhého potomka. Celý mechanismus křížení našich dvou řetězců si můžete prohlédnout na obrázku.



Obrázek 3: Křížení

Křížením nám vzniknou dva zcela nové řetězce, které si ovšem nesou některé informace od svých rodičů. Podívejme se, jaká je kvalita potomků vzešlých z křížení.

- $f(00000000) = 0$
- $f(01100111) = 9{,}631$

První potomek dopadl při vyhodnocení zcela katastrofálně a díky nulové kvalitě nemá šanci projít přes další provedení reprodukce. Oproti tomu druhý potomek je silnější než kterýkoli z jeho rodičů. [9]

1.1.5.3 Mutace

Posledním krokem ve vytváření nové generace je mutace. Mutace příliš nezasahuje do vzhledu nové generace, ale plní velmi důležitou funkci. Ukažme si jednu z nich na jednoduchém příkladě. Řekněme, že víme, že nejlepší řetězec, který může vzniknout, obsahuje na páté pozici jedničku. Náhodným losováním jsme dostali generaci, v níž žádný jedinec neobsahuje na páté pozici jedničku. V tomto okamžiku nemůžeme nikdy získat nejlepšího jedince pouze pomocí reprodukce a křížení. Nejlepší jedinec by mohl vzniknout pouze tak, že by některý jedinec na páté pozici zmutoval a v některém dalším křížení by

vhodně předal svoji informaci. Mutace může také pomoci v situaci, kdy jsou si všechny řetězce velmi podobné.

Ukažme si ještě, jaký vliv by mohla mít mutace na některý z řetězců v naší generaci. Například zmutuje-li poslední řetězec – 01100111 na čtvrté pozici, dostaneme řetězec 01110111. Tento řetězec je silnější než řetězec původní (*f*(01100111)= *9,631*, *f*(01110111)= *9,956*). To znamená, že mutace nám v tomto případě pomohla zlepšit kvalitu daného řetězce.[9]

Tabulka 6: Mutace, zdroj [9]

Řetězec	01100111
Mutovaný řetězec	01110111
Fitness původního	f(01100111)= 9,631
Fitness mutovaného	f(01110111)= 9,956

1.1.5.4 Reprodukce

Posledním krokem je reprodukce. Tato operace promítne výsledky výše uvedených algoritmů do následující generace.

1.1.6 Kód genetického algoritmu

Zapsaný ve formě pseudo-kódu by standardní genetický algoritmus vypadal následovně:

Tabulka 7: Pseudokód genetického algoritmu, zdroj [4]

```
T := 0
                          Inicializuj počáteční
Initialize G(0)
                          generaci
Evaluate G(0)             Ohodnoť
Do while not Done
T := t + 1
Select G(t) from G(t-1)   vykonej přirozený výběr
Crossover G(t)            aplikuj křížení
Mutate G(t)               aplikuj mutaci
Evaluate G(0)             Ohodnoť
Replace G(t-1) with G(t)  Ohodnoť
Loop
```

Ze současné generace budou jedinci vybraní pro smrt odstraněni a nahrazeni novými jedinci, kteří vzniknou okopírováním kvalitních jedinců nebo křížením. Vzniká nová generace, která by při vhodném nastavení všech faktorů měla být kvalitnější než předchozí.

2 Praktické využití genetických algoritmů

Co umožňuje technologie genetických algoritmů

Genetické algoritmy umožňují řešení prakticky jakékoliv úlohy. V dnešní době můžeme říci, že snad neexistuje oblast, kde by nebyla tato výpočetní metoda použita. Genetický algoritmus byl aplikován na řešení velmi složitých problémů, přičemž byla získána často nečekaná a pozoruhodná řešení, klasickými metodami nedostupná. Na principu evolučního vývoje lze optimalizovat procesy, řídit systémy i vybírat nejlepší variantu z tisíců možností. V současné době jsme schopni řešit prakticky jakoukoliv úlohu. Čím je úloha složitější, tím kvalitnější je výsledné řešení.

Použití genetických algoritmů přináší výhody především v řešení těch úloh, kde klasický přístup neposkytuje nejlepší řešení nebo zcela selhává. Zásadní výhodou genetických algoritmů je také rychlost výpočtu. Aplikace běžně nepřesáhnou hranici 10 minut, přestože se mnohdy jedná o tisíce položek s desítkami nebo stovkami podmínek. Počet podmínek je další předností tohoto výpočtu. Je totiž neomezený.[10]

Genetické algoritmy a jejich využití v systému výroby

2.1.1 Operativní řízení výroby

V podnikové praxi se velmi často objevuje problém vícekriteriálního rozhodování za omezené dostupnosti potřebných informací. Právě tento typ problémů je však základním zdrojem dalšího růstu produktivity práce. Příkladem může být problematika operativního řízení výroby.

Efektivní využití podnikových zdrojů je silně závislé na souladu mezi logistikou a kapacitním plánováním výroby. Systémů ERP (Enterprise Resource Planning), které se snaží tuto problematiku řešit, je celá řada. Praxe ale ukazuje, že prokazatelně úspěšné jsou jen ve výrobě velkosériové a hromadné. Problém je v tom, že klasické systémy plánují výrobu za předpokladu neomezených výrobních kapacit a skutečné kapacity jsou zahrnuty až následně některým z modulů CRP (Capacity Requirements Planning) a o konečných úpravách plánu rozhoduje plánovač. Tento řetězec přípravy operativních plánů přestává být efektivní s klesající velikostí dávky, snižováním velikosti skladů a zaváděním tzv. štíhlé výroby.

Řešení tohoto problému bylo hledáno za použití klasických matematických metod. Bylo sice dosaženo pozoruhodných výsledků, ale podstata problému vyřešena nebyla. Klasické matematické metody, jako jsou např. lineární programování, statistika nebo regresní a korelační analýza, jsou omezené různými požadavky (linearita, počet vzájemných vztahů, rozsah hodnot apod.). Navíc existují problémy, kde vůbec není možné tyto metody použít, protože je nelze popsat tradičními způsoby. Existence zpětných vazeb, skokové závislosti, náhodné rozhodování, poruchy, změna požadavků - to vše jsou aspekty, které jsou stále častěji součástí praktických problémů a které způsobují selhání klasických metod. [14]

Takže problém není jen v tom, že co víme, neumíme řešit, ale také v tom, že nevíme, co nevíme.

Proto stále vzniká mnoho nových metod a postupů, které se snaží tento problém uspokojivě řešit. Jednou z těchto metod je využití genetických algoritmů.

V běžné praxi nás nezajímají všechna řešení lepší než zadané podmínky, ale alespoň jedno řešení, které zadaným podmínkám vyhovuje. Proto je pro nás důležitější schopnost vyhledat řešení s možností sledovat a řídit průběh tohoto hledání podle aktuálních podmínek, než získat řešení plné příčinných vztahů a podmínek existence z nepřesných nebo neaktuálních dat. [14]

2.1.2 Čím genetické algoritmy zaujaly podnikovou praxi

Pro přiblížení problému si představme výrobu a celou logistiku zajišťující tuto výrobu jako gigantickou n-rozměrnou šachovnici, kde všechny prvky, které se účastní výroby, jsou obdobou figurek a úspěšně sehraná partie je vlastně nějaký produkt realizovaný na trhu. Ti, co znají problém, vědí, že současná výpočetní technika má problémy s klasickou dvourozměrnou šachovnicí o 64 polích (příkladem je Problém osmi královen, který je uveden v první části), a to ještě nemusí řešit problém, že většina figurek musí být včas na správném místě.

Zde se projevuje jedna velmi užitečná vlastnost genetických algoritmů. Jedinci v rámci jedné generace jsou volně rozptýleni po celém n-rozměrném prostoru omezeném pouze okrajovými podmínkami. Díky stavbě genetického algoritmu máme jistotu, že pokud některý jedinec nalezne vzhledem k zadaným podmínkám zajímavé řešení, tak ho příští generace bude zdokonalovat. Tento proces učení již v zárodku eliminuje velké množství variant neúspěšných a zároveň dochází k paralelnímu vyhledávání přípustných řešení. [14]

Klasické algoritmy jsou závislé na reálnosti matematického modelu systematicky prohledávající prostor dobrých i špatných řešení, a tak nutně, od jistého stupně složitosti problému, překračují meze dané potřebou údajů v reálném čase za přijatelné náklady. Mnoho praktiků se snaží obejít tento problém a dopouští se neoprávněné linearizace a zjednodušení vstupních podmínek. Důsledkem jsou přesné výpočty s nepřesnými čísly, dochází ke ztrátě informace, chybám a náročný výpočet zpravidla slouží pouze jako alibi.

Další velmi výhodnou vlastností řešení založených na genetických algoritmech je možnost v průběhu výpočtu sledovat, který z parametrů nebo která z podmínek se stává omezující, případně naopak kterého parametru není dosahováno a řešení začíná konvergovat mimo pásmo přípustnosti. Lze pak operativně a velmi jednoduše zasáhnout formou změny priorit, termínů nebo změnou podmínek a ve vyhledávání přípustného řešení pokračovat bez jakékoliv ztráty informací z dosavadního průběhu výpočtu. U klasických metod je při jakékoli změně výchozích podmínek potřeba celý výpočet opakovat od začátku.

Datová náročnost, doba řešení problému a snadnost ovládání realizovaných programů je na takové úrovni, že je bez problémů možno zařadit tyto systémy do dílenského plánování. Na druhé straně výkonnost a dosahované výsledky předurčují systémy využívající tyto genetické algoritmy k přehodnocení mnoha přístupů vrcholového managementu a k nahrazení dosavadních velmi drahých a náročných systémů v kvalitě i v čase.[14]

2.1.3 Vhodné úlohy pro genetické algoritmy v podnikové praxi

Převaha genetických algoritmů se projevuje u složitých úkolů. Obecně je možné říci, že se jedná o rozsáhlé nelineární problémy rozhodování za velké neurčitosti s mnoha faktory ovlivňujícími výsledek řešení. Tedy takové problémy, kde klasické metody neznají algoritmus řešení nebo jsou pro datovou náročnost nepoužitelné.

Jeden z důvodů síly genetických algoritmů je v samotné podstatě algoritmu hledání řešení, které spočívá v tom, že nehledáme nějaké určité řešení, ale skupinu přípustných řešení, z nichž vybíráme to nejlepší. Od klasických metod se genetické algoritmy odlišují způsobem, kterým se přibližujeme k řešení. Převaha genetických algoritmů se projevuje ve složitých problémech s mnoha parametry.[14]

2.1.4 Současný stav využití genetických algoritmů

Ve světě existuje již řada praktických využití genetických algoritmů ve všech oblastech lidského podnikání. Kupříkladu v oblasti výroby zemědělských strojů, ve střední Americe, určitá firma vyrábí různé zemědělské stroje na zakázku podle přání jednotlivých zákazníků. Dá se tedy říci, že každý dodaný stroj je unikátní a tedy velikost výrobní dávky je velmi blízká 1. Při rozsahu sortimentu a krátkých termínech dodávek všechny klasické metody řízení výroby selhávaly, docházelo k neustálým posuvům termínů a hromadění zásob a výroba se potýkala s neúměrně velkou rozpracovaností. Vzhledem ke stále se měnícím podmínkám a novým objednávkám se objevila potřeba každý den upravovat operativní plán výroby.

Genetické algoritmy zde našly uplatnění v nástroji pro sestavení a doplnění operativního plánu výroby.

Centrální počítač po skončení každého pracovního dne předá potřebné údaje do výroby, kde přímo na hale běžně dostupné PC přes noc projde tisíce přípustných operativních plánů a připraví je pro zahájení příští směny. Tak je nalezeno přijatelné řešení z hlediska současného stavu výroby a nových podmínek a požadavků. [14]

2.1.5 Náročnost genetických algoritmů na výpočetní techniku

Aplikace výše popsaných metod nevyžaduje nejen radikální změnu struktury firmy, ale dokonce ani změnu technického vybavení. Čím je to umožněno:

1) Velmi složité exaktní výpočty u klasických metod vyžadující velké množství parametrů a požadavky na data, která prakticky není možné jednorázově dodat. Toto nahrazují genetické algoritmy efektivním způsobem vyhledávání mnohanásobným opakováním jednoduchých operací - náhodný výběr (selekce), křížení a mutace v prostředí, které zajišťuje učení.

2) Datová náročnost je vzhledem k použité technologii značně nižší. Dochází k značné úspoře při pořizován dat.

3) Problematicky algoritmizovatelné řešení je nahrazeno výběrem nejlepšího řešení z velkého množství přijatelných řešení.

4) Při přerušení úlohy není nutno zahájit výpočet od začátku, jak je tomu u klasických metod, ale je možno pokračovat ve výpočtu.

5) Při změně počtu nebo rozsahu parametrů je nutno u klasických metod zpravidla měnit algoritmus výpočtu, což není v požadovaném reálném čase prakticky možné a omezení, která vyplývají z nutných zjednodušení prakticky vylučují jejich použitelnost. Genetické algoritmy umožňují měnit parametry i během výpočtu. Změny mají vliv pouze na délku výpočtu a umožňují tak uživateli sestavit operativní plán, který je možno kdykoli přizpůsobit měnícím se podmínkám, a tak lze nalézat průchodná řešení za respektování mnoha podmínek, parametrů, priorit a termínů. Algoritmus nás též upozorňuje na parametry, které umožní plán zprůchodnit nebo vylepšit.

Přes veškerou jednoduchost pro uživatele systémů na bázi genetických algoritmů představuje nasazení takovýchto produktů jednu změnu, a to v interpretaci určitých jevů a trénink způsobu jak na tyto jevy reagovat.[14]

Genetické programování ve výzkumu

2.1.6 Využití genetických algoritmů v klinickém výzkumu

Cesta UI i genetického programování do praktických aplikací byla dlážděna řadou překážek, nicméně některé projekty v oblasti finančnictví či zdravotnictví se již zdají být poměrně slibné. Totéž se týká i nasazení v průmyslu.

Při dolování dat ve finančnictví nebo v klinickém výzkumu nemocnosti velkých populací se již několik let používá genetických algoritmů. Děje se tak ve fázi předcházející analýze dat pomocí indukce nebo neurálních sítí.

Nejprve je třeba vyřešit otázku, na jaké intervaly rozdělit naměřené hodnoty cílové proměnné (např. mezi 100 % zdravím a 100 % úmrtností). Poté následuje volba upřednostněné časové řady. Kde máme nechat program pátrat po možných souvislostech, které by mohly být např. příčinami alergií? Je vhodnější analyzovat trendy hodinové, týdenní, měsíční, nebo víceleté? V případě, že je nutno analyzovat více trendů paralelně, musíme rozhodnout o nastavení relativních vah. Počet možných kombinací dosahuje astronomických hodnot, a proto se zde plně projeví flexibilita genetického algoritmu. K výsledku se někdy podaří dojít i během pouhých několika desítek iterací (generací).[11]

2.1.7 Klinické studie

Průkopníkem dolování dat (a samozřejmě především souvislostí mezi nimi) ze statistiky nemocnosti v EU a USA byla mj. společnost Meditelligence. Jedna z prvních velkých klinických studií za pomoci jejich metod a softwaru, která byla provedena v univerzitní nemocnici v Göteborgu, našla a vyhodnotila souvislost mezi dědičností a vznikem alergií. Tato kategorie vědeckého softwaru je přitom schopna hypotézy nejen testovat, ale také je ze zadaných dat automaticky navrhovat.[11]

Ve vývojových odděleních se dnes genetické algoritmy často používají tam, kde je potřeba najít alespoň nějaké uspokojivé řešení, ať již bude nalezený způsob skutečně optimální nebo ne. Programy obvykle rychle a snadno najdou lokální extrémy (můžeme si je představit jako náhorní planiny v případě lokálního maxima, anebo, pokud jde o minimalizaci cílové funkce, naopak důlky na cvrnkání kuliček), kde potom vyhledávání alternativ uvázne na téže — relativně uspokojivé, nikoliv však optimální — hodnotě cílové funkce.

Genetické programování skutečně někdy vede k lokálnímu, nikoliv globálnímu optimu, přestože se používá mutací, tzn. víceméně náhodných variací na „genetické obohacení“ algoritmu. V této souvislosti však lze poznamenat, že i celá řada relativně inteligentních zvířat nedokáže přijít na to, jak obejít plot, za kterým jim badatel položil potravu jako návnadu; také nejsou schopná se od potravy dočasně vzdálit. Taktéž lidé často upřednostňují krátkodobý prospěch.[11]

2.1.8 Expertní systémy

Genetické programování se uplatňuje i ve vývoji průmyslových postupů, zde však obvykle není jedinou technologií z oblasti umělé inteligence. Často bývá doplňováno znalostní technikou, především expertními systémy. Ve velkých švédských podnicích je např. používán software americké společnosti Invention Machine CoBrain a Tech Optimizer. Základ báze znalostí systému se zrodil ještě v týmu bývalého sovětského patentového úřadu.

Vývojový tým společnosti Ericsson vyřešil díky softwaru od Invention Machine problém s vyzařováním prvních vylehčených mobilních telefonů do hlavy uživatele. Za účelem minimalizace vlnění kolem telefonu navrhl expertní systém nasadit hliník. Aby vyhověl omezujícím podmínkám provozu (cena, váha), došel nakonec až k hliníkové pěně. [11]

2.1.9 Technický výzkum

Při vývoji ve firmě Electrolux zase navrhl software IM řešení problému s přisáváním koberců na vysavač; zvýšení efektu motoru vede do slepé uličky (vývěvy), ale jeho snížení vede naopak ke zhoršení funkčních vlastností přístroje. Řešení nabídnuté expertním systémem bylo i v tomto případě uspokojivé: Nechat cirkulovat vzduch, tzn. vhánět na koberec zhruba tolik vzduchu, kolik se ho zároveň vysává. Oba popsané problémy by samozřejmě dokázali vyřešit i lidé, proces by však trval déle a zřejmě by si vyžádal celkově vyšší náklady.

Aplikace z oblasti umělé inteligence se také stále častěji používají při validaci, např. při komentování a ověřování konstrukčních výkresů navržených lidmi. Jde tedy o jakousi nadstavbu nad programy typu CAD (Computer-Aided Design). Ve Skandinávii se již po léta používá znalostní technologie při ověřování bezpečnosti ropných těžebních věží v Severním moři již ve stádiu CAD výkresu.[11]

2.1.10Nejistý výsledek genetické programování

Vědci ze Sussexské univerzity v Brightonu se pomocí genetického programování se pokusili vyvinout oscilátor. Dospěli však k výsledku, který sice splňoval požadované zadání, byl však dosažen způsobem, jenž by je nejspíš ani ve snu nenapadl.[12]

Paul Layzell a Jon Bird z univerzity v Brightonu se pokusili použít genetického programování při experimentu, v rámci něhož měl počítač z připojeného obvodu s deseti tranzistory vytvořit oscilátor. Tranzistory mohou být mezi sebou různě propojovány, což dává dostatečné možnosti kombinací. Vědci tedy v počítači řídícím pokus reprezentovali každý jednotlivý spoj mezi dvěma tranzistory jako jeden gen jedince. Jelikož účelem bylo vyprodukovat takové spojení tranzistorů, které by vytvářelo oscilující obvod, definovali „hodnotící funkci“ jako funkci, která udává, jak je momentálně vyprodukovaná oscilace kvalitní (podobná dané sinusoidě). [12]

Výsledný obvod skutečně osciloval. Umělá evoluce si však našla jiný způsob, jak zadání splnit. Když vědci vytvořené schéma zkoumali, ukázalo se, že mají v rukou spíše radiopřijímač než oscilátor. Obvod totiž místo toho, aby vytvářel oscilace sám, vytvořil z tranzistorů spoj tak, že vzniknul kus rovného vodiče, fungující jako anténa - a přijímal oscilace vyzařované nedaleko stojícím počítačem.

Problém onoho experimentu spočíval v tom, že funkce fitness se nezjišťovala výpočtem, ale napojením na reálný fyzikální systém. Proto mohla evoluce dojít k odlišnému řešení, než předpokládali autoři. Vedle stojící počítač byl součástí reality, nikoliv však jejího matematického modelu.

Takové řešení plně splňuje zadání a vyhovuje hodnotící funkci, nicméně není tím, co se od něj očekávalo. Kdyby vědci přenesli zařízení do jiné místnosti nebo vypnuli onen počítač, zajisté by přestalo fungovat.

Celý háček spočíval v tom, že si výzkumníci neuvědomili takovouto možnost podvádět, a nezajistili dostatečně statické testovací prostředí. Těžko jim to můžeme zazlívat, odhalit předem všechny možné vlivy a podmínky není možné. Nejcennější na celém experimentu je však právě konfrontace s přímostí a elegancí "přírodního" výběru. Genetické algoritmy nemají žádnou inteligenci ani paměť, po změně vnějších podmínek by nejspíš k výsledku došly očekávaným postupem a obvod by osciloval sám o sobě. Z tohoto hlediska je řešení "podvodem" poměrně křehké, neboť nepřenosné, nicméně nejjednodušší za daných podmínek a plně vyhovující danému prostředí - což je přesně to, co se odehrává v rámci evoluce na Zemi po miliony let. [12]

Evoluce vždy upřednostní nejvhodnější řešení pro aktuální podmínky bez ohledu na budoucí stav. Změní-li se podmínky, řešení přestává vyhovovat a zaniká. V této souvislosti se nabízí některé možné paralely i ve vztahu k evoluci člověka. Snadno může ústit do slepé uličky - naše řešení se může ukázat jako příliš specializované a nepřenosné.[12]

2.1.11 Kde klasické metody selhávají

Genetické algoritmy lze využít v řízení výroby, řešení problémů v průmyslu, dopravě a logistice, finančnictví, matematice a mnoha dalších oborech. Zaměřují se tedy především na úlohy, kde klasické metody zcela určitě selhávají. Jedná se např. o řízení výrobního procesu. Genetický algoritmus dokáže optimálně navrhnout výrobní linku a zahrnout přitom do řešení spoustu okolních parametrů, jakými jsou např. možnost provedení dané operace na více strojích, libovolné následnosti v technologickém postupu, výpadek některého z pracovišť nebo jednotlivé operace, zahrnutí časových prodlev a nákladů spojených s přechodem od jednoho pracoviště k druhému, definice maximální délky fronty každého pracoviště atd. Podobnou úlohou je sestavení a optimalizace logistických řetězců, opět se zahrnutím mnoha

doprovodných parametrů nebo optimalizace obslužných systémů spočívající v nalezení optimálního počtu jednotlivých kanálů systému, nebo vytvoření různých variant reagujících na různou úroveň přísunu požadavků.

Moderní doba s sebou přináší stále nové a nové problémy k řešení. Protože se jedná o problémy z praxe, jsou velmi složité. K jejich řešení je nutné využít co nejlepších metod. A jednou z nich jsou genetické algoritmy. [13]

3 Implementace genetických algoritmů do prostředí Matlabu

MATLAB

MATLAB je integrované prostředí pro vědeckotechnické výpočty, modelování, návrhy algoritmů, simulace, analýzu a prezentaci dat, měření a zpracování signálů, návrhy řídicích a komunikačních systémů. MATLAB je nástroj, jak pro pohodlnou interaktivní práci, tak pro vývoj širokého spektra aplikací.

Nově k nim přibyl Genetic Algorithm Toolbox, který rozšiřuje optimalizační možnosti Matlabu o nástroje pracujícími s genetickými algoritmy. Je využitelný zejména v úlohách, které jsou obtížně řešitelné tradičními optimalizačními metodami, včetně úloh, které nejsou dobře definovatelné nebo je lze obtížně modelovat matematicky. Jsou použitelné také v případech, kdy nejsou počítané funkce spojité, či jsou vysoce nelineární, stochastické nebo mají nejisté nebo neznámé derivace.[21]

Implementace GA do prostředí Matlabu

Genetické algoritmy spadají do oblasti metod kombinatorické optimalizace. Při hledání například globálního extrému lze zakódovat body na funkci tak, že k nim lze přistupovat jako ke kusům dědičného materiálu. Poté lze s výhodami použít různých způsobů modifikace dědičného materiálu jako v přírodě (křížení, mutace). Řízení těchto rekombinačních kroků je možné parametrizací nebo (což je přirozenější) selekcí. Zde se dostáváme k parametrizaci, neboli řízení heuristiky.[17]

Síla parametrizace genetických algoritmů je v toolboxu lákavá. Nabízí se zde možnosti jako:

1) Mutace a křížení lze řídit elitřstvím, řízenou mutací (určení jak silně bude populace mutována a jak bude mutace v průběhu generací ubývat), řízením křížení (zde je možno určit, kolik procent populace z neelitního podílu generace se bude křížit)
2) Nastavením maximálního počtu generací
3) Řízením výběru rodiče na základě jeho fitness
4) Poměření fitness, což umožňuje snazší nalezení vhodných rodičů pro budoucí generaci

5) Hendikepováním známé oblasti hledání, kde tušíme, že zde pravděpodobně globální extrém neleží
6) Užití vyhledávacího algoritmu na dohledání optima poté, co GA ukončily svou činnost
7) Není možno nastavit hranice prostoru, jež bude prohledáván.[17]

Pro realizaci GA byla navržena sada funkcí. Vlastní proces programové realizace a nasazení GA je možné rozdělit na dvě základní části:

1) Navržení příslušné ohodnocovací - fitness funkce, a to ve formě funkce m-file.
2) Implementace a začlenění funkcí GA ve vlastním programu. Tento proces zahrnuje nastavení parametrů a operátorů GA. [16]

Můžeme si také zvolit, zda budeme volat funkci GA pomocí příkazového řádku, užitím toolboxu, grafického prostředí pro genetické algoritmy, anebo kombinací obou uvedených metod.

Realizace genetického algoritmu prostřednictvím příkazového řádku

3.1.1 Syntaxe algoritmu

Příkazovým řádkem se funkce genetického algoritmu volá následující syntaxí:

```
[x fval] = ga(@fitnessfun, nvars, options)
```

Kód 1: Syntaxe genetického algoritmu

kde

- `x` je proměnná dosahující řešení
- `fval` je konečná hodnota fitness funkce
- `@fitnessfun` je funkce fitness
- `nvars` je počet nezávislých proměnných pro fitness funkci
- `options` jsou možnosti nastavení struktury genetického algoritmu.

3.1.2 Řešení jednoduché optimalizační funkce

Mějme příklad, kdy chceme minimalizovat funkci o dvou proměnných s následujícím předpisem:

```
min f(x) = 100 * (x(1)^2 - x(2)) ^2 + (1 - x(1))^2;
```

tak, aby vyhovovala následujícím požadavkům:

```
x1*x2 + x1 - x2 + 1.5 <=0,

10 - x1*x2 <=0,

0 <= x1 <= 1,

0 <= x2 <= 13
```

Nejprve vytvoříme M-file který pojmenujeme fitness_funkce.m. Bude obsahovat následující kód, který definuje fitness funkci pro daný problém:

```
Function y = funkce_fitness(x)

y = 100 * (x(1)^2 - x(2)) ^2 + (1 - x(1))^2;
```

Kód 2: Fitness funkce

Výpočetní jednotka předpokládá, že fitness funkce vloží proměnnou x, kde x je řádkový vektor s mnoha prvky, které reprezentují počet proměnných v daném problému. Fitness funkce propočítá hodnotu funkce a vrátí její skalární hodnotu jako argument y.

Druhý M-file se bude týkat omezujících podmínek, které jsme zvolili pro naši funkci. Obsahuje předpis funkce:

```
Function [c, ceq] = omezujici_podminky(x)

   c = [1.5 + x(1)*x(2) + x(1) - x(2);

   -x(1)*x(2) + 10];
```

Kód 3: Omezující podmínky pro finess funkci část 1.

Abychom minimalizovali naší fitness funkci pomocí GA funkce, voláme funkci z příkazového řádku, kde současně definujeme počet proměnných a limity proměnných:

```
ObjectiveFunction = @fitness_funkce;

nvars = 2;

LB = [0 0];    % spodní hranice

UB = [1 13];   % horní hranice

ConstraintFunction = @omezujici_podminky;
```

```
[x,fval] = ga(ObjectiveFunction,nvars,[],[],[],[],LB,UB,
...ConstraintFunction)
```

Kód 4: Omezující podmínky pro fitness funkci část 2.

Operace proběhla. Zobrazují se výsledky

```
Best              max               Stall

Generation        f-count           f(x)      constraint  Generations

    1             943               13623            0       0

    2             1666              13578.2          0       0

    3             2374              13578.2          0       0

x = 0.8122   12.3122

fval =   1.3578e+004
```

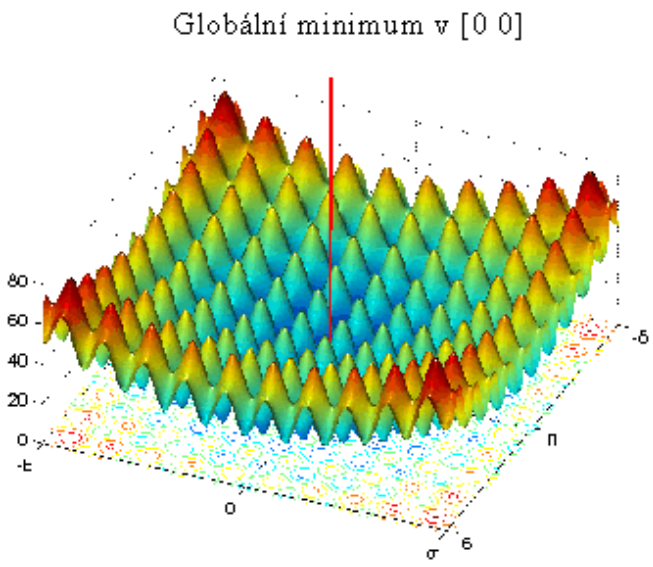
Získáme tedy číselná řešení, která vyhovují stanoveným podmínkám. Vizualizace dat probíhá pomocí toolboxu, grafickém uživatelské prostředí, kterému se věnuje následující část práce.

Genetický algoritmus vytvořený pomocí toolboxu

3.1.3 Rastriginova funce

Rastriginova funkce se velmi dobře hodí k řešení minimalizačních problémů. Tato funkce má má mnoho lokálních minim, ale pouze jedno globální minimum, které se nachází v bodě [0;0], pokud hodnota funkce je 0, v ostatních případech bude hodnota funkce vyšší.

Rastriginova funkce se často užívá k testování genetických algoritmů, protože mezi mnoha lokálními minimy se běžnými metodami hledá globální maximum jen velmi obtížně.



Graf 2: Rastriginova funkce, zdroj [20]

Pro dvě nezávislé proměnné je tato funkce definována:

$$Ras(x) = 20 + x_1^2 x_2^2 - 10(\cos 2\pi x_1 + \cos 2\pi x_2)$$

3.1.4 Nastavení parametrů GA

Nejprve je nutné zadat do toolboxu parametry funkce.

```
LB = [0 0];  UB = [10 20];
```

Spodní a horní hranice možných řešení algoritmů.

```
PopulationSize: 20
```

Naše populace se bude skládat z dvaceti jedinců.

```
Initial range [0;1]
```

Předpokládáme, že hledané řešení leží v pravém horním kvadrantu, proto nastavíme vstupní oblast populace [0;1].

```
EliteCount: 2
```

Algoritmus vybere 2 elitní jedince (jedince s nejvyšší fitness funkcí), kteří postoupí do další generace, ostatní budou dále modifikováni.

```
CrossoverFraction: 0.8000
```

Udává, ve kterém místě řetězce proběhne křížení.

```
Generations: 100
```

Maximální limit počtu generací.

```
TimeLimit: Inf
```

Časový limit pro genetický algoritmus, v tomto případě není definován.

```
FitnessLimit: -Inf
```

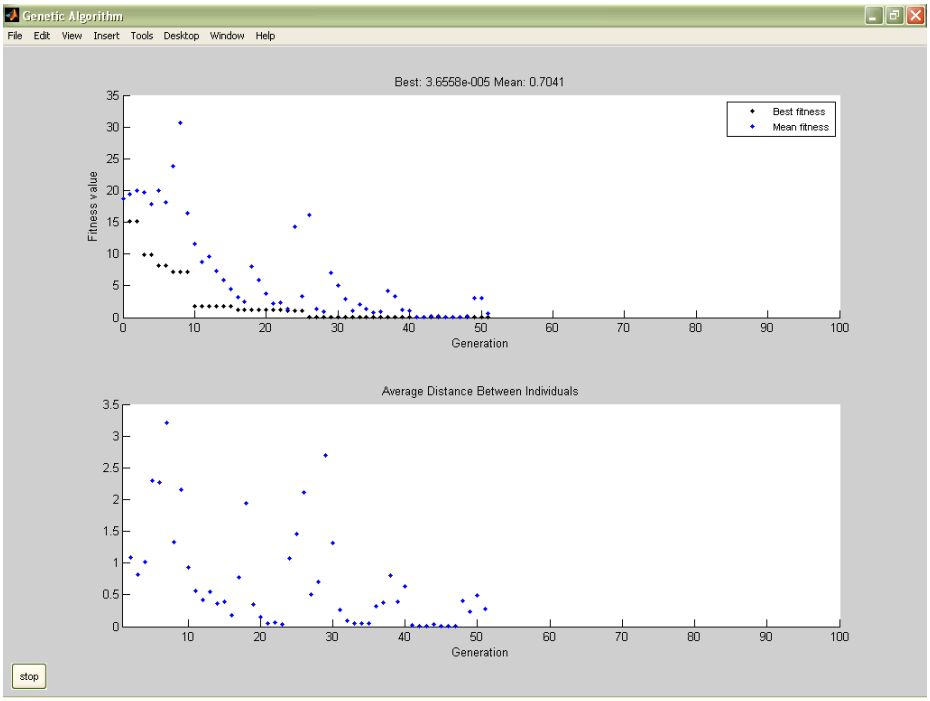
Fitness limit. Pokud by byl definován, algoritmus by se zastavil při jeho dosažení.

```
FitnessScalingFcn: @fitscalingrank

SelectionFcn: @selectionstochunif

CrossoverFcn: @crossoverscattered

MutationFcn: @mutationgaussian
```

Nastavení funkcí pro výpočet fitness, selekci, křížení a mutaci. Tyto funkce jsou v Matlabu předdefinované.

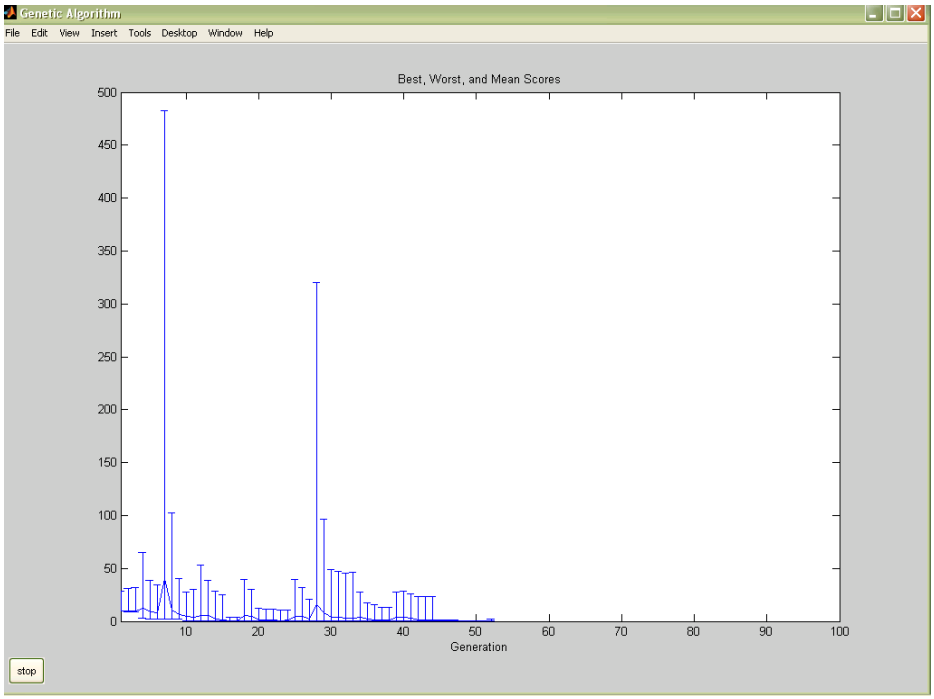
3.1.5 Vizualizace výsledků Rastriginovy funkce

Poté, co algoritmus proběhl, se zobrazují výsledky. Na následujícím grafu vidíme zobrazené nejlepší a mezní finess jedinců, druhý graf zobrazuje vzdálenost mezi jedinci.



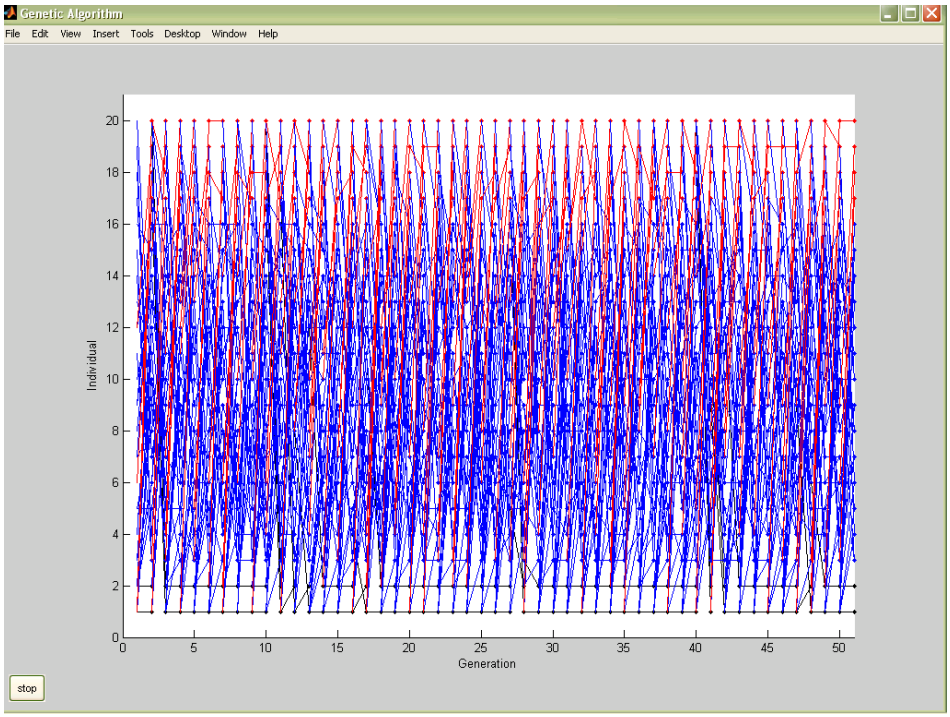
Graf 3: Nejlepší a mezní fitness, distance

Výsledky jedinců. Můžeme vidět, že jsme v našem příkladu měli dva favority na globální minimum.



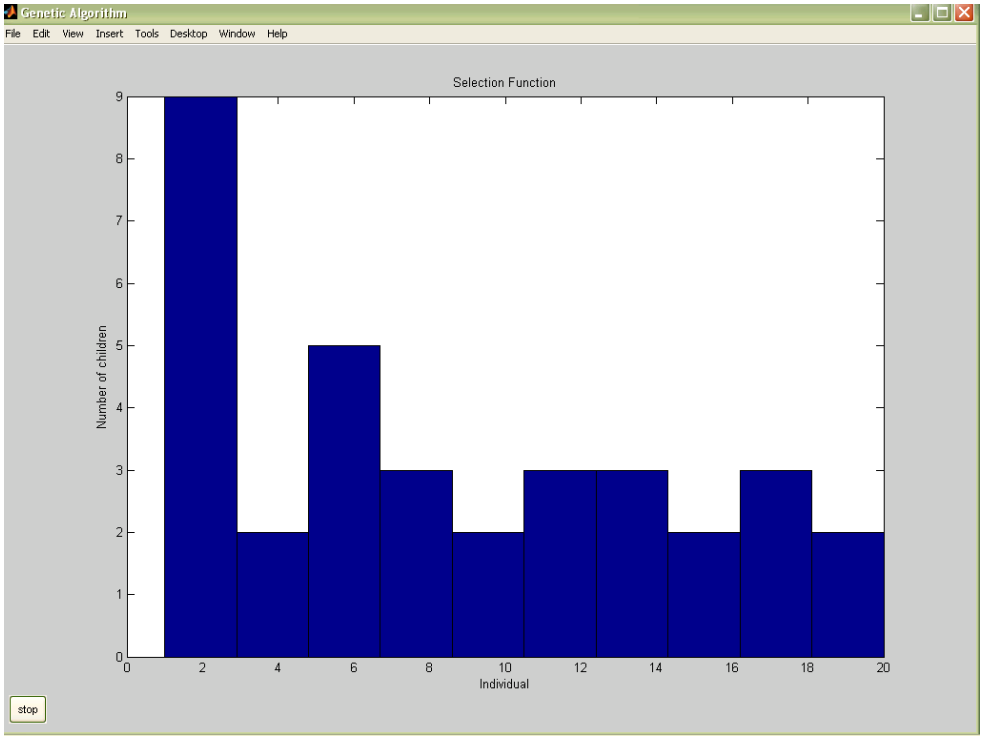
Graf 4: Nejlepší, nejhorší a mezní výsledky

Genealogie: Zobrazuje kombinace a míšení genů v jednotlivých krocích algoritmů.



Graf 5: Genealogie

Funkce selekce. Zobrazuje jedince a počet jejich potomků. U prvního jedince je zřejmý vysoký počet, tedy i vysoká fitness funkce.



Graf 6: Selekce

Numerické vyjádření výsledků:

Fitness function value: 1.0445268192195485

Final point: Čísla jsou odpovídají souřadnice, které stanovují globální miminum Rastriginovy funkce.

1) 0,44244

2) 9.10163e-10 ≈ 0

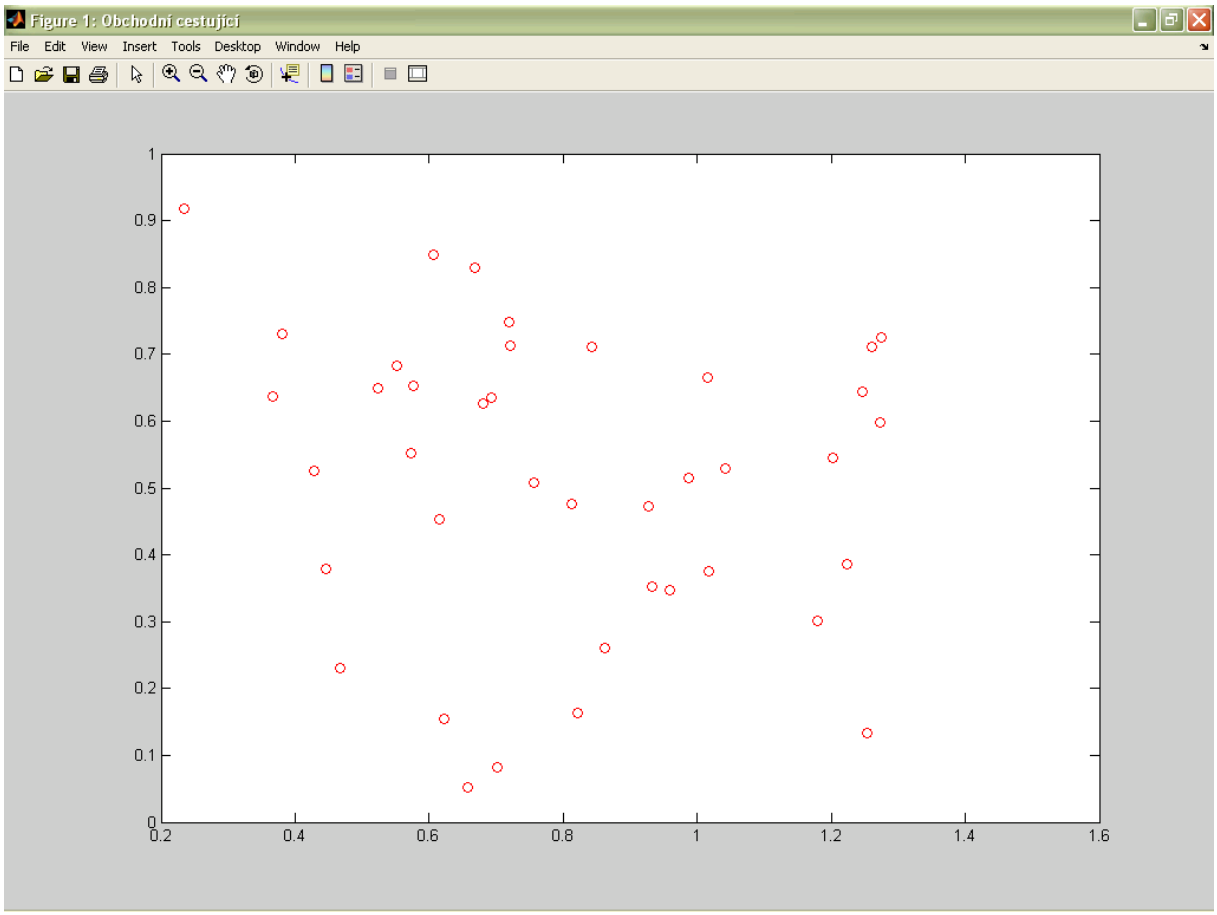
Běžný optimalizační problém řešený genetickými algoritmy

3.1.6 Problém obchodního cestujícího

V následující části této práce je ukázáno, jakým způsobem se genetické algoritmy uplatňují v řešení optimalizačních úloh.

Jak již bylo zmíněno v první části této práce, Problém obchodního cestujícího je úloha, ve které je stanoven počet měst, které musí cestující libovolně projet, a kdy je vzdálenost cesty mezi městy známá. Cílem je nalézt nejkratší cestu mezi všemi městy tak, aby obchodník strávil na cestě co nejméně času.

Červená kolečka na následujícím obrázku reprezentují rozmístění měst, které obchodník musí procestovat a dodat nebo vyzvednout zboží.



Graf 7: Problém obchodního cestujícího

3.1.7 Lokace v prostoru

Máme tedy zadána umístění měst. První krok, který podnikneme, je výpočet matice vzdáleností pro všechna města.

```
distances = zeros(cities);

for count1=1:cities,

    for count2=1:count1,

        x1 = locations(count1,1);

        y1 = locations(count1,2);

        x2 = locations(count2,1);

        y2 = locations(count2,2);

             distances(count1,count2)=sqrt((x1-x2)^2+(y1-
y2)^2);

    distances(count2,count1)=distances(count1,count2);

    end;

end;
```

Kód 5: Lokace v prostoru, zdroj [20]

3.1.8 Propojení v prostoru

Nyní vytvoříme matici permutací vzdáleností jednotlivých měst (neboli jedinců). Funkci pro matici permutací označíme (P), kde každý element reprezentuje uspořádaný sled měst jako permutovaný vektor. To znamená, že obchodník bude cestovat v pořadí specifikovaném v P_i. Funkce vrátí matici o velikosti populace.

```
Function pop =
create_permutations(NVARS,FitnessFcn,options)
```

Kód 6: Permutační funkce

Tato funkce nám tedy vytvoří populaci permutací, každou délky `NVARS`.

Argumenty funkce jsou

- `NVARS`: Počet proměnných
- `FITNESSFCN`: Fitness funkce
- `OPTIONS`: Možnosti nastavení struktury užité pro GA

```
totalPopulationSize = sum(options.PopulationSize);

n = NVARS;

pop = cell(totalPopulationSize,1);

for i = 1:totalPopulationSize

    pop{i} = randperm(n);

end
```

Kód 7: Permutace, zdroj [20]

3.1.9 Křížení

Tato funkce vykoná křížení matice permutací a populace a vrátí matici potomků, kteří jsou kombinací rodičů.

```
Function xoverKids  =
crossover_permutation(parents,options,NVARS, ...

    FitnessFcn,thisScore,thisPopulation)
```

Kód 8: Funkce křížení

`CROSSOVER_PERMUTATION` je vstupní křížící funkce.

`XOVERKIDS` funkce výsledných permutací

Argumenty funkcí jsou

- `PARENTS`: Rodiče, kteří byli vybráni na základě selekce
- `OPTIONS`: Možnosti volby generované z GAOPTIMSET
- `NVARS`: Počet proměnných
- `FITNESSFCN`: Fitness funkce
- `THISSCORE`: Vektor současné populace
- `THISPOPULATION`: Matice jedinců současné populace

```
nKids = length(parents)/2;

xoverKids = cell(nKids,1);

index = 1;

for i=1:nKids

    parent = thisPopulation{parents(index)};

    index = index + 2;

    % V tomto okamžiku se z dětí stávají rodiče

    p1 = ceil((length(parent) -1) * rand);

    p2 = p1 + ceil((length(parent) - p1- 1) * rand);

    child = parent;

    child(p1:p2) = fliplr(child(p1:p2));

    xoverKids{i} = child;

end
```

Kód 9: Křížení, zdroj [20]

3.1.10Mutace

Mutační funkce mění pořadí v řetězci jednotlivých jedinců. Vrátí mutovaný pořádek permutací.

```
Function mutationChildren = mutate_permutation(parents
,options,NVARS, ...

    FitnessFcn, state,
thisScore,thisPopulation,mutationRate)
```

Kód 10: Funkce pro mutaci

`MUTATE_PERMUTATION`: vstupní mutační funkce

`MUTATIONCHILDREN`: výstupní funkce mutovaných permutací

Argumenty funkce jsou

- `PARENTS`: Rodiče, které budeme mutovat
- `OPTIONS`: Možnosti volby nastavení GA

- NVARS: Počet proměnných
- FITNESSFCN: Funkce fitness
- THISSCORE: Vektor současné populace
- THISPOPULATION: Matice jedinců současné populace
- MUTATIONRATE: Míra mutace

```
mutationChildren = cell(length(parents),1)

for i=1:length(parents)

parent = thisPopulation{parents(i)}

p = ceil(length(parent) * rand(1,2));

child = parent;

child(p(1)) = parent(p(2));

child(p(2)) = parent(p(1));

mutationChildren{i} = child;

end
```

Kód 11: Mutace, zdroj [20]

3.1.11 Fitness funkce

Nyní je třeba stanovit fitness funkci. Fitnes je celková vzdálenost pro daný sled měst v X.

```
function scores =
traveling_salesman_fitness(x,distances)
```

Kód 12: Definice fitness funkce

- SCORES propočítává fitness jedince.
- DISTANCE(A,B) je vzdálenost města A od města B

```
Scores = zeros(size(x,1),1);

for j = 1:size(x,1)

p = x{j};

    f = distances(p(end),p(1));

    for i = 2:length(p)

        f = f + distances(p(i-1),p(i));

    end

    scores(j) = f;

end
```

Kód 13: Výpočet fitness funkce, zdroj [20]

GA volá fitness funkce jako jediný argument „x“. Naše fitness funkce má v tuto chvíli dva argumenty: x a vzdálenost. Proto užijeme anonymní funkci k zachycení hodnoty přidaného argumentu, vzdálenostní matici. Vytvoříme fitness funkci FitnessFcn, která vezme jediný vstup x a dva vstupní parametry se přiřadí jako parametry funkce.

```
FitnessFcn = @(x)
traveling_salesman_fitness(x,distances);
```

Kód 14: Výsledná fitness funkce

3.1.12Volitelné nastavení toolboxu

Nyní vytvoříme nastavení struktury možností pro výpočetní zařízení, aby nastavilo data typ a populační rozmezení.

```
Options = gaoptimset('PopulationType',
'custom','PopInitRange', ...

    [1;cities]);
```

Kód 15: Volitelné nastavení toolboxu, část 1.

Vybereme funkce vytvoření permutací jedinců, křížení a mutace a zadáme ukončovací podmínky pro algoritmus (pomocí StallGenLimit).

```
Options =
gaoptimset(options,'CreationFcn',@create_permutations,
...

    'CrossoverFcn',@crossover_permutation, ...

    'MutationFcn',@mutate_permutation, ...

    'Generations',500,'PopulationSize',60, ...

    'StallGenLimit',200,'Vectorized','on');
```

Kód 16: Volitelné nastavení toolboxu, část 2.

3.1.13Zavolání konečné funkce

Nakonec zavoláme výslednou funkci genetického algoritmu s našimi zadanými informacemi.

```
numberOfVariables = cities;

[x,fval,reason,output] =
ga(FitnessFcn,numberOfVariables,options)
```

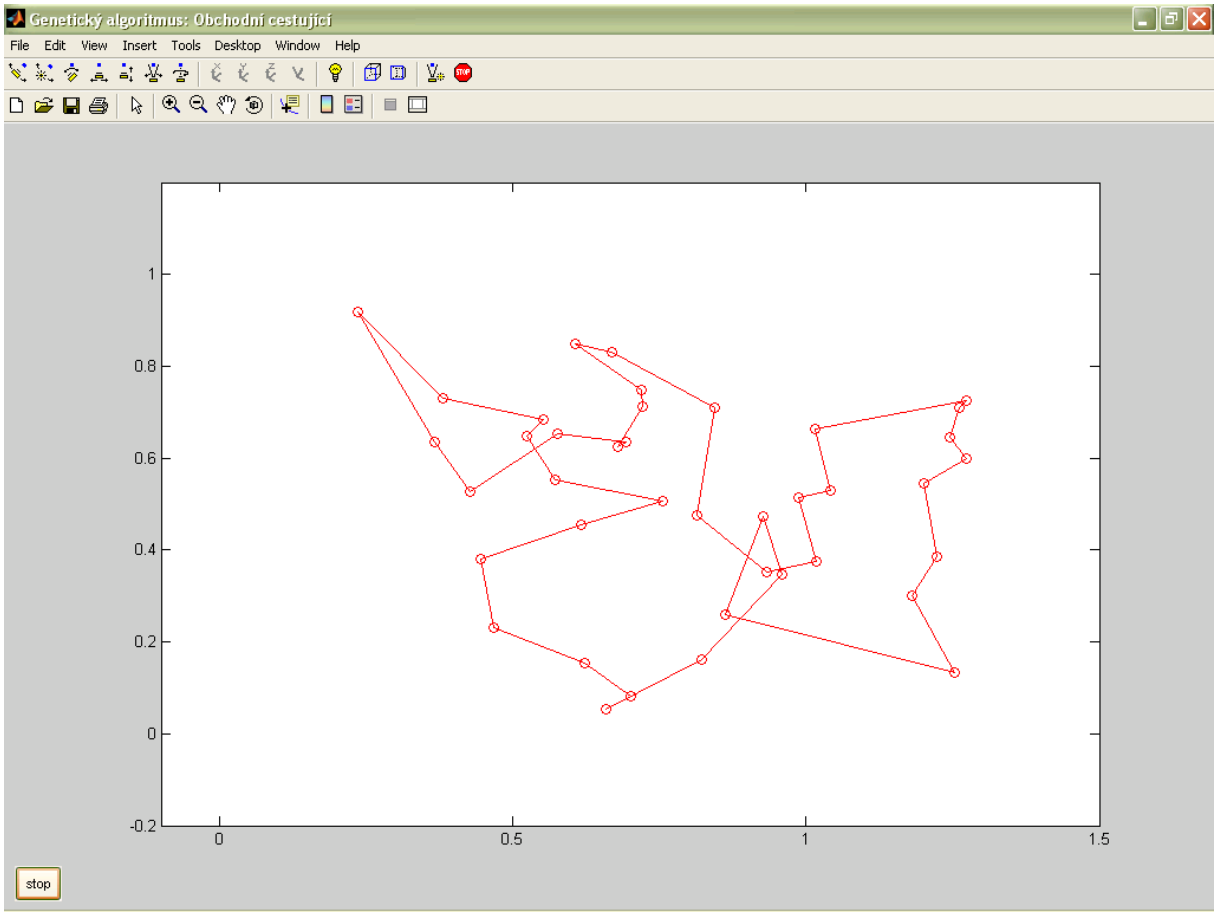
Kód 17: Volání funkce

3.1.14Zobrazení výsledků

Matlab propočítal proměnné a zobrazil následující řešení problematiky:

```
Optimization terminated: average change in the fitness value less than
options.TolFun.

x = [1x40 double]

fval =5.8573

output =

      randstate: [35x1 double]

      randnstate: [2x1 double]

      generations: 203

            funccount: 12180

            message: [1x86 char]
```

Zadaná kritéria výsledku jsou převedena do následujícího grafu:



Graf 8: Genetický algoritmus: trasa pro obchodního cestujícího

4 Závěrečné shrnutí

Tato bakalářská práce se zabývá řešením běžných optimalizačních úloh s využitím techniky genetických algoritmů. Genetické algoritmy patří do třídy evolučních algoritmů, které mimo ně zahrnují také evoluční programování, evoluční strategii a genetické programování. Jsou to vyhledávací algoritmy založené na mechanismu přirozeného výběru a principech genetiky. Jejich velkou výhodou je poměrná jednoduchost.

V první kapitole jsme se seznámili s ideovým vzorem pro genetické algoritmy - principy vývoje, které se uplatňují v přírodě. Zde existují populace jednotlivých živočišných druhů, složených z jedinců různých vlastností. Tyto vlastnosti jsou prvotně zakódovány v jejich genech, které tvoří větší celky, chromozómy. Při křížení vznikají noví jedinci, kteří mají zpravidla náhodně část genů od jednoho rodiče a část genů od rodiče druhého. Přitom ve zvlášť výjimečném případě může dojít k náhodné změně některého genu v chromozómu, tzv. mutaci, která může být pro další vývoj druhu příznivá nebo ne. Podle svých vlastností má každý z potomků větší nebo menší schopnost obstát v přirozeném výběru a vytvořit další generaci. Proces výběru se stále opakuje a v jeho průběhu se zlepšují genetické vlastnosti daného druhu. Tak probíhala celá evoluce v přírodě.

V teorii umělé inteligence je genetický algoritmus proces postupného vylepšování populace jedinců opakovanou aplikací genetických operátorů, který vede k evoluci takových jedinců, kteří lépe vyhovují stanoveným podmínkám než jedinci, ze kterých vznikli. Proces konverguje k situaci, kdy je populace tvořena jen těmi nejlepšími jedinci.

Druhá kapitola se věnuje praktickému využití genetických algoritmů v moderním světě. V praxi se pomocí genetických algoritmů řeší úlohy optimalizace, využívají se k vyhledávání nejlepší topologie, v technologii a výrobě a v průmyslové automatizaci a jako alternativní metody učení neuronových sítí. Na rozdíl od gradientních metod, které reprezentují hledání lokálního minima nebo maxima pomocí jednoho zpřesňujícího se řešení, představují genetické algoritmy jiný přístup, který používá populaci prozatímních řešení, jež paralelně procházejí parametrický prostor a navzájem se ovlivňují a modifikují pomocí genetických operátorů. Tím se dosahuje toho, že populace jedinců najde správné řešení rychleji, než kdyby se prohledával prostor izolovaně.

Praktický příklad využití evoluční optimalizace byl uveden v kapitole třetí. Genetický algoritmus může představovat cestu pro řešení rozličných matematických i technických úloh. V této práci je ukázán možný přístup k řešení optimalizačních problémů využitím

programovacího prostředí MATLAB a jeho toolboxu, který významně rozšiřuje optimalizační možnosti Matlabu o nástroje pracujícími s genetickými algoritmy. Je využitelný zejména v úlohách, které jsou obtížně řešitelné tradičními optimalizačními metodami, včetně úloh, které nejsou dobře definovatelné nebo je lze obtížně modelovat matematicky, což je také problematika hledání nejkratší cesty mezi více místy známá jako Problém obchodního cestujícího.

V toolboxu lze snadno a přehledně navolit předpoklady a parametry pro daný genetický algoritmus, aniž bychom museli nutně znát přesnou syntaxi příkazů nutných pro práci v příkazovém řádku. Kromě číselných výstupů současně obsahuje sadu výstupních grafů, ve které je průběh genetického algoritmu postupně zaznamenáván a z které získáme poměrně přesnou představu, jak se vyvíjel náš algoritmus krok za krokem. Výsledkem práce v Matlabu je tak elegantní a komplexní řešení daného problému, které je také, jak již vyplývá z povahy genetických algoritmů, šité na míru reálným podmínkám.

5 Seznam obrázků, grafů, kódů a tabulek

6 Literatura

1. KAJÁNEK, Petr, PŘIKRYL, Josef. *Seznámení se s genetickými algoritmy* [on-line]. 19.3.2006. Dostupné z: <http://mujweb.cz/www/prikrylj/Genetickealgoritmy.html>.
2. OBITKO, Marek. *Genetické algoritmy – matrice života v počítačích* [on-line]. 19.3.2006. Dostupné z <http://www.scienceworld.cz/sw.nsf/ID/9B80774399F1B837C1256E9700489AEC?OpenDocument&cast=1>.
3. ÁRVAYOVÁ – ZEŤÁKOVÁ, Zuzana, SABO, Michal. *Evolučné algoritmy* [on-line]. 19.3.2006. Dostupné z <http://alife.tuke.sk/index.php?clanok=85> a <http://alife.tuke.sk/index.php?clanok=116>.
4. KUCHAŘÍK, Milan. *Genetické algoritmy* [on-line]. 19.3.2006. Dostupné z <http://kfe.fjfi.cvut.cz/~kucharik/projects/ga/pojmy.html>.
5. TEDA, Jaroslav. *Genetické algoritmy a jejich aplikace v praxi* [on-line]. 20.3.2006. Dostupné z <http://www.programujte.com/view.php?cisloclanku=2005072601>
6. VOMLEL, Jiří. *Genetické algoritmy* [on-line]. 17.5.2006. Dostupné z <http://staff.utia.cas.cz/vomlel/slides/presentace-ga.pdf#search=%22genetick%C3%A9%20algoritmy%22>.
7. VOMLEL, Jiří. *Laboratoř inteligentních řešení* [on-line]. 17.5.2006. Dostupné z <http://staff.utia.cas.cz/vomlel/slides/presentace-ga.pdf>
8. HRADECKÝ, Tomáš. *Praktikum z informatiky, shrnutí referátu kolegy Vůjtěcha na téma Genetické algoritmy z 9.5.2001* [on-line]. 20.5. 2006. Dostupné z <http://sweb.cz/hradtom/dcv/praktikum/recenze4.doc>.
9. MALÝ, Marek. *Genetické algoritmy* [on-line]. 20.5. 2006. Dostupné z <http://alpha.ujep.cz/~maly/vyuka/ruzne/neuronky-genetika/Ga.doc>.
10. Autor neuveden. Internetové stránky firmy Hestley. *Genetické algoritmy – web* [on-line]. 5.10.2006. Dostupné z <http://www.hestley.com/hestley/cz/geneticke-algoritmy-reseni.htm>.
11. KRATOCHVÍL, Milan. *Cesty umělé inteligence a genetického programování* [on-line]. 5.10.2006. Dostupné z <http://www.scienceworld.cz/sw.nsf/ID/B583DF2CA0CC1A93C1256E970048C6CC?OpenDocument&cast=1>.
12. MIKLÍK, Aleš. *Genetické programování: anténa místo oscilátoru* [on-line]. 6.10.2006. Dostupné z

<http://www.scienceworld.cz/sw.nsf/ID/9D33983E234172F8C1256E970048C43E?OpenDocument&cast=1>.

13. NEKVINDA, Michal. *Genetické algoritmy a jejich využití v řízení výroby, I. část* [on-line]. 7.11.2006. Dostupné z <http://www.systemonline.cz/site/trendy/syst1_2.htm>.
14. NEKVINDA, Michal. *Genetické algoritmy a jejich využití v řízení výroby, II. část* [on-line]. 7.11.2006. Dostupné z <http://www.system.ccb.cz/site/trendy/syst2_2.htm>.
15. JINDRA, Jindřich. *Centrální plánování železnice pomocí Matlab Web Serveru – bakalářská práce* [on-line]. 30.1.2007. Dostupné z <http://dce.felk.cvut.cz/dolezilkova/diplomky/2005/bp_2005_jindra_jindrich/bp_2005_jindra_jindrich.pdf>.
16. MATOUŠEK, Radek. *Aplikace GA v prostředí Matlab* [on-line]. 30.1.2007. Dostupné z <http://dsp.vscht.cz/konference_matlab/matlab00/matousra.pdf>.
17. BLAŽEK, J., HABIBBALA, V., PAVLISKA, V. *Vybrané heuristiky pro globální optimalizaci a jejich implementace v Matlabu* [on-line]. 31.1.2007. Dostupné z <http://www.volny.cz/habiballa/publ/matlab05.pdf>.
18. MARTIKAINEM, Jarno, OVASKAZ, Seppo J. *Hierarchical Two-Population Genetic Algorithm* [on-line]. 31.1.2007. Dostupné z <http://www.ripublication.com/ijcirv2/ijcirv2n4_6.pdf>.
19. TEDA, Jaroslav. *Genetické algoritmy a jejich aplikace v praxi – pokračování* [on-line]. 1.2.2007. <http://www.programujte.com/tisk_clanku.php?cisloclanku=2005072601-Geneticke-algoritmy-a-jejich-aplikace-v-praxi>.
20. Autor neuveden. Oficiální stránky společnosti The MathWorks. *Genetic Algorithm and Direct Search Toolbox 2.0.2. – web* [on-line]. 1.2.2007. Dostupné z <http://www.mathworks.com/products/gads/demos.html>.
21. Autor neuveden. Stránky skupiny Humusoft. *Matlab, popis produktu* [on-line]. 1.2.2007. Dostupné z <http://www.humusoft.cz/matlab/matlab.htm>
22. ŠTEFKA, David. *Alternativy k evolučním optimalizačním algoritmům - diplomová práce.*[on-line]. 1.2.2007. Dostupné z <http://www.insophy.cz/doc/stefka_optimalizace.pdf>.
23. Kolektiva autorů, Internetová encyklopedie Wikipedie. *Genetický algoritmus.*[on-line]. 1.2.2007. Dostupné z <http://cs.wikipedia.org/wiki/Genetick%C3%BD_algoritmus>.
24. KVASNIČKA, V., POSPÍCHAL, J., TIŇO, P. *Evolučné algoritmy.* STU Bratislava

2000. 80-227-1377-5.

25. MAŘÍK, V., ŠTĚPÁNKOVÁ, O. *Umělá inteligence (1).* 1. vyd. Academia, Praha 1993. 80-200-0496-3.
26. MAŘÍK, V., ŠTĚPÁNKOVÁ, O., LAŽANSKÝ, J. *Umělá inteligence (3)*. 1. vyd. Academia, Praha, 2001. 80-200-0472-6.
27. OLEJ, V. *Modelovanie ekonomických procesov na báze výpočtovej inteligencie*, Miloš Vognar- M&V, Hradec Králové, 2003. 80-90329-9-1.
28. DEMEL, J. *Grafy a jejich aplikace.* 1.vyd. Praha, Academia, 2002. 80-200-0990-6.
29. LAWLER, E.L., LENSTRA, J.K., RINNOOY KAN, A.H.G., SHMOYS, D.B. The Traveling Salesman Problem, A Guided Tour of Coinatorial Optialization, Wiley, Chichester, 1985.
30. PEKNY, J.F., Miller, D.L. A parallel branch-and-boun algoritm for solving large asymetric traveling salesman problems, *Mathematical Programming* 55(1992)17.
31. KUTIL, M. Scheduling Toolbox First Preview. In MATLAB 2004 – *Sborník příspěvků 12. ročníku konference* .Praha VŠCHT, 2004. 297-303. 80-7080-550-1.

ÚDAJE PRO KNIHOVNICKOU DATABÁZI

Název práce	Genetické algoritmy ve zvoleném softwarovém prostředí
Autor práce	Michala Maděrová
Obor	Systémové inženýrství a informatika
Rok obhajoby	2007
Vedoucí práce	Ing. Jan Panuš
Anotace	Cílem této práce je studium genetických algoritmů obecně, jejich využití v podnikové praxi a lékařském a technologickém výzkumu a samotný návrh genetických algoritmů implementovaný do prostředí MATLABU.
Klíčová slova	Genetický algoritmus, genetické programování, Matlab, Genetic Algorithm Toolbox, Problém obchodního cestujícího.