ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA ELEKTROTECHNICKÁ

BAKALÁŘSKÁ PRÁCE

Řešení problému navrhování neúplných vyvážených bloků pomocí algoritmu POEMS

Praha, 2010 **Autor: Jan Sálus**

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne 25.5.2010

podpis

Poděkování

Děkuji především vedoucímu bakalářské práce Ing. Jiřímu Kubalíkovi, Ph.D. za podnětné připomínky a rady při řešení práce.

Abstrakt

Cílem této práce bylo naimplementovat algoritmus POEMS pro řešení problému navrhování neúplných vyvážených bloků (BIBD). K tomu jsem použil implementaci POEMS s oddělenou algoritmickou a problémově závislou částí. Navrhnul jsem několik variant, které vedou ke zlepšení algoritmu. Algoritmus POEMS jsem porovnal s několika jinými algoritmy, používanými pro řešení BIBD, na standardně používaných testovacích datech. Dosažené výsledky jsou slibné. POEMS nachází řešení srovnatelná s porovnávanými algoritmy.

Klíčová slova

kombinatorická optimalizace, evoluční algoritmy, lokální optimalizační algoritmy

Abstract

The goal of this work was to implement POEMS algorithm for solving problem of balanced incomplete blocks design (BIBD). I used implementation of POEMS with separate algorithmic and problem-dependent part. I designed several variations that led to improvement of algorithms. I compared POEMS algorithm with several other algorithms which are used for solving BIBD on the test data, standardly used. The achieved results are promising. POEMS solution is comparable with others.

Key words

combinatorial optimization, evolutionary algorithm, stochastic local search algorithms

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Jan Sálus

Studijní program: Softwarové technologie a management

Obor: Inteligentní systémy

Název tématu: Řešení problému navrhování neúplných vyvážených bloků pomocí algoritmu POEMS

Pokyny pro vypracování:

1. Prostudujte stávající klasické a netradiční postupy (založené např. na evolučních algoritmech) pro řešení Balanced Incomplete Block Design (BIBD) problému.
2. Prostudujte metody evolučních algoritmů a zaměřte se na algoritmus Prototype Optimization with Evolved Improvement Steps (POEMS).
3. Naimplementujte algoritmus POEMS pro řešení BIBD problému.
4. Experimentálně ověřte funkčnost algoritmu na standardních testovacích datech, dosažené výsledky vyhodnoťte a porovnejte s jinými algoritmy.

Seznam odborné literatury: Dodá vedoucí práce.

Vedoucí bakalářské práce: Ing. Jiří Kubalík, Ph.D.

Platnost zadání: do konce zimního semestru 2010/2011

L.S.

prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry

doc. Ing. Boris Šimák, CSc.
děkan

V Praze dne 5. 2. 2010

Obsah

Seznam obrázků **viii**

Seznam tabulek **ix**

1 Úvod **1**

2 Popis problému **3**

2.1 Základní popis 3

2.2 Složitost a existence 3

2.2.1 Počet řešení 6

2.3 Stavový prostor 7

2.4 Symetrický vzor 7

3 Použitý optimalizační algoritmus **8**

3.1 Obecný popis evolučních algoritmů 8

3.1.1 Reprezentace řešení 9

3.1.2 Inicializace počáteční populace 9

3.1.3 Genetické operátory 10

3.1.4 Fitness funkce 11

3.1.5 Model evolučního algoritmu 12

3.2 Obecný popis algoritmu POEMS 13

3.2.1 Evoluční algoritmus v POEMS 13

Jedinec 13

Populace 14

Fitness 14
Selekce 14
Nahrazení 15
3.2.2 Model POEMS algoritmu 15
3.3 Popis přizpůsobení POEMS pro řešený problém 16
3.3.1 Reprezentace řešení 16
3.3.2 Základní akce pro modifikace řešení 16
3.3.3 Ohodnocovací funkce 18
3.3.4 Inicializace počátečního řešení 19
3.3.5 Výběr akce 19
3.3.6 Modifikace algoritmu 20

4 Experimenty **22**
4.1 Popis porovnávaných algoritmů 22
4.1.1 Lokální prohledávání HC 23
4.1.2 Lokální prohledávání TS 23
4.1.3 Genetický algoritmus 24
4.2 Výčet vybraných testovacích problémů 24
4.2.1 Jak tyto problémy řešily porovnávané algoritmy 24
4.3 Konfigurace našeho POEMS 27
4.4 Vlastní experimenty 28

5 Závěr **32**

Literatura **35**

A Obsah přiloženého CD **I**

Seznam obrázků

3.1 Chromozom 8
3.2 Genom 9
3.3 Genotyp 9
3.4 Ruletový výběr 10
3.5 Turnajový výběr 10
3.6 Dvojbodové křížení 10
3.7 Vícebodové křížení 11
3.8 Mutace 11
3.9 Ukázka fitness funkce 12
3.10 Populace v POEMS 14
3.11 Aplikování akce $bit - flip$ s parametry col = 4, row = 2 17
3.12 Aplikování akce $swap - col$ s parametry col = 4, p_1 = 1, p_2 = 3 17
3.13 Aplikování akce $swap - row$ s parametry row = 4, p_1 = 1, p_2 = 5 18
3.14 Fronta 20

4.1 Vývoj řešení algoritmu POEMS 31
4.2 Vývoj řešení algoritmu POEMS s frontou o velikosti 5 31

Seznam tabulek

4.1 Tabulka vybraných BIBD problémů 24
4.2 Tabulka počtu uspěšného nalezení řešení 25
4.3 Tabulka výsledků lokálního prohledávání 25
4.4 Tabulka výsledků tabu prohledávání 26
4.5 Tabulka výsledků genetického algoritmu 26
4.6 Tabulka vlastních výsledků - řízená inic. a jedna akce v POEMS 28
4.7 Tabulka vlastních výsledků - řízená inicializace a jedna akce v EA 29
4.8 Tabulka vlastních výsledků - náhodná inicializace a dvě akce v EA 30
4.9 Shrnutí všech vlastních výsledků 30

Kapitola 1

Úvod

Generace blokového návrhu je velmi známý kombinatorický problém, který je velmi těžké vyřešit. Teorie blokového návrhu vznikla především díky práci R.A.Fishera a F.Yatese [1] v počátku 30. let 19 století. Tento problém má řadu variant, z nichž je jedna populárnější než ostatní a je to tzv. navrhování neúplných vyvážených bloků, anglická zkratka BIBD tj. Balanced Incomplete Block Designs (dále jen BIBD). Konstrukce BIBD zpočátku úřadovala jen v experimentální oblasti, nicméně dnes lze použít v různých oblastech jako je šifrování, teorie kódování a v řadě dalších oblastí. V těchto oblastech se setkáváme s kombinatorickými problémy, které k ochraně informace často využívají časové náročnosti výpočtu řešení tzn. je časově neúnosné čekat na dekódování informace, která je poté neplatná či zastaralá. Také BIBD je kombinatorický problém, řadící se do nedeterministicky polynomiálně těžkých (NP-těžkých) úloh. Třída úloh NP je třída rozhodovacích úloh, které lze vyřešit v polynomiálním čase na nedeterministickém Turingově stroji. NP-těžké úlohy jsou potom nějtěžší problémy v množině NP, které zasahují i nad tuto množinu úloh. Proto tento problém dobře slouží jako výkonnostní test pro řadu optimalizačních algoritmů a technik. Evoluční algoritmy, které se dnes úspěšně aplikují na řadu optimalizačních problémů, se výborně hodí pro optimalizaci těchto složitých problémů, protože jsou velmi robustní vůči prohledávanému okolí, které je u složitých úloh značně členité.

Motivací pro vytvoření této práce byl již zmíněný složitý kombinatorický problém, který jsem se rozhodl řešit pomocí genetického programování, konkrétně pomocí algoritmu POEMS tj. *Prototype Optimalization with Evolved Improvement Steps*. POEMS je stochas-

tický optimalizační algoritmus, který ve svém běhu opakovaně využívá evoluční algorimus ke zlepšení řešení. Samozřemě další motivací bylo vytvoření perfektního optimalizačního algoritmu, který by překonal ostatní algoritmy, používané na tento problém. Tím byla stanovena vysoká úroveň, ke které jsem se po vylepšující modifikaci algoritmu POEMS přiblížil.

První část práce se zabývá formulací problému BIBD jeho existencí a složitostí. Je zde kladen důraz na existenci řešení, podmínky nutné nikoliv postačující k tomu aby daná instance BIBD problému měla řešení. Udává celkový počet řešení BIBD a velikost jeho úplného okolí. V další části je detajně popsán evoluční a POEMS algoritmus. Dále popisuje přizpůsobení pro daný problém a také nová vylepšení stávající implementace algoritmu POEMS. Poslední část je určena experimentům, ve kterých výsledky z POEMS porovnávám s výsledky lokálního prohledávání, genetického algoritmu a tabu prohledávání, které jsou v této části stručně popsány.

Kapitola 2

Popis problému

2.1 Základní popis

BIBD je v podstatě definováno jako uspořádání z v různých objektů (kde objekt je řádkový vektor), v b blocích (kde blok je sloupcový vektor). Společně v a b tvoří matici výskytu (příklad 2.10). Jsou uspořádány tak, že každý blok b obsahuje přesně k odlišných objektů, každý objekt v se vyskytuje přesně v r blocích a každé dva rozdílné páry objektů (tudíž řádky matice) nastanou společně přesně v λ blocích (pro $k,\ r,\ \lambda > 0$) [2].

Standardní cesta pro reprezentaci BIBD je matice výskytu (incidenční matice) $M \equiv \{m_{ij}\}_{v \times b}$, což je binární $v \times b$ matice s přesně r jedničkami na řádek, k jedničkami na sloupec, λ jedniček pro vektor vzniklý z vynásobení každých dvou rozdílných řádků a kde $m_{ij} \in \{0, 1\}$ je roven 1, když i-tý objekt je obsažen v j-tém bloku a 0 jinak. V tomto kontextu reprezentuje m_{ij} výskyt objektu i v bloku j v matici M. BIBD je specifikován pěticí parametrů $\langle v,\ b,\ r,\ k,\ \lambda \rangle$, které jsou již popsané výše [2].

2.2 Složitost a existence

Generace BIBD je NP-těžký (nedeterministicky polynomiální) problém. Ve třídě NP lze úlohy řešit v polynomiálním čase na nedeterministickém Turingově stroji, tj. na počítači, který umožňuje v každém kroku rozvětvit výpočet na n větví, v nichž se posléze řešení hledá sou-

časně [13]. NP-těžké úlohy jsou potom nejtěžší problémy v NP, které často zasahují i nad tuto množinu NP úloh [13] a právě proto se BIBD stává perfektním výkonnostním testem. Problém BIBD má širokou škálu variant od jednoduchých až po ty složité. Škálovatelnost a složitost problému je vhodná k otestování chování různých programovacích technik či algoritmů [3].

Tento problém může být nezvladatelný i pro poměrně malé šířky matice výskytu M. Jako důkaz o obtížnosti problému je existence mnoha různých otevřených příkladů, které doposud nebyly vyřešeny (jistě to může být tím, že jejich řešení prostě neexistuje) [2]. Zda je daný problém řešitelný, lze ukázat pomocí Fisherovy nerovnosti, viz věta 2.4 a důkaz 3. Pro existenci BIBD problému musí být také splněny podmínky parametrů tj. věta 2.1 [3].

Věta 2.1 (Podmínky parametrů): *Konfigurace těchto pěti parametrů* $\langle v,\ b,\ r,\ k,\ \lambda\rangle$ *musi vyhovět následující vztahům:*

$$b \cdot k = v \cdot r \tag{2.1}$$

$$\lambda \cdot (v-1) = r \cdot (k-1) \tag{2.2}$$

Důkaz ($\boldsymbol{b \cdot k = v \cdot r}$): Máme v řádků a v každém takovém řádku se vyskytuje r prvků, pak $v \cdot r$ vyjadřuje celkový počet vyskytujících se prvků v matici výskytu. Na druhé straně máme b sloupců a každý sloupec obsahuje k prvků, tak číslo $b \cdot k$ také vyjadřuje celkový počet prvků.

Důkaz ($\boldsymbol{\lambda \cdot (v-1) = r \cdot (k-1)}$): Máme x a y (x a y jsou řádky matice výskytu), které se současně vyskytují v λ sloupcích, počet kombinací y k x je $v-1$ (matice výskytu má v řádků). Takže $\lambda \cdot (v-1)$ je počet párů s x. V x se objevuje přesně r prvků a každý prvek z x může být spárován s $k-1$ dalšími prvky (v každém sloupci je přesně k prvků). Pak $r \cdot (k-1)$ je počet párů s x.

Věta 2.2 (Laplaceova): *Nechť A a B jsou čtvercové matice. Pak* $det(A) \cdot det(B) = det(A \cdot B)$.

Věta 2.3 (Jednotková matice pro vzory): *Když A je matice výskytu pro konfiguraci* $\langle v, b, r, k, \lambda\rangle$ *pak*

$$det(A \cdot A^T) = (r-\lambda) \cdot I + \lambda \cdot J \tag{2.3}$$

kde I je $v \times v$ *jednotková (diagonální) matice a J je* $v \times v$ *matice tvořená ze samých jedniček.*

Věta 2.4 (Fisherova nerovnost): *Konstrukce BIBD s konfigurací* $\langle v,\ b,\ r,\ k,\ \lambda\rangle$ *existuje tehdy když platí* $b \geq v$.

Důkaz (Fisherovi nerovnosti): Předpokládejme, že $b > v$ a A je matice výskytu. K matici A můžeme přidat $v - b$ nulových sloupců a dostaneme libovolnou matici B o velikosti $v \times v$. Vzhledem k tomu, že nulové sloupce nezmění násobení matic, tak paltí $A \cdot A^T = B \cdot B^T$. Dále víme, že B je čtvercová matice a tedy podle věty 2.2 [9] můžeme sestavit následující rovnici:

$$det(A \cdot A^T) = det(B \cdot B^T) = det(B) \cdot det(B^T) = 0$$

kde $det(B) = 0$ je z důvodu nulových sloupců v matici B.
Podle věty 2.3 můžeme napsat následující jednotkovou matici

$$det(A \cdot A^T) = det \begin{pmatrix} r & \lambda & \lambda & \cdots & \lambda \\ \lambda & r & \lambda & \cdots & \lambda \\ \lambda & \lambda & r & \cdots & \lambda \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \lambda & \lambda & \lambda & \cdots & r \end{pmatrix} \tag{2.4}$$

Determinant z rovnice 2.4 můžeme upravit odečtením prvního sloupce od všech ostatních a přičtením všech řádků k prvnímu, tím získáme rovnici 2.5.

$$det(A \cdot A^T) = det \begin{pmatrix} r + \lambda \cdot (v-1) & 0 & 0 & \cdots & 0 \\ \lambda & r - \lambda & 0 & \cdots & 0 \\ \lambda & 0 & r - \lambda & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \lambda & 0 & 0 & \cdots & r - \lambda \end{pmatrix} \tag{2.5}$$

pak

$$det(A \cdot A^T) = [r + \lambda \cdot (v-1)] \cdot (r - \lambda)^{(v-1)} \tag{2.6}$$

Protože $r, v\ a\ \lambda$ jsou kladné a nenulové, tak $r + \lambda \cdot (v-1) > 0$ a od $k < v$ máme $r > \lambda$, tak (rovnice 2.6) nemůže být nikdy rovna 0 $\Rightarrow$ *SPOR*.

2.2.1 Počet řešení

Věta 2.5 (Počet řešení): *Můžeme říct, že libovolná splnitelná instance BIBD má $b!$ řešení.*

Důkaz (Počet řešení): Předpokládejme, že najdeme optimální řešení k nějaké instanci BIBD, pak můžeme libovolně prohazovat sloupce incidenční matice a toto řešení se vzhledem k optimalitě nezmění, protože prohazování celých sloupců neporušuje žádné (řádkové, sloupcové a skalární) omezení řešení. Takových kombinací (sloupců) máme $b!$. Celkový počet řešení, které jsou unikátní, je $b!$.

Někdo by mohl podotknout, že i prohazování řádků nemění vlastnosti řešení a tudíž ke každému prohození sloupců můžeme vytvořit další $v!$ řešení. Počet všech kombinací prohození řádků je taktéž $v!$. Potom by výsledný počet řešení byl $v! \cdot b!$. Tento počet řešení není skutečný, protože se zde vyskytují symetrická řešení (ukázáno na příkladu 2.1).

Příklad 2.1 (Počet řešení): Mějme instanci $\langle 3, 3, 2, 2, 1 \rangle$. Výsledné řešení instance je A.

$$A = \begin{pmatrix} 1\ 1\ 0 \\ 1\ 0\ 1 \\ 0\ 1\ 1 \end{pmatrix} \tag{2.7}$$

prohozením 1. a 2. sloupce matice A získáme řešení A'

$$A' = \begin{pmatrix} 1\ 1\ 0 \\ 0\ 1\ 1 \\ 1\ 0\ 1 \end{pmatrix} \tag{2.8}$$

prohozením 2. a 3. řádku matice A' získame řešení A''

$$A'' = \begin{pmatrix} 1\ 1\ 0 \\ 1\ 0\ 1 \\ 0\ 1\ 1 \end{pmatrix} \tag{2.9}$$

je vidět, že A a A'' jsou ekvivalentní.

2.3 Stavový prostor

Věta 2.6 (Velikost prostoru): *Celý stavový prostor (úplné okolí) instance* $\langle v,\ b,\ r,\ k,\ \lambda\rangle$ *s binární reprezentací matice výskytu je* $2^{v\times b}$.

Důkaz (Počet řešení): Počet prvků v matici výskytu je $v \times b$ a každý prvek může nabývat hodnoty 0 nebo 1, z toho je zřejmé, že počet všech možných kombinací je $2^{v\times b}$ a tedy úplné okolí instance BIBD.

2.4 Symetrický vzor

Případ kdy $b = v$ reprezentuje speciální návrh, který se nazívá *symetrický návrh* $\Rightarrow$ kde matice výskytu má rozměr $v \times v$. Přímým důsledkem z $b = v$ je rovnost $r = k$, aby daný symetrický návrh existoval (2.2). Příkladem symetrického návrhu řešení je mative B v (2.10).

Příklad 2.2 (Příklad BIBD): Pro $\langle v,\ b,\ r,\ k,\ \lambda\rangle$ je zadání:
$A = \langle 8, 14, 7, 4, 3\rangle$ *a* $B = \langle 7, 7, 3, 3, 1\rangle$ (rešení 2.10)

Řešení: Matice výskytu

$$A = \begin{pmatrix} 0&0&0&0&1&0&0&1&1&1&1&1&1&0 \\ 0&1&0&0&1&1&1&0&1&0&0&0&1&1 \\ 0&0&1&1&1&0&1&1&0&1&0&0&0&1 \\ 1&0&1&0&1&1&1&0&0&0&1&1&0&0 \\ 1&0&0&1&0&1&0&0&1&1&1&0&0&1 \\ 0&1&1&1&0&1&0&0&0&1&0&1&1&0 \\ 1&1&0&1&0&0&1&1&0&0&1&0&1&0 \\ 1&1&1&0&0&0&0&1&1&0&0&1&0&1 \end{pmatrix}, \quad B = \begin{pmatrix} 0&1&0&1&0&1&0 \\ 1&0&0&1&0&0&1 \\ 1&1&1&0&0&0&0 \\ 0&0&1&0&0&1&1 \\ 0&1&0&0&1&0&1 \\ 1&0&0&0&1&1&0 \\ 0&0&1&1&1&0&0 \end{pmatrix} \tag{2.10}$$

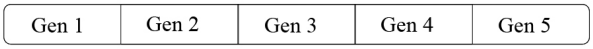
Kapitola 3

Použitý optimalizační algoritmus

Použil jsem algoritmus Optimalizace prototypu pomocí vyvíjení zlepšujících kroků (*Prototype Optimalization with Evolved Improvement Steps* - POEMS), což je globální optimalizační algoritmus řadící se do skupiny evolučních algoritmů (EA) optimalizující v jednom kritériu.

3.1 Obecný popis evolučních algoritmů

Evoluční algoritmy (EA) [4] jsou netradiční výpočetní metody, které napodobují procesy přírodní evoluce (procesy známé z biologie jako dědičnost, mutace, přirozený výběr a křížení dále v části 3.1.3). Přirozená evoluce je proces, při kterém se postupem času z nejjednodušších organismů vyvinuly složitější. Jedinou hybnou silou "vypozorovanou z přírody", která posouvá vývoj dopředu, je schopnost silnějších jedinců přežít v konkurenčním boji o potravu či získání partnera. Silní jedinci poté odevzdávají svoji genetickou informaci dalším generacím, zatím co slabší nepřizpůsobivý jedinci jednoduše vyhynou. Tento evoluční proces je ve své podstatě jednoduchý nicméně efektivní a velmi robustní [11].



Obrázek 3.1: Chromozom

3.1.1 Reprezentace řešení

Reprezentace řešení je inspirováno biologickými chromozomy (řetězce informací). V každé buňce živého organismu je sada chromozomů tvořící model celého organismu. Chromozomy se skládají z genů (obr. 3.1). Každý gen reprezentuje nějakou vlastnost nebo rys např. barva očí. Možné stavy genů (alely) mohou nabývat různých hodnot tj. například již zmíněná barva očí může být modrá, hnědá atd. Každý gen má svou pevnou pozici v chromozomu. Celý genetický materiál organismu se nazývá genom (obr. 3.2). Konkrétní "nastavení" genů v genomu se nazývá genotyp (obr. 3.3). Genotyp určuje tzv. fenotyp, což je vnější projev znaků a vlastností organismů [11].

Barva očí	Výška	Váha	Síla	Inteligence

Obrázek 3.2: Genom

Modrá	180	75	60	125

Obrázek 3.3: Genotyp

Jedním ze způsobů reprezentace řešení jsou číselné řečezce (binarní řetězce, celočíselné řetězce, ...), ale používají se i jiné reprezentace jako stromy, pole, matice, ...

3.1.2 Inicializace počáteční populace

Populaci tvoří množina chromozomů, kde každý chromozom uchovává jedno řešení daného problému (část 3.1.1). Základním parametrem, který udává počet jedinců v populaci, je zde velikost populace. Počáteční populace může být generována náhodně nebo informovaně [12].

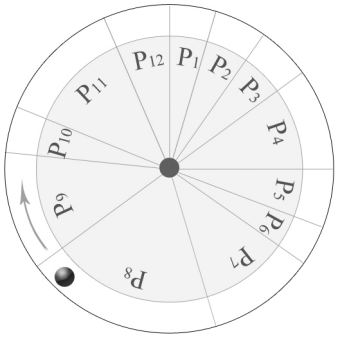
- Náhodná inicializace - je vygenerování celé populace z náhodně vzniklých jedinců (náhodným generátorem). Může dojít k nagenerování "špatných"jedinců a tak k prodloužení celého výpočtu.

- Informovaná inicializace - k vygenerování populace využívá řešení, která jsou nasměrováná pomocí apriorní znalosti o podobě hledaného řešení, tím se může urychlit celý výpočet a může dosahovat lepších výsledků. Na druhou stranu to může vést k nevratnému nasměrování výpočtu k suboptimálnímu řešení.

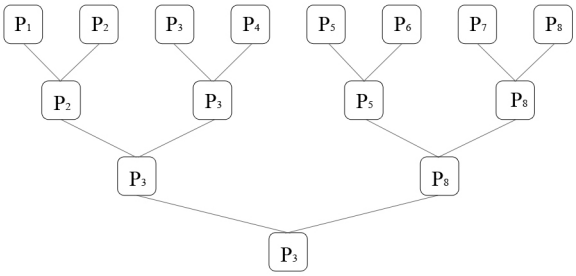
3.1.3 Genetické operátory

Selekce, neboli v přírodě přirozený výběr (přežití nejsilnějších jedinců), můžeme reprezentovat několika způsoby:

- Ruletové kolo - kde pravděpodobnost výběru jedince je úměrná jeho kvalitě obr. 3.4.
- Turnajový výběr - zde se vybírá N náhodných jedinců a provedením turnaje se vybere nejlepší z nich obr. 3.5.



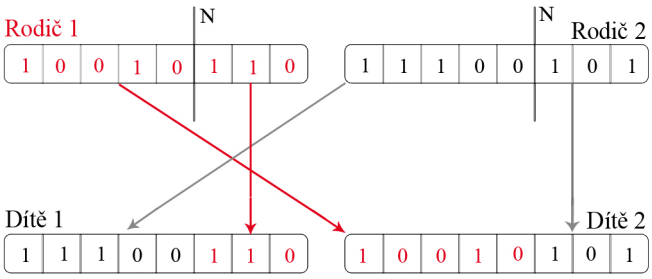
Obrázek 3.4: Ruletový výběr



Obrázek 3.5: Turnajový výběr

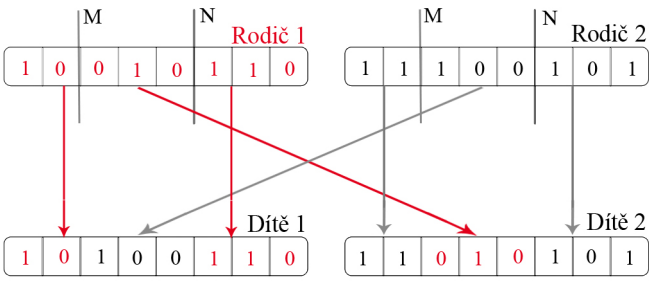
Křížení předpokládá, že každý pár rodičů má v průměru dva potomky. Křížení tedy probíhá mezi dvěma jednici, ze kterých dvojbodovým nebo vícebodovým křížením vznikají jejich dva potomci.

- Dvojbodové - Potomci vznikají z náhodné části genetické informace získané od jednoho rodiče a ze zbývající části, kterou jim předá druhý rodič. Ukázané na obr. 3.6.



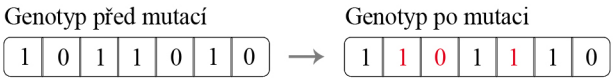
Obrázek 3.6: Dvojbodové křížení

- Vícebodové - Potomci vnikají z více náhodných částí genetické informace získané od jednoho rodiče a zbývající části získají od druhého rodiče. Ukázané na obr. 3.7.



Obrázek 3.7: Vícebodové křížení

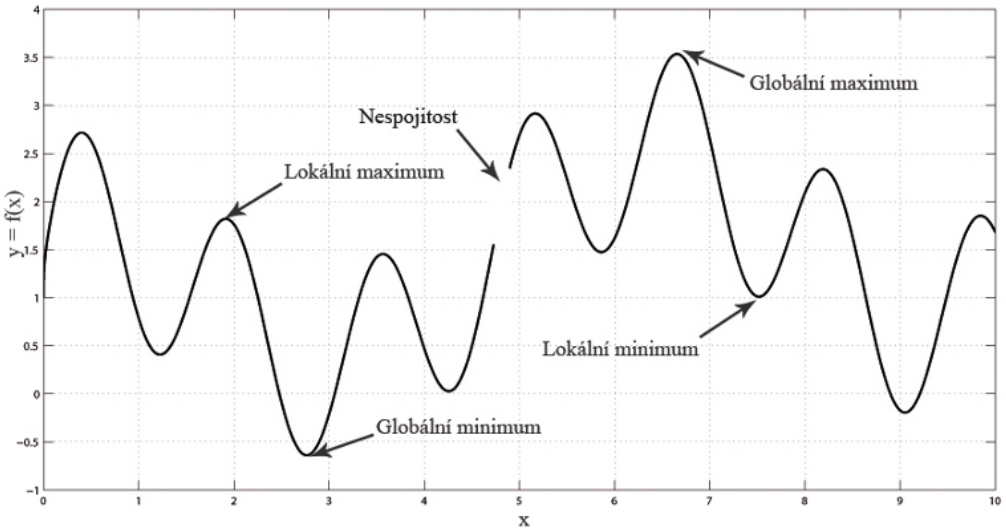
Mutace je náhodná přeměna genetického materiálu v chromozomu (genotypu), znázorněno na obr. 3.8. Mutace přispívá k zachování diverzity populace.



Obrázek 3.8: Mutace

3.1.4 Fitness funkce

Fitness je číselné vyjádření kvality každého jedince a může nabývat různých hodnot. Pro každý problém pak sestavujeme tzv. ohodnocovací (fitness) funkci. Ohodnocovací funkce je jediný identifikátor, který má algoritmus k dispozici, aby směroval prohledávání prostoru do jeho slibných oblastí. Většinou nevíme, jak fitness funkce vypadá. Obecně může být jakákoliv, může mít nespočet lokálních optim či nespojitostí (obr. 3.9), které ovšem EA dokáže překonat.



Obrázek 3.9: Ukázka fitness funkce

3.1.5 Model evolučního algoritmu

Začátkem EA se vygeneruje a ohodnotí počáteční populace (část 3.1.2). Poté začíná evoluce přirozeným výběrem jedinců (část 3.1.3), kteří se budou podílet na vzniku nové generace. Z vybraných jedninců, zvaných rodiče, vzniknou jejich křížením potomci. Následně se někteří jedinci přizpůsobí mutací, která náhodně změní některé jejich vlastnosti. Závěrem jednoho cyklu je ohodnocení nově vzniklých potomků a jejich zařazení do populace.

Tento postup (Algoritmus 1) se iteračně opakuje, čímž se kvality řešení v populaci postupně vylepšují. Algoritmus se obvykle zastaví při dosažení postačující kvality řešení, případně po předem dané době či daném počtu iterací.

Algoritmus 1: Evoluční algoritmus

```
Inicializace: Počáteční nastavení a ohodnocení populace;
while není splněna ukončovací podmínka do
    Selekce: Vyber z populace rodiče;
    Křížení: Vytvoř z rodičů potomky;
    Mutace: Náhodně změň některé rysy některých potomků;
    Ohodnocení: Ohodnoť potomky;
    Nahrazení: Zařaď ponomky do populace;
return Nejlepší z populace;
```

3.2 Obecný popis algoritmu POEMS

Algoritmus POEMS [5] je iterativní stochastický algoritmus, který pracuje s jedním řešením, tzv. *prototypem*, které se snaží v jednotlivých iteracích zlepšovat. Prohledávané okolí je definováno množinou základních akcí, které se aplikují na prototyp. Z těchto akcí se sestavují tzv. sekvence akcí. Okolí definované takovýmito sekvencemi akcí je příliš velké na to, aby se dalo úplně prohledat, proto se k prohledávání okolí používá EA.

3.2.1 Evoluční algoritmus v POEMS

EA v algoritmu POEMS je specifický svým jedincem, uspořádáním populace, ohodnocováním jedinců, přirozeným výběrem a nahrazením jedinců. Jinak ale probíhá stejným způsobem jako EA v části 3.1.

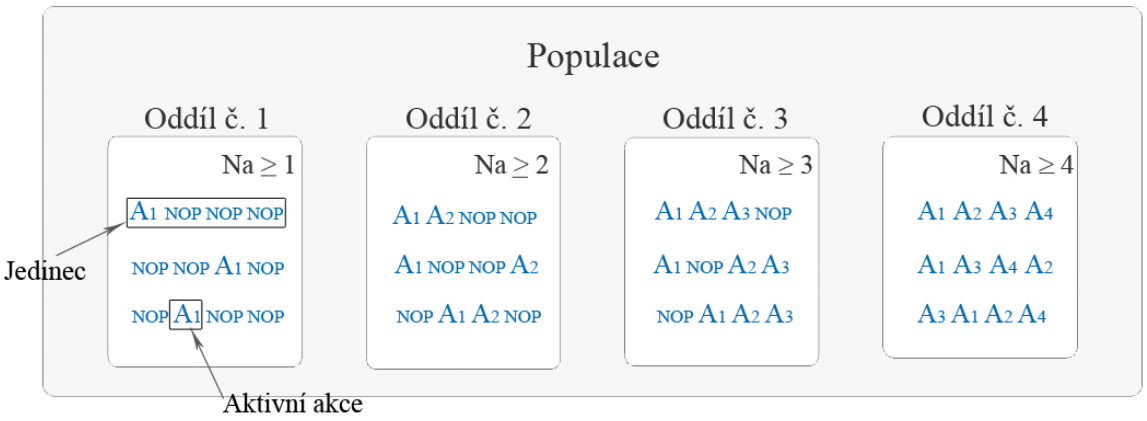
Jedinec

Jedinec nereprezentuje celé řešení, nýbrž jen množinu operací, tzv. sekvenci akcí, která modifikuje aktuální řešení. Sekvence, které jsou genotypem jedince, se skládají z akcí. Velikost genotypu jedince bývá menších rozměrů než velikost problému (řešení) [7]. Genem jedince je tedy akce, což je základní operace, která modifikuje řešení. Podle účinku akcí na řešení je lze rozdělit na dva typy.

- **Aktivní akce** jsou akce, které libovolně modifikují řešení. Druhů těchto aktivních akcí může být obecně více a jejich počet je závislý na řešeném problému.
- **Neaktivní akce**, neboli NOP akce, je opakem aktivní akce, tedy řešení se po její aplikaci nezmění (nemá na řešení žádný vliv).

NOP akce v sekvencích mají speciální úlohu - nedělat nic, podle které se efektivně uskutečňuje proměnná délka sekvencí (délka podle počtu aktivních akcí), což může být výhodou v průběhu výpočtu [6].

Neaktivní akce v sekvenci se může změnit mutací či křížením na aktivní.



Obrázek 3.10: Populace v POEMS

Populace

Populace je tvořena ze sekvencí akcí, které jsou v populaci uspořádány do tzv. oddílů. Jejich počet je dán délkou genotypu a sekvence jsou do nich rozděleny podle počtu svých aktivních akcí (obr. 3.10). Takže v oddílu č.1 jsou jen sekvence s minimálně jednou aktivní akcí, v oddílu č.2 pak sekvence s minimálně dvěmi aktivními akcemi atd.

Fitness

Před ohodnocením se v sekvenci odstraní případné duplicitní akce, které by se mohly vyrušit. Jedna z duplicitních akcí se zanechá v sekvenci a ostatní jsou nahrazeny NOP akcí [7]. Protože sekvence nemá žádnou informaci o kvalitě řešení (jen informaci jak mění *prototyp*) je aplikována na *prototyp*, a poté se tepve zjistí jeho kvalita. Tato kvalita je fitness aplikované sekvece. Fitness sekvence je tedy zjištěna z kvality *prototypu* po aplikování této sekvence, pokud ho sekvence mění. Pokud ne, je sekvence penalizována nastavením nejhorší možné hodnoty.

Selekce

Selekce, neboli přirozený výběr, nevybírá jedince z celé populace, nýbrž jen z oddílu, který se náhodně volí vždy při výběru.

Nahrazení

Nově vzniklý ohodnocený jedinec prochází (po oddílech) populací a porovnává se s ostatními jedinci. Populaci začíná procházet od oddílu, ve kterém je nejméně aktivních akcí (tj. oddíl č.1). Když narazí na jedince, který je horší nebo alespoň stejně dobrý, tak ho v populaci nahradí. Pokud na takového jedince v oddílu nenarazí pokračuje dál za podmínky, že neporušuje omezení dalšího oddílu, jinak se do populace nezařadí.

3.2.2 Model POEMS algoritmu

Algoritmus POEMS, jehož pseudokód je znázorněn v Algoritmus 2, začíná vygenerováním jednoho počátečního řešení (*prototypu*). Zabudovaný EA, který ve svém běhu vyvíjí sekvence akcí, vrátí za n ohodnocení nebo generací nejlépe vyšlechtěnou sekvenci. Ta se poté aplikuje na kopii *prototypu*, kterou nazýváme *kandidát*. Pokud je tento *kandidát* lepší nebo alespoň stejně dobrý jako *prototyp*, zaujme jeho místo, jinak *prototyp* zůstává stejný.

Tento postup se opakuje tak dlouho, dokud není splněna ukončovací podmínka (počet iterací, celkový počet ohodnoceni či postačující kvalita řešení).

Algoritmus 2: Optimalizace prototypu s vyvíjením zlepšujících kroků

```
1 Prototype ⟵ Počáteční nastavení prototypu();
2 i ⟵ 1;
3 while není splněna ukončovací podmínka do
4     NejSekvence ⟵ Spusť EA(Prototype);
5     Kandidát ⟵ PoužijNa(NejSekvence, Prototype);
6     if Kandidát NeníHoršíNež Prototype then
7         Prototype ⟵ Kandidát;
8     i ⟵ i + 1;
9 return Prototype
```

3.3 Popis přizpůsobení POEMS pro řešený problém

Do stávající implementace algoritmu POEMS [10], ve které je oddělena algoritmická část od problémově závislé, jsem vytvořil problémovou část týkající se BIBD. Do této části jsem naimplementoval metody: reprezentace řešení, akce pro modifikaci řešení, výběr akce, ohodnocovací funkci, inicializaci řešení a další potřebné.

3.3.1 Reprezentace řešení

Pro reprezentaci BIBD jsem klasicky zvolil matici výskytu (incidenční matici) $M \equiv \{m_{ij}\}_{v \times b}$ s binární reprezentaci numerických hodnot. Takže $m_{ij} \in \{0, 1\}$ a korektní řešení obsahuje přesně r jedniček na řádek, k jedniček na sloupec a λ jedniček pro vektor vzniklý vynásobením každých dvou rozdílných párů řádků. V m_{ij} je prvek roven 1, když i-tý objekt je obsažen v j-tém bloku a jinak 0.

3.3.2 Základní akce pro modifikace řešení

Implementoval jsem tři struktury okolí tj. jak se řešení mohou měnit. V POEMS se přímo jedná o akce, které nám vylepšují toto řešení (popsáno v části 3.2).

Akce bit-flip. První taková změna okolí neboli akce, vzniká přímo z binární reprezentace řešení tj. matice výskytu M. Tato akce invertuje jeden vybraný prvek z matice výskytu. Vytváří okolí, které je založeno na Hammingově vzdálenosti a označíme jí jako $bit-flip$. Nechť $H(M1, M2)$ je Hammingova vzdálenost mezi rozdílnými maticemi výskytu $M1$ a $M2$. Pak je okolí bit-flip $N_{bit-flip}$ definováno $N_{bit-flip}(M) = \{M' \mid H(M, M') = 1\}$. Zjevně velikost tohoto okolí je $|N_{bit-flip}(M)| = vb$ [2].

V implementaci má akce $bit - flip$ dva parametry (col, row) udávající pozici prvku v matici výskytu, krerý má být změněn. Parametr col udává číslo sloupce a row udává číslo řádku. Ukázka aplikace akce je vidět na obr. 3.11.

$$\begin{array}{c} \\ \text{row}\rightarrow \\ \\ \\ \end{array}\begin{array}{c} \begin{array}{cccccc} & & & \text{col}\downarrow & & \end{array} \\ \begin{pmatrix} 1&0&0&1&0&1 \\ 1&1&0&0&0&0 \\ 0&0&1&0&1&1 \\ 1&0&0&1&1&0 \end{pmatrix} \end{array} \rightarrow \begin{pmatrix} 1&0&0&1&0&1 \\ 1&1&0&1&0&0 \\ 0&0&1&0&1&1 \\ 1&0&0&1&1&0 \end{pmatrix}$$

Obrázek 3.11: Aplikování akce $bit - flip$ s parametry col = 4, row = 2

Akce swap-col. Druhá akce neboli možná struktura okolí se nazývá $swap - col$. Základní myšlenka této akce je prohození libovolného prvku z jednoho bloku s jiným z téhož bloku (prohození dvou prvků v jednom sloupci matice výskytu). Prohození prvků v určitém sloupci nemění sloupcové omezení, ovšem může změnit řádkové omezení a omezení skalárního násobení. Okolí $swap - col$ je definováno jako $N_{swap-col}(M) = \{M' \mid \exists! i, j, k : m_{ij} = m'_{kj} = 0, m'_{ij} = m_{kj} = 1\}$. Velikost tohoto prohledávaného okolí je $|N_{swap-col}(M)| = bk(v-k)$ [2].

V implementaci má akce $swap - col$ tři parametry (col, p_1, p_2) udávající pozici prvků v matici výskytu, kreré mají být prohozeny. Parametr col udává, ve kterém sloupci budou prohozeny prvky p_1 a p_2. Ukázka aplikace akce je vidět na obr. 3.12.

$$\begin{array}{c} \\ p_1\rightarrow \\ \\ p_2\rightarrow \\ \end{array}\begin{array}{c} \begin{array}{cccccc} & & & \text{col}\downarrow & & \end{array} \\ \begin{pmatrix} 1&0&0&1&0&1 \\ 1&1&0&0&0&0 \\ 0&0&1&0&1&1 \\ 1&0&0&1&1&0 \end{pmatrix} \end{array} \rightarrow \begin{pmatrix} 1&0&0&1&1&1 \\ 1&1&0&0&0&0 \\ 0&0&1&0&0&1 \\ 1&0&0&1&1&0 \end{pmatrix}$$

Obrázek 3.12: Aplikování akce $swap - col$ s parametry col = 4, p_1 = 1, p_2 = 3

Akce swap-row. Třetí nadefinovaná struktura okolí (akce) je v podstatě stejná jako předchozí akce $swap - col$ s tím rozdílem, že se zde nejedná o prohazování ve sloupci, ale v řádku. Nazývá se $swap - row$, kde dochází k prohození libovolného prvku z jednoho objektu za

jiný prvek ze stejného objektu (prohození dvou prvků v jednom řádku matice výskytu). Operace nemění řádkové omezení, ale ostatní se mohou změnit. Okolí $swap - row$ je definována jako $N_{swap-row}(M) = \{M' \mid \exists! i, j, k : m_{ij} = m'_{ik} = 0, m'_{ij} = m_{ik} = 1\}$. Velikost prohledávaného okolí je $|N_{swap-row}(M)| = vr(b-r)$ [2].

V implementaci má akce $swap - row$ tři parametry (row, p_1, p_2) udávající pozici prvků v matici výskytu, kreré mají být prohozeny. Parametr row udává, ve kterém řádku budou prohozeny prvky p_1 a p_2. Ukázka aplikace akce je vidět na obr. 3.13.

$$\begin{array}{c} \\ \\ \text{row}\rightarrow \\ \\ \end{array}\begin{array}{c} \begin{array}{cccccc} & \mathbf{p_1} & & & \mathbf{p_2} & \\ & \downarrow & & & \downarrow & \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \end{array} \rightarrow \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Obrázek 3.13: Aplikování akce $swap - row$ s parametry row = 4, p_1 = 1, p_2 = 5

3.3.3 Ohodnocovací funkce

Pro řešení problému BIBD pomocí POEMS algoritmu jsem musel stanovit objektivní neboli fitness funkci. Fitness jsem tedy definoval jako stupeň porušení omezení, které udávají parametry $v,\ b,\ r,\ k,\ \lambda$ ze zadání. Pro obecný případ instance $\langle v,\ b,\ r,\ k,\ \lambda\rangle$ je stupeň porušení omezení definován rovnicí 3.1.

$$f^{\langle v,\ b,\ r,\ k,\ \lambda\rangle}(M) = \sum_{i=1}^{v} \phi_{ir}(M) + \sum_{j=1}^{b} \phi'_{jk}(M) + \sum_{i=1}^{v-1} \sum_{j=i+1}^{v} \phi''_{ij\lambda}(M) \tag{3.1}$$

kde

$$\phi_{ir}(M) = |r - \sum_{j=1}^{b} m_{ij}|,\ \phi'_{ik}(M) = |k - \sum_{i=1}^{v} m_{ij}|,\ \phi''_{ij\lambda}(M) = |\lambda - \sum_{k=1}^{b} m_{ik} m_{jk}|.$$

Všimněme si, že pro danou matici výskytu M, je návratová hodnota objektivní (fitness) funkce součtem všech nesrovnalostí od optimálního řešení. Pro dosažení optimálního (správného) řešení je zřejmé, že se problém musí minimalizovat.

Pokud se pro nastavení M* našla taková návratová hodnota z fitness funkce, která je rovna nule $f^{\langle v,\, b,\, r,\, k,\, \lambda\rangle}(M^*) = 0$, potom se jedná o globální optimum. Je zřejmé, že takovýchto globálních optim může být více (věta 2.5).

3.3.4 Inicializace počátečního řešení

Uvažoval jsem tři typy nastavení počátečního řešení. Jednu triviální náhodnou inicializaci a poté v rámci zlepšení efektivnosti dvě nasměrované inicializace.

Náhodná inicializace. Vygenerování celé $v \times b$ incidenční matice náhodně s pravděpodobností 50% (0×1) pro každou pozici v matici výskytu.

Řízená po řádcích. Podle parametrů $\langle v,\ b,\ r,\ k,\ \lambda\rangle$ víme, že každý řádek incidenční matice obsahuje přesně r prvků, proto jsem do každého řádku matice (na náhodnnou pozici) vygeneroval r prvků.

Řízená po sloupcích. Také víme, že každý sloupec incidenční matice obsahuje přesně k prvků, proto jsem do každého sloupce matice (na náhodnou pozici) vygeneroval k prvků.

3.3.5 Výběr akce

Existuje řada možnotí, jak akce vybírat. Výběr akcí je spojen s určením počátečního nastavení řešení. Zvolením některé z řízených inicializací, kde je nasměrované řešení, je již splněno potřebné řádkové nebo sloupcové omezení (je nevhodné je měnit $\Rightarrow$ použití $swap - row$ nebo $swap - col$). Všechny možnosti pro výběr akcí musí splňovat rovnici $P_{swap-row} + P_{swap-col} + P_{bit-flip} = 100\%$.

- Varianta vybrání jedné jediné akce pro celý běh algoritmu POEMS. Vybraná akce má po celou dobu běhu 100% pravděpodobnost a je v souladu s počátečním nastavením řešení. Pokud inicializace po řádcích tak akce $swap - row$, pokud po sloupcích tak $swap - col$.

- Varianta vybrání jedné jediné akce pro celý běh EA. Vždy na začátku EA, před vygenerováním populace, se s pravděpodobností 50% vybere akce $swap - row$ nebo $swap - col$, která zůstává platná po celý běh EA. Pro inicializaci po řádcích nebo po sloupcích.

- Varianta vybrání více akcí pro celý běh EA. Na začátku EA se s 50% pravděpodobností vybere kobinace akcí $swap - row$ + $bit - flip$ nebo $swap - col$ + $bit - flip$, a to pro náhodné počáteční nastavení řešení.

3.3.6 Modifikace algoritmu

Pro vylepšení stávajícího algoritmu POEMS jsem zavedl tzv. frontu (frontu o délce N). Do této fronty je vygenerováno N řešení, která se mohou pohybovat odlišnými směry.

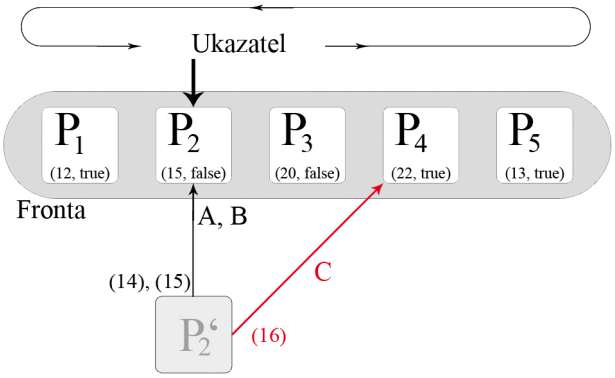
V počátečním nastavení se vygeneruje N *prototypů* místo jednoho, které zaplní frontu. Protože pro vývoj sekvencí akcí v EA potřebuje jen jeden *prototyp*, máme ukazetel, který ukazuje na jeden prvek z fronty (na aktuální prvek). Ke každému prvku z fronty navíc přiřadíme speciální příznak, který určuje, jestli pro něj byl spuštěn evoluční algoritmus (zda se může vylepšit). Na začátku má tedy každý prvek z fornty nastavený příznak na $false$.

Na začátku je ukazatel logicky na prvním prvku fronty. Pro aktuální prvek P_u (prvek s ukazatelem) je spuštěn EA, který mu vyvine sekvenci akcí. Sekvence se poté aplikuje (použije) na kopii *prototypu* $\Rightarrow$ *kandidát,* P'_u. Zde mohou nastat tři varianty jako na obr. 3.14.

(A) P'_u je lepší než P_u

(B) P'_u je stejně kvalitní jako P_u

(C) P'_u je horší než P_u



Obrázek 3.14: Fronta

Add. A Pokud EA vyvine takovou sekvenci, která vylepšuje aktuální prvek fronty tak je nahrazen nově vzniklým lepším prvkem. $P_u \longleftarrow P'_u$. Tato operace nastavuje příznak na $false$ a nemění pozici ukazatele.

Add. B Pokud z EA vyjde sekvence, která změní aktuální prvek tak, že nově vzniklý prvek je stejně kvalitní jako předchozí. Provede se $P_u \longleftarrow P'_u$ operace, která nastavuje příznak na $false$ a nemění pozici ukazatele.

Add. C Pokud je nově vzniklý prvek horší než aktuální, dochází u aktuálního prvku k nastavení příznaku na $true$ a nedochází k přepsání novým. Pokud ale existuje prvek P_j, který je z prvků s příznakem $true$ nejhorší a je také horší než P'_u tak $P_j \longleftarrow P'_u$. U P_j se nastaví příznak na $false$ a nakonce se inkrementuje ukazatel podle $u = (u + 1)\ mod\ N$.

Popsaná fronta je znázorněná v algoritmu 3.

Algoritmus 3: POEMS s frontou

```
1  P ⟵ Počáteční nastavení fronty();
2  i ⟵ 1;
3  u ⟵ 1;
4  while není splněna ukončovací podmínka do
5      NejSekvence ⟵ Spusť EA(P_u);
6      P'_u ⟵ PoužijNa(NejSekvence, P_u);
7      if P'_u NeníHoršíNež P_u then
8          P_u ⟵ P'_u;
9          P_u nastav příznak 0;
       else
10         P_u nastav příznak 1;
11         index ⟵ dej index nejhoršího z P, jehož příznak je 1;
12         if P'_u NeníHoršíNež P_index then
13             P_index ⟵ P'_u;
14             P_index nastav příznak 0;
15         u ⟵ 1 + (u mod N);
16     i ⟵ i + 1;
17 return nejlepší z P
```

Kapitola 4

Experimenty

Po dokončení všech úprav jsem provedl experimenty, aby jsem si ověřil výsledky a kvalitu našeho řešení.

4.1 Popis porovnávaných algoritmů

Ve studii, ze které jsem vycházel [2] tento problém řešili za pomocí tří algoritmů a to s *lokálním prohledáváním (hill climbing), genetickým algoritmem (genetic algoritm) a tabu prohledávání (taboo search)* popsané níže. Ve všech algoritmech uvažovali dvě struktury okolí (u nás akce). A to změna bitu tedy $bit - flip$ a prohození v řádku $swap - row$. Tyto struktury okolí nijak vzájemně nekombinovali. Zvolili buď jen $bit - flip$ nebo $swap - row$. Tyto struktury okolí jsou popsány v části 3.3.2. Všechny algoritmy byly spuštěny 30-krát pro každou instanci problému. Jeden běh algoritmu byl omezen na $2 \cdot 10^6$ ohodnocení. Výpis porovnávaných problémů je v části 4.2.

4.1.1 Lokální prohledávání HC

První algoritmus, který byl použit v článku [3] je metoda největšího stoupání (anglicky hill climbing). Tato metoda svůj běh začíná v náhodně zvoleném bodě a přechází vždy do nejlepšího stavu ve svém okolí. Uváznutí algoritmu v lokálním optimu se řeší restartováním algoritmu z nové pozice. Pokud se algoritmus dostane do stejného místa (nalezne stejné řešení) je možné, že se jedná o globální optimum. Pokud najde lepší řešení, tak se jednalo o lokální optimum. V případě, že najde horší řešení, tak je opět pravděpodobné, že se jednalo o globální optimum. Popis algoritmu je videt na Algorithm 4.

Algoritmus 4: Lokální prohledávání - strategie prvního zlepšení

```
1 x ⟵ Počáteční nastavení();
2 while není splněna ukončovací podmínka do
3     y ⟵ Nejlepší_z_okolí(x);
4     if Lepší než(y, x) then
5         x ⟵ y;
6 return x;
```

4.1.2 Lokální prohledávání TS

Druhým porovnávaným algoritmem je tabu prohledávání (anglicky taboo search). Jedná se taktéž o lokální prohledávání, ve kterém se zabraňuje krátkému cyklení mezi stavy pomocí tabu seznamu. Tabu seznam je seznam zakázaných stavů (stavy ve kterých algoritmus už byl). Pseudokód tabu prohledávání Algorithm 5.

Algoritmus 5: Tabu prohledávání

```
1 x ⟵ Počáteční nastavení();
2 y ⟵ x;
3 M ⟵ [ ];
4 while není splněna ukončovací podmínka do
5     y ⟵ Nejlepší_z_okolí(N(y, d) − M);
6     M ⟵ AktualizacePaměti(M, y);
7     if LepšíNež(y, x) then
8         x ⟵ y;
9 return x;
```

4.1.3 Genetický algoritmus

Posledním porovnávaným algoritmem je genetický algoritmus. Jehož popis je popsán výše v části 3.1. Parametry EA zvolili následující:

- Velikost populace: 100
- Pravděpodobnost křížení: $p_C = 0.9$
- Pravděpodobnost mutace: $p_M = 1 / l$, kde l je proměnná konstanta závislá na velikosti instance problému, tedy $l = v \cdot b$
- Velikost turnaje: 10% velikosti populace tj. 10

4.2 Výčet vybraných testovacích problémů

K otestování našeho algoritmu jsem vybral 15 instancí BIBD problému opět ze studie [2] uvedené v tabulce 4.1. Všechny vybrané instance vyhověly třem nutným podmínkám pro existenci optimálního řešení.

Tabulka 4.1: Tabulka vybraných BIBD problémů

ID	$\langle v, b, r, k, \lambda \rangle$	ID	$\langle v, b, r, k, \lambda \rangle$	ID	$\langle v, b, r, k, \lambda \rangle$
1.	$\langle 8, 14, 7, 4, 3 \rangle$	6.	$\langle 11, 55, 20, 4, 6 \rangle$	11.	$\langle 12, 66, 33, 6, 15 \rangle$
2.	$\langle 8, 42, 21, 4, 9 \rangle$	7.	$\langle 18, 34, 17, 9, 8 \rangle$	12.	$\langle 21, 42, 10, 5, 2 \rangle$
3.	$\langle 13, 26, 8, 4, 2 \rangle$	8.	$\langle 9, 72, 32, 4, 12 \rangle$	13.	$\langle 10, 90, 36, 4, 12 \rangle$
4.	$\langle 14, 26, 13, 7, 6 \rangle$	9.	$\langle 15, 45, 21, 7, 9 \rangle$	14.	$\langle 31, 31, 6, 6, 1 \rangle$
5.	$\langle 13, 39, 15, 5, 5 \rangle$	10.	$\langle 13, 52, 24, 6, 10 \rangle$	15.	$\langle 25, 40, 16, 10, 6 \rangle$

4.2.1 Jak tyto problémy řešily porovnávané algoritmy

Jak již bylo zmíněno výše, všechny porovnávané algoritmy byly řešeny se strukturou okolí $bit - flip$ nebo $swap - row$. U všech porovnávaných algoritmů měly výrazně lepší výsledky

Tabulka 4.2: Tabulka počtu uspěšného nalezení řešení

Algoritmus	*Nalezené instance / Všechny instance*
$HC_{bitflip}$	1 / 15
$HC_{swaprow}$	6 / 15
$GA_{bitflip}$	5 / 15
$GA_{swaprow}$	5 / 15
$TS_{bitflip}$	4 / 15
$TS_{swaprow}$	10 / 15

se strukturou okolí $swap - row$ než se strukturou $bit - flip$. Také jeden z použitých algoritmů byl výrazně lepší než oba zbývající. Nejlépe tento problém vyřešilo Tabu prohledávání, které našlo řešení v 10-ti z 15-ti instancí. Shrnuté výsledky porovnávaných algoritmů jsou vidět v tabulce 4.2 a podrobné výsledky všech jsou uvedeny v tabulkách 4.3, 4.4 a 4.5.

Tabulka 4.3: Tabulka výsledků lokálního prohledávání

	$HC_{bit-flip}$			$HC_{swap-row}$		
ID	*Průměr*	*Nejlepší Fitness*	*Počet řešení*	*Průměr*	*Nejlepší Fitness*	*Počet řešení*
1.	3.5	0	2	0.0	0	30
2.	12.1	8	0	0.0	0	30
3.	16.2	12	0	6.1	4	0
4.	38.8	33	0	13.4	6	0
5.	27.0	20	0	8.6	5	0
6.	19.2	14	0	4.1	0	3
7.	85.6	79	0	34.9	28	0
8.	25.9	13	0	2.0	0	15
9.	54.9	41	0	17.6	13	0
10.	42.2	36	0	10.0	4	0
11.	51.7	43	0	7.4	0	1
12.	64.9	56	0	36.9	29	0
13.	28.0	14	0	5.7	0	3
14.	134.7	122	0	100.8	87	0
15.	167.6	150	0	98.6	88	0

Tabulka 4.4: Tabulka výsledků tabu prohledávání

	$HC_{bit-flip}$			$HC_{swap-row}$		
ID	*Průměr*	*Nejlepší Fitness*	*Počet řešení*	*Průměr*	*Nejlepší Fitness*	*Počet řešení*
1.	3.3	0	10	0.0	0	30
2.	1.6	0	19	0.0	0	30
3.	10.6	4	0	0.0	0	30
4.	22.7	15	0	5.8	4	0
5.	12.9	9	0	0.3	0	28
6.	5.5	4	0	0.0	0	30
7.	48.8	39	0	20.5	16	0
8.	1.5	0	19	0.0	0	30
9.	23.4	16	0	6.3	4	0
10.	14.4	10	0	0.4	0	27
11.	12.6	5	0	0.0	0	30
12.	40.5	32	0	18.2	14	0
13.	2.2	0	17	0.0	0	30
14.	109.2	89	0	22.9	0	12
15.	107.7	99	0	65.7	56	0

Tabulka 4.5: Tabulka výsledků genetického algoritmu

	$HC_{bit-flip}$			$HC_{swap-row}$		
ID	*Průměr*	*Nejlepší Fitness*	*Počet řešení*	*Průměr*	*Nejlepší Fitness*	*Počet řešení*
1.	0.0	0	30	0.0	0	30
2.	0.7	8	24	0.0	0	30
3.	6.5	4	0	5.6	4	0
4.	17.0	11	0	10.0	6	0
5.	10.0	4	0	6.7	4	0
6.	4.0	0	5	2.3	0	13
7.	47.2	38	0	28.9	24	0
8.	2.0	0	17	4.0	0	5
9.	25.7	17	0	18.6	13	0
10.	14.9	9	0	12.0	7	0
11.	12.4	7	0	10.6	6	0
12.	39.2	32	0	36.2	25	0
13.	5.2	0	1	7.2	0	2
14.	107.4	80	0	97.9	78	0
15.	125.2	107	0	94.9	81	0

4.3 Konfigurace našeho POEMS

Pro průběh experimentů jsme zvolili následující parametry:

- Velikost populace: 100

  velikost populace je závislá na:

  - Velikost oddílu: 20
  - Počet genů v sekvenci: 5

    $5 \cdot 20 = 100$ *viz. 3.2*

- Počet ohodnocení v jednom běhu EV: 1000
- Pravděpodobnost křížení: 75%
- Pravděpodobnost mutace: 25%
- Pravděpodobnost změny bitu při mutaci: 50%
- Počet jedinců k turnaji: 3
- Pravděpodobnost výběru libovolné akce: 50% *viz. 4.4 a 3.3.5.*
- Maximální počet ohodnocení: 2000000
- Strategie fitness: minimalizace
- Počet restartů: 30

  za resatart je považováno spuštění algoritmu od začátku ovšem z jiným počátečním nastavením (jiným seedem) v případě, že předchozí běh nalezl řešení nebo mu vypršel maximální počet ohodnocení.

- Seed: prvních 30 prvočísel

  Seed je počáteční nastavení pseudo-náhodného generátoru čísel.

  Seznam: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113

- Počáteční nastavení řešení: řízené $\times$ náhodné.

4.4 Vlastní experimenty

Pro experimenty jsem z řady kombinací nastavení vybral několik konfigurací. Konfigurace POEMS byla nastavena viz. 4.3 a k tomu následující výběry akcí a inicializace počátečního řešení.

1. Výběr jedné $swap$ akce pro celý běh POEMS a k tomu příslušná řízená inicializace.
2. Řízená inicializace a výběr akce $swap - row$ nebo $swap - col$ pro každý běh EA.
3. Náhodná inicializace a výběr akcí $swap - row\ a\ bit - flip$ nebo $swap - col\ a\ bit - flip$ pro každý běh EA. Pravděpodobnost pro $swapy$ 80%, pro $bit - flip$ 20%.

Důvodem užití fronty je postup prvků ve frontě odlišným směrem a tedy větší pravděpodobnost dosažení cíle.

Add 1. Myšlenkou je, že u řízené inicializace po řádcích nebo po sloupcích je již splněno jedno ze tří omezení. Použitím příslušné $swap$ akce se neporušují řádková nebo sloupcová omezení a mohou prohazováním prvků v bloku či objektu vylepšit ostatní omezení. V tabulce 4.8 jsou uvedeny výsledky této konfigurace bez fronty a s frontou délky 5.

Tabulka 4.6: Tabulka vlastních výsledků - řízená inic. a jedna akce v POEMS

	$POEMS$			$POEMS - fronta\ 5$		
ID	*Průměr*	*Nejlepší Fitness*	*Počet řešení*	*Průměr*	*Nejlepší Fitness*	*Počet řešení*
1.	0.0	0	30	0.0	0	30
2.	0.0	0	30	0.0	0	30
3.	4.7	0	2	4.5	0	2
4.	10.9	8	0	9.7	6	0
5.	6.6	4	0	5.2	4	0
6.	3.3	0	6	1.3	0	20
7.	27.7	21	0	24.8	20	0
8.	0.3	0	28	0.0	0	30
9.	12.6	9	0	10.9	7	0
10.	7.0	4	0	5.3	0	2
11.	5.1	4	0	3.8	0	4
12.	27.3	21	0	26.2	22	0
13.	2.3	0	13	0.5	0	26
14.	73.4	0	1	62.2	0	1
15.	76.7	65	0	76.6	69	0

Add 2. Výběr vždy jedné ze $swap$ akcí pro jeden běh EA přináší možnost vymanění se z lokálního optima. Kdy s akcí, které jdou proti stávajícímu splněnému omezení, se může vyvinout taková sekvence akcí (stejně kvalitní jako aktuální řešení), která sice porušuje již splněné omezení, ale poté se z toho nového řešení může najít lepší. U fronty větší délky než jedna může dojít i k zhoršení řešení, které potom vede k lepším výsledkům. Výsledky pro algoritmy bez fronty a s frontou délky 5 jsou v tabulce 4.7.

Tabulka 4.7: Tabulka vlastních výsledků - řízená inicializace a jedna akce v EA

	$POEMS$			$POEMS - fronta\ 5$		
ID	*Průměr*	*Nejlepší Fitness*	*Počet řešení*	*Průměr*	*Nejlepší Fitness*	*Počet řešení*
1.	0.0	0	30	0.0	0	30
2.	0.0	0	30	0.0	0	30
3.	4.3	0	3	3.8	0	4
4.	11.0	6	0	9.6	0	1
5.	6.1	0	1	4.7	0	2
6.	0.8	0	24	0.5	0	26
7.	28.8	23	0	27.1	23	0
8.	0.3	0	29	0.0	0	30
9.	14.1	9	0	12.3	8	0
10.	5.7	0	1	6.5	4	0
11.	4.0	0	4	4.2	0	3
12.	27.2	19	0	29.0	21	0
13.	0.5	0	26	1.0	0	21
14.	72.1	50	0	76.6	39	0
15.	78.0	71	0	85.0	75	0

Add 3. Hlavním důvodem pro zvolení náhodné inicializace je přirozený průběh výpočtu. U řízené inicializace se totiž může stát, že se řešení nasměruje špatným směrem a nedojde k nalezení optimálního řešení. Zde jsem musel použít všechny tři akce a to právě kvůli náhodné inicializaci, protože $swapy$ neumí aktivovat nebo deaktivovat prvek (akce $bit - flip$ ano). Např. náhodnou inicializací instance $\langle 8, 14, 7, 4, 3\rangle$ se v 8×14 matici výskytu objeví 20 jedniček, ale tato instance pro splnění všech omezení potřebuje $8 \cdot 4$ to je 32 jedniček. Tohoto počtu se jen prohazováním řádků nebo sloupců nedosáhne, proto akce $bit - flip$. Výsledky této konfigurace jsou uvedeny v tabulce 4.8.

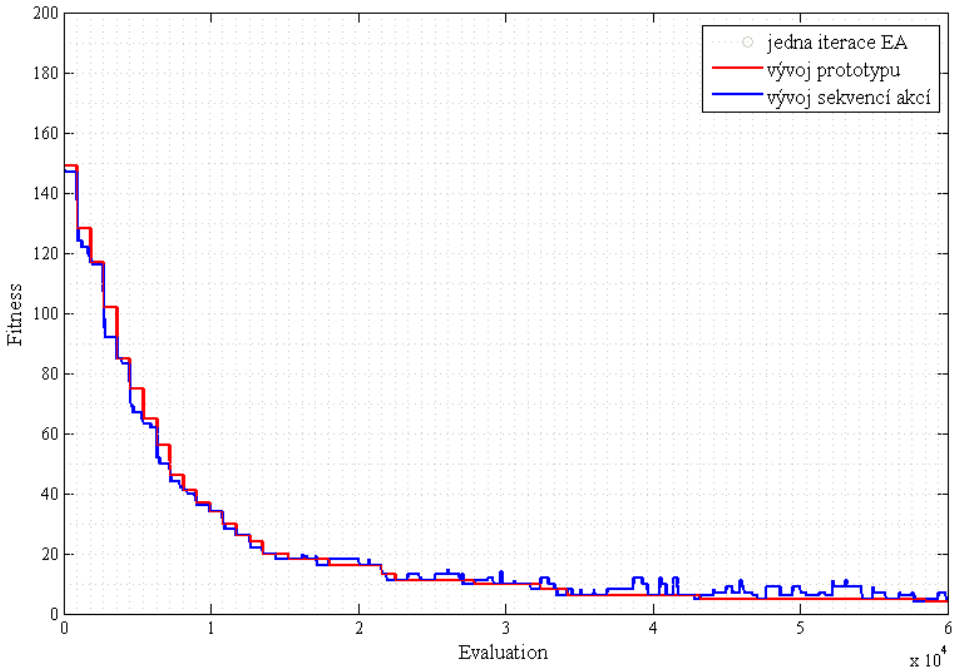
Tabulka 4.8: Tabulka vlastních výsledků - náhodná inicializace a dvě akce v EA

	$POEMS$			$POEMS - fronta\,5$		
ID	*Průměr*	*Nejlepší Fitness*	*Počet řešení*	*Průměr*	*Nejlepší Fitness*	*Počet řešení*
1.	0.0	0	30	0.0	0	30
2.	0.0	0	30	0.0	0	30
3.	4.2	0	5	4.9	0	1
4.	40.5	39	0	9.8	6	0
5.	6.8	4	0	5.2	4	0
6.	2.5	0	12	1.0	0	22
7.	29.0	21	0	28.0	23	0
8.	0.1	0	29	0.0	0	30
9.	13.8	11	0	13.0	8	0
10.	7.4	4	0	6.4	4	0
11.	5.3	4	0	4.1	0	4
12.	28.4	21	0	27.9	21	0
13.	0.9	0	23	0.1	0	29
14.	78.0	15	0	81.1	64	0
15.	82.8	72	0	83.0	77	0

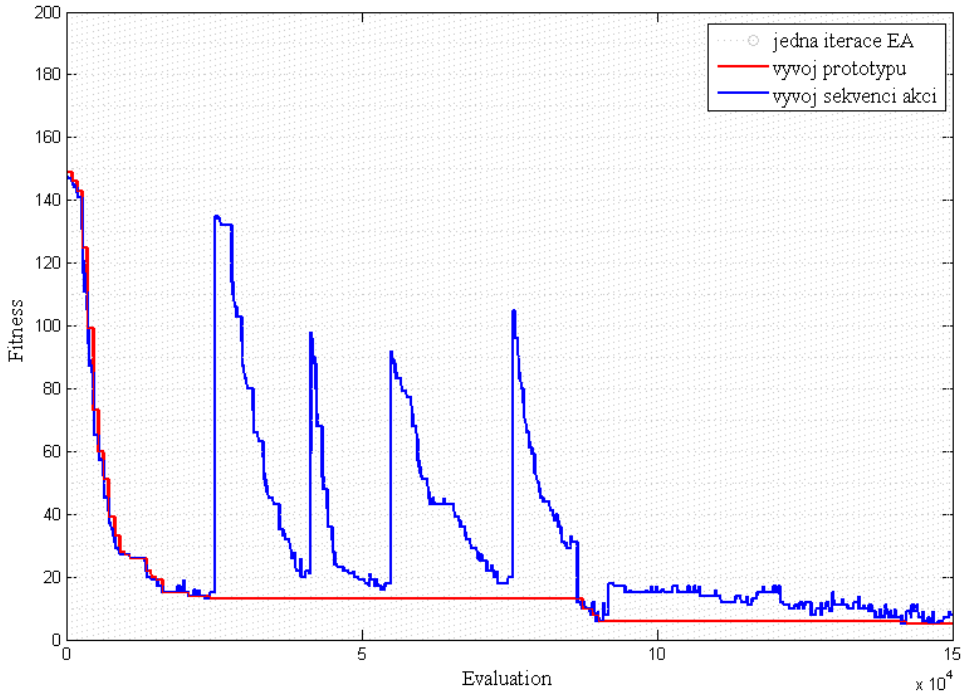
Shrnuté výsledky všech nastavení algoritmů jsou v tabulce 4.9. Dále můžeme vidět v rozdílné průběhy algoritmů s frontou a bez fronty. V grafu na obr. 4.1 je průběh bez použití fronty a graf na obr. 4.2 je s frontou o délce 5. Z grafu je vidět, že algoritmus s frontou má pomalejší konvergenci než bez fronty. To je logické, protože neobsluhuje jen jedno řešení ale postupně pět řešení. Hlavní výhodou fronty je to, že dané řešení se v ní může zhoršit a později zase vylepšit a tím překonat překážku, kterou by samotný algoritmu nepřekonal, protože sekvence akcí nevytváří dostatečně velké okolí. Ze zhoršeného jedince, který je tzv. blíže k horizontu, je "vidět" dál až za horizont a možnost přesunu za něj, kde může nalézt lepší řešení.

Tabulka 4.9: Shrnutí všech vlastních výsledků

Algoritmus POEMS	*Nalezené instance / Všechny instance*
swap v POEMS bez fronty	7 / 15
swap v POEMS fronta 5	9 / 15
swap v EA bez fronty	9 / 15
swap v EA fronta 5	9 / 15
swap & bitflip v EA bez fronty	6 / 15
swap & bitflip v EA fronta 5	7 / 15



Obrázek 4.1: Vývoj řešení algoritmu POEMS



Obrázek 4.2: Vývoj řešení algoritmu POEMS s frontou o velikosti 5

Kapitola 5

Závěr

Cílem této práce bylo seznámení s jednokriteriální variantou algoritmu POEMS a jeho implementování pro řeření problému BIBD. V této implementaci bylo důležité dodržet oddělení algoritmické části od problémově závislé. Dále pro implementaci bylo třeba vymyslet akce modifikující řešení, určit výběr těchto akcí, zvolit reprezentaci řešení, fitness funkci a další.

Úspěšnost algoritmu jsem posoudil na základě experimentálních výsledků, které jsem porovnal s dalšími třemi algoritmy - lokální prohledávání, tabu prohledávání a genetický algoritmus. K testování problému jsem vybral tři různé výběry akcí - $swap$ v celém POEMS, $swap$ v EA a $swap + bit - flip$ v EA, z toho měly první dvě nastavenou řízenou inicializaci řešení a poslední náhodnou. U řízené inicializace se očekávaly lepší výsledky, nicméně mohlo u nich dojít k nevratnému nasměrování výpočtu špatným směrem, proto jsem problém otestoval i s náhodnou inicializací řešení. U náhodné inicializace jsem k $swap$ akcím musel přidat akci $bit - flip$ z důvodu správného počtu prvků v matici výskytu. Výsledky nebyly tak dobré jak jsem očekával, nicméně byly dostačující.

Inspirován algoritmem Tabu prohledávání jsem zavedl rozšíření algoritmu POEMS, které používá frontu kandidátů na prototyp. Použití fronty umožňuje navrátit se při prohledávání k některému z předchozích řešení. Umožnuje také přijmout řešení, které je horši než modifikovaný prototyp a takto uniknout z lokalního optima. Důležité bylo zvolit správnou velikost fronty - krátká fronta by znamenala menší pravděpodobnost vyváznutí z lokálního optima a dlouhá fronta potom pomalou konvergenci a tedy nenalezení řešení v daném limitu.

Frontu jsem testoval pro stejná tři nastavení výběru akcí z důvodu jejich porovnání. Od této fronty jsem očekával výrazné zlepšení, ke kterému nedošlo, zlepšení bylo jen malé.

Naimplementováný algoritmus POEMS pro řešení BIBD problému sice funguje dobře, je lepší než lokální prohledávací algoritmy a genetický algoritmus, nicméně Tabu prohledávání dává lepší výsledky.

Literatura

[1] Fisher, R. A.: *The design of experiments*. British Medical Journal. 1936.

[2] Rueda, D. R.;, Cotta, C.; Fernández, A. J.: *Finding Balanced Incomplete Block Designs with Metaheuristics*. Proceedings of the 9th European Conference on Evolutionary Computation in Combinatorial Optimization. Springer-Verlag Berlin Heidelberg. 2009.

[3] Cherowitzo, B.: *Combinatorial Desings: Balanced Incomplete Block Designs*. Dostupný z WWW: http://www-math.cudenver.edu/~wcherowi/courses/m6406/cslnb.html. 2006.

[4] Goldberg, D. E.: *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley. 1989.

[5] Kubalík, J.: *Solving the Sorting Network Problem Using Iterative Optimization with Evolved Hypermutations*. Proceedings of the 11th Annual conference on Genetic and evolutionary computation. ACM - New York. 2009.

[6] Kubalík, J.: *Solving the Multiple Sequence Alignment Problem Using Prototype Optimization with Evolved Improvement Steps*.Springer-Verlag Berlin Heidelberg. 2009.

[7] Kubalík, J.; Faigl, J.: *Iterative Prototype Optimisation with Evolved Improvement Steps*. Springer-Verlag Berlin Heidelberg. 2006.

[8] Pošík, P.: *Optimalizace černé skříňky*. Přednáška - Aplikace umělé inteligence. 2009.

[9] Olšák, P.: *Úvod do algebry, zejména lieární*. Fakulta elektrotechnická ČVUT v Praze. 2007.

[10] Sálus, J.: *Implementace algoritmu POEMS v Javě*. Semestrální projekt. 2009.

[11] Sekaj, I.: *Evoluce v počítači aneb evoluční a genetické algoritmy*. Dostupný z WWW: http://www.posterus.sk/?p=3364. 2009.

[12] Kubalík, J.: *Evoluční výpočetní techniky*. Přednáška - Základy umělé inteligence. 2009.

[13] Šůcha, P.: *Složitost algoritmů*. Přednáška - Optimalizace v inteligentních systémech. 2010.

Příloha A

Obsah přiloženého CD

K této práci je přiloženo CD:

- Řešení problému BIBD pomocí algoritmu POEMS.pdf: *elektronická verze práce*
- Články:
  - Combinatorial Desings Balanced Incomplete Block Designs.pdf
  - Finding Balanced Incomplete Block Designs with Metaheuristics.pdf
  - Implementace algoritmu POEMS v Javě.pdf
  - Iterative Prototype Optimisation with Evolved Improvement Steps.pdf
  - Přednáška - AUI - Optimalizace černé skříňky.pdf
  - Přednáška - OIS - Složitost algoritmů.pdf
  - Přednáška - ZUI - Evoluční výpočetní techniky.pdf
  - Solving the Multiple Sequence Alignment Problem Using Prototype Optimization with Evolved Improvement Steps.pdf
  - Solving the Sorting Network Problem Using Iterative Optimization with Evolved Hypermutations.pdf
- Experimenty:
  - bez fronty a jednou $swap$ akcí v celém POEMS

  - fronta velikos 5 a jednou $swap$ akcí v celém POEMS
  - bez fronty a výběr $swap$ akce pro EA
  - fronta velikos 5 a výběr $swap$ akce pro EA
  - bez fronty a výběr $swap$ akce $+ bit - flip$ akce v EA
  - fronta velikos 5 a výběr $swap$ akce $+ bit - flip$ akce v EA

- Zdrojové kódy:
  - POEMS_sourcecode: zdrojové kódy algoritmu POEMS
  - Pomocné skripty
    * Spoustec: *Spouštěč testů - java*
    * funkce.m: *Vykreslení průběhu - MATLAB*
    * vysledky.m: *Skript na vyhodnocení výsledků z testů - MATLAB*