České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů

Diplomová práce

Použití genetických algoritmů pro syntézu prostředků pro vestavěnou diagnostiku

Filip Čech

Vedoucí práce: Ing. Petr Fišer Ph.D.

Studijní program: Elektrotechnika a informatika strukturovaný magisterský
Obor: Informatika a výpočetní technika
leden 2008

Poděkování

Na tomto místě bych rád poděkoval panu Ing. Petru Fišerovi Ph.D. za jeho cenné rady a připomínky.

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.

Nemám vážný důvod proti užití tohoto školního díla ve smyslu $60 Zákona č. 121/2000 Sb., o právu autorském , o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 16.1.2008 .....

Abstract

This thesis is dealing with a design of a genetic algorithm for a specific problem in a synthesis of tools for build-in self-test. Design of the algorithm is focused on chromosome encoding and potentialities of different genetic operators application. Within the tests, the best combination of genetic operators will be evaluated. Achieved results will be compared to existing ones, randomly controlled Column-Matching algorithm and a random generator. At the end, a design of a messy genetic algorithm for this problem will be discussed.

Abstrakt

Diplomová práce se zabývá návrhem genetického algoritmu pro specifický problém při syntéze prostředků pro vestavěnou diagnostiku. Návrh algoritmu se zaměřuje na kódování chromozomu a vhodnosti použití různých genetických operátorů. V rámci testů bude vyhodnocena nejlepší kombinace genetických operátorů. Dosažené výsledky genetického algoritmu budou porovnány s existujícím, náhodně řízeným Column-Matching algoritmem a náhodným generátorem řešení tohoto problému. Na závěr bude provedena analýza návrhu MGA (Messy Genetic Algorithm) pro tento problém.

Obsah

1 Úvod 13
2 Popis problému 16
2.1 Metoda Column-Maching 17
2.2 B matice 17
2.3 Existující Column-Matching algoritmy 18
2.4 Náhodně řízený Column-Matching algoritmus 19
2.5 Column-Matching metoda pomocí genetických algoritmů 20
3 Úvod do genetických algoritmů 21
3.1 Simple Genetic Algorithm 21
3.2 Teorie schémat 22
3.3 Hypotéza stavebních bloků 23
3.4 Messy Genetic Algorithm 23
4 Návrh genetického algoritmu - GCMA1 25
4.1 Analýza složitosti problému 25
4.2 Kódování jedince 26
4.3 Použité metody křížení 29
4.3.1 1-bodové křížení 30
4.3.2 Uniformní křížení 31
4.4 Použité metody mutace 32
4.4.1 Náhodná mutace matches 33
4.4.2 Vložení do permutace 33
4.4.3 Prohození sousedů permutace 33
4.5 Selekce 34
4.5.1 Turnaj 34
4.6 Výpočet fitnes 34
4.6.1 Zahazování sloupců 36
4.7 Reprodukční cyklus 37
4.8 Vlastnosti populace 38
4.9 Provedené testy 40
4.9.1 Existující Column-Matching algoritmy 41
4.9.2 Úvodní test genetických algoritmů 41
4.9.3 Vliv metody zahazování slopců 44
4.9.4 Test selekčního tlaku 44
4.9.5 Test poměru křížení 46
4.9.6 Test parametru mutace matches 47
4.10 Naměřené výsledky a porovnání s existujícími metodami 47
5 Modifikace genetického algoritmu – GCMA2 49
5.1 Potenciál kvality kombinace 49
5.2 Vlastnosti sloupců matice T 52

5.3 Kódování jedince a genetické operátory 55
5.3.1 Křížení 55
5.3.2 Mutace 55
5.3.3 Selekce 56
5.4 Změny výpočtu fitnes 57
5.4.1 Změny metody zahazování sloupců 58
5.5 Počítání celkového počtu spárování a přímých spárování 58
5.6 Změny reprodukčního cyklu 59
5.7 Provedené testy 59
5.7.1 Úvodní test 59
5.7.2 Vliv metody zahazování sloupců 63
5.7.3 Test selekčního tlaku 64
5.7.4 Test parametru křížení 66
5.7.5 Test parametru mutace matches 67
5.7.6 Test parametru mutace rank 68
5.7.7 Testy stability výsledného řešení 69
5.7.8 Vliv potenciálu kvality 70
5.7.9 Závislost kvality řešení a doby výpočtu na velikosti populace 71
5.8 Naměřené výsledky a porovnání s existujícími metodami 72
5.9 Optimalizace GCMA2 76
6 Messy Genetic Algorithm pro Column-Matching problém 77
7 Závěr 80
8 Použité zdroje 82
Příloha A – Seznam použitých zkratek 83
Příloha B – Nastavení parametrů programu 84
Příloha C – Obsah přiloženého CD 85

Seznam obrázků

Obr. 1.1 Blokové schéma diagnostických testů [Sta1] 13
Obr. 1.2 Struktura mixed-mode BISTu [Fis1] 14
Obr. 1.3 Vytvoření testovacích vzorků [Fis1] 14
Obr. 1.4 Schéma TPG [Fis1] 14
Obr. 2.1 Přiřazení řádek [Fis1] 16
Obr. 2.2 Příklad spárování [Fis1] 17
Obr. 2.3 Přiřazení vektorů pomocí B matice [Fis1] 18
Obr. 4.1 Reprezentace jedince na úrovni řádků [Fis1] 27
Obr. 4.2 Reprezentace jedince na úrovni párovaných sloupců [Fis1] 28
Obr. 4.3 1-bodové křížení 31
Obr. 4.4 Uniformní křížení 32
Obr. 4.5 Náhodná mutace matches 33
Obr. 4.6 Vložení do permutace 33
Obr. 4.7 Prohození sousedů permutace 34
Obr. 4.8 Zahození sloupce 36
Obr. 5.1 Matice T, redukovaná matice T, matice závislostí, matice doplňků 54
Obr. 5.2 Pole columnsTR_mapping 54
Obr. 5.3 Mutace pole matches 56

Seznam tabulek

Tabulka 4.1 Srovnání SGA a Column-Matching problému 29
Tabulka 4.2 Závislost velikosti populace na vstupním zadání 40
Tabulka 4.3 Naměřené hodnoty náhodně řízeného Column-Matching algoritmu 41
Tabulka 4.4 Rozdělení vstupních matic podle velikosti 42
Tabulka 4.5 Parametry úvodního testu GCMA1 42
Tabulka 4.6 Výsledky úvodního testu GCMA1 43
Tabulka 4.7 Výsledky testu vlivu metody zahazování sloupců GCMA1 44
Tabulka 4.8 Výsledky testu selekčního tlaku GCMA1 45
Tabulka 4.9 Výsledky testu poměru křížení GCMA1 46
Tabulka 4.10 Výsledky testu parametru mutace matches GCMA1 47
Tabulka 4.11 Doporučené nastavení vstupních parametrů GCMA1 48
Tabulka 4.12 Naměřené výsledky GCMA1 a srovnání s Thorough a Random Search 48
Tabulka 5.1 Změna podílu jedniček s přibývajícími páry 50
Tabulka 5.2 Počet hlavních sloupců vstupních matic 53
Tabulka 5.3 Nastavení vstupních parametrů během úvodních testů GCMA2 60
Tabulka 5.4 Výsledky měření s použitím celkového počtu spárování v rámci fitnes 60
Tabulka 5.5 Výsledky měření s použitím redukovaného počtu spárování v rámci fitnes 61
Tabulka 5.6 Výsledky měření bez metody zahazování sloupců GCMA2 64
Tabulka 5.7 Výsledky měření parametru selekce GCMA2 65

Tabulka 5.8 Výsledky měření parametru křížení GCMA2 66
Tabulka 5.9 Výsledky měření parametru mutace matches GCMA2 67
Tabulka 5.10 Výsledky měření parametru mutace pole rank GCMA2 68
Tabulka 5.11 Výsledky měření stability řešení GCMA2 69
Tabulka 5.12 Výsledky měření vlivu potenciálu kvality 71
Tabulka 5.13 Výsledky měření vlivu potenciálu kvality – porovnání 71
Tabulka 5.14 Měření závislosti kvality řešení na velikosti populace GCMA2 71
Tabulka 5.15 Doporučené nastavení vstupních parametrů GCMA2 73
Tabulka 5.16 Naměřené hodnoty GCMA2 73
Tabulka 5.17 Porovnání GCMA2 s náhodným generátorem 74
Tabulka 5.18 Závěrečné porovnání GCMA1, GCMA2 a Thorough Search 75
Tabulka 5.19 Zrychlení optimalizace při použití 1-bodového křížení v GCMA2 76

Seznam grafů

Graf 4.1 Porovnání průběhu fitnes GCMA1 1-bodového a uniformního křížení 43
Graf 4.2 Porovnání vývoje fitnes GCMA1 při různém selekčním tlaku 46
Graf 5.1 Průběh fitnes a dalších vlastností populace GCMA2 61
Graf 5.2 Porovnání průběhu fitnes 1-bodového a uniformního křížení GCMA2 63
Graf 5.3 Porovnání průběhu fitnes při různém selekčním tlaku GCMA2 65
Graf 5.4 Porovnání průběhu fitnes při různém poměru křížení GCMA2 66
Graf 5.5 Porovnání průběhu fitnes při různém míře mutace matches GCMA2 67
Graf 5.6 Porovnání průběhu fitnes při různé mutaci rank GCMA2 69
Graf 5.7 Porovnání průběhu fitnes při různé velikosti populace GCMA2 72

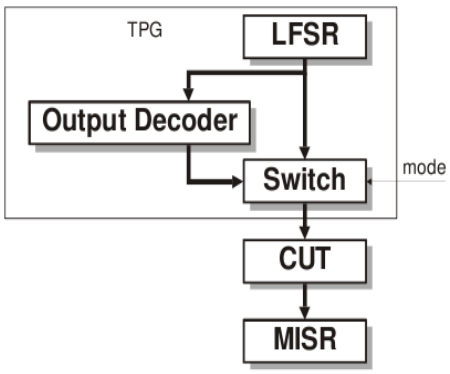
1 Úvod

Vzhledem k rostoucí složitosti současných VLSI obvodů se stává jejich testování čím dál tím důležitější. Ze stejných důvodů je pro testování nemožné využití externích testovacích nástrojů a je nutné použít metod vestavěné diagnostiky (Build-In Self-Test - BIST), které spočívají v "obalení" testovaného obvodu (Circuit Under Test - CUT) vestavěnými diagnostickými prostředky.

Obr. 1.1 Blokové schéma diagnostických testů [Sta1]

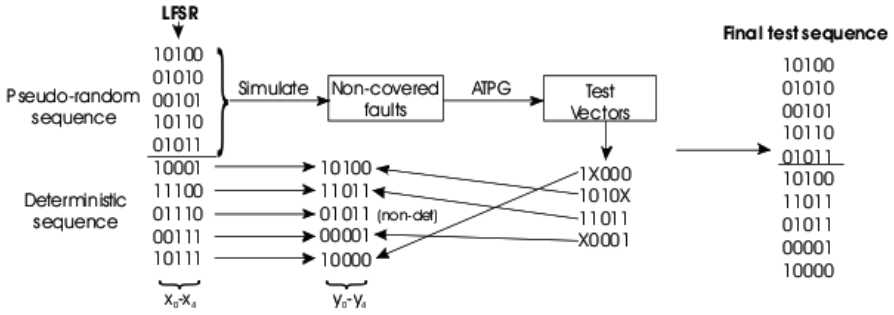
Při návrhu diagnostického testu je nutné brát ohled na následující 4 faktory: pokrytí poruch (množství poruch, které test dokáže odhalit), prostorové nároky testu (velikost zdrojů potřebných pro generování testu, paměťové členy D-KO), doba trvání testu a délka návrhu BIST testu. Velké pokrytí znamená buď dlouhou dobu testu (triviální test) nebo velké prostorové nároky (deterministický BIST). Pseudo náhodné testování znamená nejjednodušší kompromis prvních 3 faktorů. S minimálními prostorovými nároky může být obvod otestován z více než 90% v poměrně malém počtu taktů (tisíce).
Metodou testování, která v sobě zahrnuje prvky pseudo-náhodného a deterministického testování je mixed-mode BIST. Jednoduše odhalitelné poruchy jsou detekovány pseudo náhodnou částí mixed-mode BISTu, deterministická část je určena pro odhalení poruch zbývajících.

Diplomová práce se bude blíže zabývat návrhem generátoru testovacích vzorků (test pattern generator – TPG) pro mixed-mode BIST. Mixed-mode BIST probíhá ve dvou po sobě jdoucích fázích – pseudo-náhodná a deterministická. Jeho struktura je zachycena na obr. 1.2. Pseudo-náhodné vzorky jsou vytvořené v generátoru pseudo-náhodných vzorků (psuedo-random pattern generator – PRPG) nejčastěji realizovaném jako lineární zpětnovazební registr (linear feedback shift register - LFSR). V pseudo-náhodné fázi jsou tyto vzorky přímo předávány testovanému obvodu. V deterministické fázi jsou pseudo-náhodné vzorky transformovány v kombinačním obvodu (*výstupní dekodér*, Output Decoder - OD) na vzorky deterministické, které byly předpočítány pomocí ATPG nástroje (automatic test pattern generator). Výstupní dekodér je kombinační obvod, jehož návrh je závislý na předpočítaných pseudo-náhodných a deterministických vzorcích. V dalších kapitolách bude navržen a otestován genetický algoritmus, který si dává za cíl snížit složitost TPG, konkrétně se snaží minimalizovat *logiku výstupního dekodéru* a obvodů sloužících pro přepínání mezi pseudo-náhodnou a deterministickou fází (*přepínací logika*).

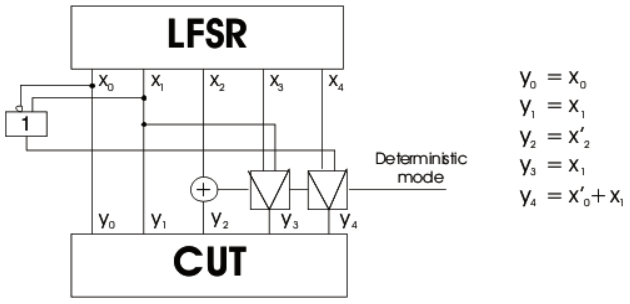


Obr. 1.2 Struktura mixed-mode BISTu [Fis1]

Ilustrační příklad mixed-mode BISTu je znázorněn na obr. 1.3. Zde se provádí syntéza BISTu pro 5-vstupý obvod. Nejdříve je obvodu předloženo 5 5-bitových vzorků vygenerovaných v LFSR, které mají za úkol detekovat jednodušše odhalitelné poruchy. Následně se z těchto vzorků pomocí poruchové simulace naleznou neodhalené poruchy, pro které se vygenerují deterministické vzorky v ATPG. Poté se provede syntéza logiky výstupního dekodéru pro tento test a následné pseudo-náhodné vzorky. Výsledné schéma TPG je na obr. 1.4. Zde můžeme vidět, že u někerých výstupů (y_0, y_1) není zapotřebí žádné logiky dekodéru ani přepínací logiky a pro některé (y_2, y_3) je nutná pouze přepínací logika.



Obr. 1.3 Vytvoření testovacích vzorků [Fis1]



Obr. 1.4 Schéma TPG [Fis1]

Vstupem algoritmu bude sada pseudo-náhodných vzorků, které budou dohromady tvořit *kódovou matici*, společně se sadou deterministických vzorků tvořící *testovací matici*. Výstupem algoritmu bude Booleova funkce, která bude popisovat logiku výstupního dekodéru. Přepínací logika i logika výstupního dekodéru se dá jednoduše z této fuknce odvodit, a to následujícím způsobem:

- Výstupní proměnná y_i je rovna vstupní proměnné x_i. Pak k realizaci y_i nebude zapotřebí žádné logiky přepínací ani výstupního dekodéru (např. y_0 a y_1 na obr. 1.4).
- Výstupní proměnná y_i je rovna negaci vstupní proměnné x_i. Pak logika výstupního dekodéru bude implementována jako invertor. Přepínací logika pro y_i bude multiplexer. Ve skutečnosti je výhodnější oba obvody spojit do hradla XOR (např. y_2 na obr. 1.4).
- Výstupní proměnná y_i je rovna proměnné x_j, přitom $i \neq j$. Zde je zapotřebí přepínací logiky v podobě multiplexeru (např. y_3 na obr. 1.4).
- Výstupní proměnná y_i je rovna negaci proměnné xj, přitom $i \neq j$. Navíc oproti předchozímu případu je nutné negovat výstup z LFSR, tedy přidat invertor. Nicméně pokud klopné obvody použité v LFSR mají negované výstupy, není invertoru potřeba.
- Výstupní proměnná nesplňuje žádné z předchozích pravidel, tzn. je složena z více vstupních proměnných. Pak je složitost logiky výstupního dekodéru závislá na Booleově funkci výstupní proměnné, zatímco přepínání bude zajišťovat opět multiplexer (např. y_4 na obr. 1.4). Minimalizace těchto výstupních proměnných není součástí této práce. Pro jejich výpočet využiji optimalizátoru Booleových funkcí BOOM, viz. [Fis2], [BOOM].

Je zřejmé, že nejmenší prostorové nároky má první zmíněná varianta. V dalších případech prostorové nároky postupně rostou. Z toho bude vycházet i genetický algoritmus, který se bude v první řadě snažit minimalizovat počet výstupních proměnných, které jsou složeny z více vstupních proměnných, zároveň však bude upřednostňovat takové výstupní proměnné y_i, které budou rovny vstupní proměnné x_i, případně její negaci.

Práce bude mít následující strukturu. V kapitole 2 důkladněji popíšu řešený problém a uvedu již existující metody řešení. Další část se bude zabývat základy genetických algoritmů, především jejich nejjednodušší variantou Simple Genetic Algorithm (SGA) a pokročilou metodou Messy Genetic Algorithm (MGA). Návrh a výsledky prvotního genetického algoritmu, který zde budu označovat zkratkou GCMA1 (Genetic Column-Matching Algorithm verze 1), budou prezentovány v kapitole 4. V kapitole následující pak bude navržena modifikoce tohoto algoritmu (GCMA1) tak, že se pokusí využít některých specifických vlastností vstupních matic. Kapitola 6 nastíní problémy spojené s návrhem MGA pro řešení tohoto problému. Kapitola 7 bude obsahovat shrnutí a závěr.

2 Popis problému

Řekněme, že máme n-bitový PRPG běžící po dobu *p* cyklů v deterministické fázi. Vzorky generované pomocí PRPG můžou být popsány maticí C (kódová matice) rozměrů (*p*, *n*). Tyto vzorky PRPG mají být transformovány výstupním dekodérem na testovací vzorky předpočítané ATPG. Testovací vzorky jsou definovány maticí T (testovací matice). Pro *r*-vstupový CUT a test sestávající se z *s* vektorů má matice T rozměry (*s*, *r*). Řádky matic budeme dále nazývat jako vektory.

V matici T mohou být vektory čistě deterministické nebo mohou obsahovat *neúplně určené* (*don't care*) hodnoty, a to v závislosti na ATPG algoritmu použitém pro generování testovací sady. Don't care hodnoty umožňují jednodušší realizaci výstupního dekodéru.

Pro rozměry matic platí následující omezení:

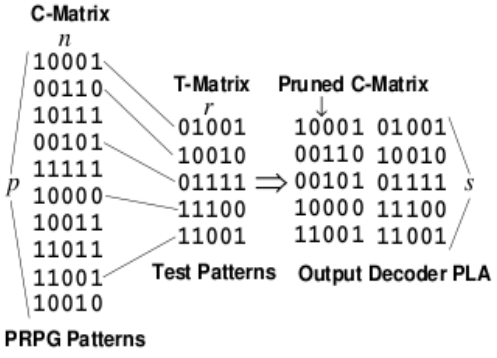
1) $p \leq 2^{n-1}$ (největší počet různých řádků C při *n* sloupcích)

2) $p \geq s$ (musí být dostatečné množství vzorků PRPG pro implementaci testovacích vzorků)

Pro počet sloupců matic C a T (tzn. *n* a *r*) neplatí žádné omezení, nicméně v dalším textu budeme pro jednoduchost uvažovat *n* = *r*.

Výstupní dekodér upravuje vektory matice C tak, aby se transformovaly na všechny vektory matice T. Protože je tato metoda vztažena na kombinační obvody, není důležité pořadí testovacích vzorků, ve kterém jsou předkládány CUT. Z toho vyplývá, že pořadí vektorů matice T může být libovolné. Nalezení transformace mezi maticí C a T znamená přiřadit ke každému řádku matice T řádek matice C. Tuto transformaci ilustruje obr. 2.1, ve kterém jsou 5-bitové testovací vektory přiřazeny k 5-bitovým PRPG vzorkům.

Výstupní dekodér je kombinační blok, který přepočítává *s* *n*-bitových vektorů matice C na *s* *r*-bitových vektorů matice T. Dekodér je reprezentován Booleovou funkcí, která má *n* vstupů a *r* výstupů a může být jednoduše popsána pravdivostní tabulkou, kde výstup koresponduje s maticí T a vstup se skládá z *s* vektorů matice C přiřazených k matici T. Množinu takovýchto *s* vektorů budeme označovat jako prořezaná matice C.

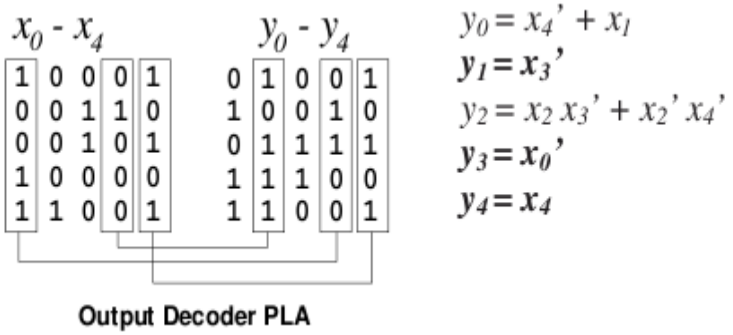


Obr. 2.1 Přiřazení řádek [Fis1]

2.1 Metoda Column-Matching

Column-Matching metoda je založena na přiřazení všech řádek matice T k některým řádkům matice C tak, že některé sloupce matice T jsou rovny některým sloupcům prořezané matice C. V dalším textu budeme tuto rovnost nazývat jako *spárování sloupců* nebo pouze *spárování*. Spárování sloupců nám umožňuje provést takovou implementaci výstupního dekodéru, kde není za potřebí žádné logiky. Díky tomu, že v dnešní době je většina D-KO vybavena i negovanými výstupy, může být tento nápad rozšířen na *negativní párování*, kde mohou být spárovány sloupce s opačnými hodnotami ve všech řádcích.

Ukázkový příklad si můžeme ukázat na obr. 2.2. Zde jsou zobrazeny spárované sloupce prořezané matice C a matice T z obr. 2.1. Sloupec y_1 matice T je negativně spárován s x_3 sloupce matice C, dále y_3 negativně s x_1 a y_4 s x_4. Tudíž výstupy y_1, y_3 a y_4 dekodéru nebudou implementovány pomocí žádné kombinační logiky.



Obr. 2.2 Příklad spárování [Fis1]

2.2 B matice

V této části se budu zabývat algoritmem popsaným ve [Fis1], který pro konkrétní spárování přiřadí vektory vygenerované PRPG (vektory matice C) k vektorům matice T. Algoritmus využívá efektivní heuristiky založené na blokující matici B. Blokující matice je binární matice (obsahuje pouze hodnoty "0" a "1") rozměrů (p, s), tedy obsahuje tolik sloupců, kolik je řádků v matici T a tolik řádků, kolik je řádků v matici C. Hodnota "1" v poli B[k, l] značí, že k-tý řádek matice C může být přiřazen k l-tému řádku matice T, hodnota "0" značí opak.

Na počátku algoritmu jsou všechna pole matice B vyplněna hodnotou "1", protože neexistují žádné omezení pro přiřazení řádek. Jakmile je i-tý sloupec matice C spárován s j-tým sloupcem matice T, jsou pole [k, l] matice B nastaveny na "0", pokud k-tý řádek matice C obsahuje ve sloupci i opačnou hodnotu než l-tý řádek matice T v j-tém sloupci. Tedy řádky, které mají opačné hodnoty v párovaných sloupcích, k sobě nemůžou být přiřazeny.

B[k, l] := "0" pokud (C[k, i] != T[l, j] a T[l, j] != don't care)

V případě, že se má provést negativní spárování, pak jsou pole matice B nastaveny na "0", pokud jsou v příslušných polích stejné hodnoty.

Poté, co se provede tato úprava matice B pro všechny párované sloupce, musí se rozhodnout, jakým způsobem přiřadit řádky matice C k řádkům matice T. V případě, že matice T

neobsahuje neúplně určené hodnoty, jedná se o jednoduchý úkol, protože v matici B nejsou takové řádky, které by měly hodnotu "1" ve více než jednom sloupci (jeden PRPG vzorek nemůže být přiřazen k více než jednomu testovacímu vzorku). Výsledné přiřazení řádek znamená pouze výběr jedné řádky z možných pro každý sloupec. Bohužel pokud matice T obsahuje neúplně určené hodnoty, může být v řádcích matice B více než jedenkrát hodnota "1", protože některé hodnoty v testovacích vzorcích budou rozhodnuty až po přiřazení. Toto dělá z přiřazení řádek NP těžký problém. Příklad přiřazení si ukážeme na obr. 2.3, kde mají být vektory matice T t_1-t_5 přiřazeny k vektorům matice C c_1-c_6. Zde existují dvě možná řešení tohoto problému:

	t_1	t_2	t_3	t_4	t_5
c_1	1	0	0	1	0
c_2	0	1	0	0	0
c_3	0	1	0	0	0
c_4	0	0	1	0	0
c_5	0	0	1	0	1
c_6	0	0	0	1	1

$t_1 - c_1$
$t_2 - c_2$ or c_3
$t_3 - c_4$
$t_4 - c_6$
$t_5 - c_5$

Obr. 2.3 Přiřazení vektorů pomocí B matice [Fis1]

Protože je matice B často velmi veliká, není možné tento problém řešit exaktně a musí se použít nějaké heuristiky. Výběr vhodného algoritmu je klíčový pro dosažení dobrých výsledků. Na příklad pokud bychom na počátku přiřadili c_1 k t_4 z obr. 2.3, neměl by algoritmus žádné řešení, protože by nebylo možné žádné přiřazení k t_1.

Za tímto účelem je použito hladového algoritmu. Na počátku je vybrán sloupec matice B s nejmenším počtem hodnot "1" a je k němu přiřazena řádka, která má v tomto sloupci hodnotu "1" a přitom nejmenší počet "1" v ostatních sloupcích. Pokud v určitém okamžiku existuje sloupec, ve kterém nejsou žádné "1", končí algoritmus chybou a přiřazovací proces je zastaven.

2.3 Existující Column-Matching algoritmy

Do této doby bylo vyvinuto několik algoritmů řešících celý proces párování. V této kapitole zmíním Exact Search, Fast Search a Thourough Search. Algoritmy Fast Search a Thorough Search se detailně zabývá [Fis1] a budou sloužit k porovnání s navrženým genetickým algoritmem.

V algoritmu pro nalezení přesného řešení (Exact Search Algorithm) je nutné projít všechny kombinace různého párování, což vede k nalezení optimálního řešení z pohledu počtu spárovaných sloupců. Nicméně časová složitost tohoto algoritmu je s počtem sloupců exponenciální, a tak není možné použít tohoto algoritmu pro praktické problémy.

Dále může být použita jednoduchá heuristika. Pokud je během přiřazování řádek odhalena dvojice sloupců matice T a matice C (*pár*), kterou se nepovede spárovat (*neplatný pár*), je celý proces zastaven. Toto je nejrychlejší algoritmus a může být často použitelný pro řešení problémů s velkým počtem vstupních proměnných CUT. Protože se přiřazení řádků opakuje po přidání každé párované dvojice sloupců a může existovat maximálně *r* spárovaných dvojic, pak v nejhorším případě může mít tento algoritmus složitost $O(r \cdot p \cdot s)$. Tato situace nastane pokud se podaří

spárovat všech *r* sloupců matice T. Tento algoritmus nazýváme Fast Search.

Výsledek může být dále vylepšen, pokud v případě, že narazíme na neplatný pár, zkoušíme spárovat všechny další možné páry. Toto samozřejmě způsobí delší dobu běhu programu, nicméně stejně tak se zvětší i počet spárovaných sloupců. Tento algoritmus se nazývá Thorough Search. Jeho nejhorší možná složitost je $O(n \cdot r^2 \cdot p \cdot s)$, nicméně v nejlepším případě se může dostat na úroveň rychlého průchodu.

2.4 Náhodně řízený Column-Matching algoritmus

V této části důkladněji popíšu metody Fast Search a Thorough Search popsané ve [Fis1], které budou sloužit k porovnání s navrhovaným genetickým algoritmem GCMA1 i GCMA2. Především se zaměřím na Fast Search, jelikož Thorough Search je pouze malou modifikací.

V praxi se setkáváme většinou s takovými problémy, kde je počet PRPG vzorků mnohem větší než počet vzorků testovacích ($p >> s$). Z toho důvodu můžou být na počátku algoritmu spárovány téměř jakékoliv sloupce, protože je velký výběr z možných přiřazení k řádkům matice C, a proto je výběr sloupců pro spárování řízen náhodně.

Když jsou vybrány dva sloupce určené pro spárování (*kandidáti*), musí být prověřena platnost tohoto páru pomocí B matice. Proto po přidání nového páru musí následovat přiřazení řádků, které o platnosti páru rozhodne. Jestliže vznikne při přiřazení chyba, párovací proces je zastaven a jako výsledek je bráno poslední platné spárování. Poté co doběhne párovací proces, nahradí se neúplně určené hodnoty odpovídajícími hodnotami “0“ a “1“ z matice C a proběhne kompakce matice T, která často zkrátí délku BISTu a také zjednoduší logiku výstupního dekodéru. Nakonec se z matice odstraní spárované výstupní proměnné a hodnoty zbývajících výstupních proměnných jsou vytvořeny pomocí dvouúrovňového minimalizátoru Booleových funkcí.

Algoritmus může být popsán pomocí následujícího pseudokódu. Vstupem do algoritmu jsou matice C a T, výstupem je minimalizovaná Booleova funkce.

```
ColumnMatching(C, T) {
   // inicializuje matici B
   for (k = 0; k < C_matrix_rows; k++)
      for (l = 0; l < T_matix_rows; l++)
         B[k, l] = “1“;
   A = NULL;
   do {
      // nahodne vybereme sloupce
      i = random(C_matrix_rows);
      j = random(T_matrix_rows);
      for (k = 0; k < C_martrix_rows; k++)
         // modify blocking matrix
         for (l = 0; l < T_martrix_rows; l++)
            if (T[l, j] != DC && C[k, i] != T[l, j])
               B[k,l] = “0“;
      A' = A;                  // provedeme zalohu prirazeni
      A = MakeRowAssignment(B);
   } while (A != FAILED);
   Substititute_DCs(T);              // nahrazeni DC hodnot pomoci 0 nebo 1
   CompactTest(T);                   // provedeme test kompaktnosti
   ExtractMatches(C, T);             // odstranime sparovane sloupce
   F = Minimize(A');          // vytvorime celkovou vystupni funkci
   return F;
}
```

Thorough Search představuje pouze malou modifikaci. Pokud při přiřazení narazíme na

neplatný pár, nekončí proces přiřazování chybou, ale zkouší se i další možné kombinace sloupců T a C. Proces je ukončen, až když neexistuje žádný další pár, který by přiřazení vyhovoval.

2.5 Column-Matching metoda pomocí genetický algoritmů

V této práci se budu zabývat genetickým algoritmem, který se pokusí nalézt pro každou instanci problému co největší počet spárování s možností negativního párování. Vedlejším optimalizačním kritériem bude nalezení co největšího počtu přímých spárování. Algoritmus bude uvažovat neúplně určené hodnoty matice T a nebude klást žádná omezení na počet sloupců matic C a T. Pro implementaci a popis v dalším textu jsem zvolil jazyk C++. Program byl odladěn a otestován na operačním systému Linux Fedora 6.

3 Úvod do genetických algoritmů

Genetické algoritmy jsou metody vycházející z principů evoluční biologie. Využití této metody je vhodné ve velice složitých problémech, pro které neexistuje exaktní algoritmus, který by nalezl řešení v přijatelném čase. Na začátku algoritmu se vytvoří počáteční *populace* složená z *jedinců*, kteří reprezentují jedno z možných řešení daného problému. Abstraktní reprezentace jedince se nazývá *chromozóm*. Jedinci mohou být v počáteční populaci vytvořeni buďto náhodně nebo pomocí heuristiky. Poté následuje proces postupného vylepšování populace opakovanou aplikací genetických operátorů, který vede k evoluci takových jedinců, kteří lépe vyhovují stanoveným podmínkám než jedinci z předchozích generací. Pro vyhodnocení kvality jedince se využívá funkce *fitnes* a její vlastnosti závisí na konkrétním problému. Genetický algoritmus definuje operátory *křížení*, *mutace* a *selekce*. Informace o genetických algoritmech byly čerpány z [Whi1], [Smi1], [Har1], [Gol3], [GA] a [WIKI1].

3.1 Simple Genetic Algorithm

Nejednodušším a nejčastějším příkladem genetického algoritmu je SGA (Simple Genetic Algorithm), kde je pro *zakódování chromozómu* využito binární pole. Jednotlivé pozice chromozómu se nazývají *geny*.

Aplikací operátoru křížení nad dvěma jedinci (*rodiči*) vznikne jedinec (*potomek*), který má část binárního pole od jednoho rodiče a zbytek od druhého. Volbou genů, které dostane potomek od prvního rodiče můžeme odlišit druh křížení:

1) Jednobodové křížení - Náhodně zvolíme pozici k v binárním poli. Od prvního rodiče dostane potomek geny 1 až k, od druhého zbytek.
2) Vícebodové křížení (n-bodové) - Náhodně zvolíme n pozic k_1, k_2, ..., k_n, $k_1 > k_2 > .. > k_n$. Od prvního rodiče dostane potomek geny 1 až k_1, k_2 až k_3, atd., od druhého dostane zbytek genů.
3) Uniformní křížení - Pro každý gen se náhodně rozhoduje, od kterého rodiče jej potomek získá.

Operátor mutace je v SGA prováděn tak, že každý gen je s pravděpodobností p_M invertován. Proměnná p_M se nazývá pravděpodobnost mutace.

Výběr jedinců pro novou generaci neboli selekce, je založen na evolučním principu, že přežívají pouze ti nejlepší jedinci. Jedinci, kteří mají lepší fitnes mají větší pravděpodobnost výběru. Této myšlenky se drží i následující nejčastěji používané metody pro selekci:

1) Ruleta – Každému jedinci se přiřadí část pomyslné rulety. Velikost této části může být závislá na kvalitě fitnes nebo na umístění jedince v populaci seřazené podle fitnes. Jedinec je vybrán, jestliže se ruleta zastaví v části přiřazené tomuto jedinci. Pro výběr n jedinců je nutné n-krát otočit ruletou.
2) Turnaj – Na počátku turnaje se z populace vybere zcela náhodně n jedinců. Vítězem turnaje, tj. vybraný jedinec, se stane ten, který má z vybraných nejlepší fitnes.
3) Lineární selekce - Pravděpodobnost výběru jedince je lineárně závislá na umístění jedince v

populaci seřazené podle fitnes

Na počátku algoritmu se vytvoří první populace. Tato populace je složená buď ze zcela náhodných jedinců nebo je možné pro jejich vytvoření použít heuristiky. Další populace jsou vytvářeny podle následujícího reprodukčního cyklu:

1) Vyber rodiče z populace do párovacího pole podle zvolené selekční metody, velikost pole = velikost populace
2) Zamíchej jedince v párovacím poli
3) Pro každý za sebou jdoucí pár proveď křížení s pravděpodobností p_C, jinak rodiče zkopíruj
4) Pro každého potomka proveď mutaci s pravděpodobností p_M pro každý bit
5) Zaměň celou populaci vzniklými potomky

Genetický algoritmus končí v momentě, kdy již není pravděpodobné další zlepšení. Sledovaným faktorem bývá nejčastěji průměrná fitnes jedinců v populaci.

3.2 Teorie schémat

Vysvětlení funkčnosti genetických algoritmů nabízí teorie od J. H. Hollanda. Teorie je založena na *schématech*, tzn. šablony zastupující podmnožinu řetězců mající společné hodnoty na některých pozicích. Například uvažujme binární řetězec délky 6. Schéma 1**0*1 popisuje množinu všech řetězců délky 6, mající hodnotu 1 na pozicích 1 a 6 a 0 na pozici 4. Znak * je symbol, který znamená, že na dané pozici může být hodnota jak 1, tak 0. Řádem schematu se označuje počet definovaných hodnot v šabloně, zatímco určující délka je vzdálenost mezi první a poslední pozicí s definovanou hodnotou. Schéma 1**0*1 je řádu 3 a jeho určující délka je rovna 5. Fitnes schématu je průměrná fitnes všech řetězců, které mu odpovídají. Teorie schémat říká, že průměrná fitnes krátkých schémat nízkého řádu s přibývajícími generacemi exponenciálně roste. Popisuje jí následující nerovnice:

$$m(H,t+1) \geq \frac{m(H,t)f(H)}{a_T}[1-p], \tag{3.1}$$

kde $m(H, t)$ je počet řetězců patřících do schématu H v generaci t, $f(H)$ je fitnes schématu H a a_T je průměrná fitnes generace t. Pravděpodobnost roztržení p je pravděpodobnost, že křížení nebo mutace zruší schéma H. Tato pravděpodobnost může být vyjádřeno následovně:

$$p = \frac{d(H)p_C}{(l-1)} + o(H)p_M$$

kde $o(H)$ je řád schématu, l je délka řetězce, p_M je pravděpodobnost mutace a p_C je pravděpodobnost křížení. Z výše uvedeného mimo jiné vyplývá, že schéma s menší určující délkou $d(H)$ bude zrušeno s menší pravděpodobností. Důvodem, proč je ve vztahu 3.1 použita nerovnost, je skutečnost, že teorie schémat opomíjí malou, přesto nenulovou pravděpodobnost, že bude řetězec patřící do schématu H nově vytvořen, například mutací řetězce, který do schématu H nepatří.

3.3 Hypotéza stavebních bloků

Důvody úspěchů genetických algoritmů nejsou příliš známy. Nejznámější úvahou na tomto poli je *hypotéza stavebních bloků* od D. Goldberga. Skládá se z popisu abstraktního adaptivního mechanismu, který provádí adaptaci kombinováním stavebních bloků (schémata nízkého řádu s malou určující délkou a nadprůměrnou fitness), a předpokladu, že genetický algoritmus tento mechanismus nepřímo, ale přitom efektivně implementuje. Protože stavební bloky mají malou určující délku, vyplývá z této hypotézy, že nejvhodnější metodou křížení je 1-bodové křížení. Naopak nejvíce tyto bloky narušuje křížení uniformní.

Hypotéza stavebních bloků však byla mnohokrát ostře kritizována, a to především z důvodu, že postrádá teoretické ověření. Navíc řada experimentů ukázala, že uniformní křížení je často tou nejlepší metodou. Nesouhlas s touto teorií vyslovil na základě svých zkušeností např. D. Fogel: "Obecně vede uniformní křížení k lepším výsledkům než 2-bodové křížení, které vede k lepším výsledkům než 1-bodové křížení."

3.4 Messy Genetic Algorithm

Z myšlenky stavebních bloků vychází Messy Genetic Algorithm (dále pouze MGA). MGA se od klasického SGA liší následovně:

1) MGA používá proměnnou délku řetězců (chromozómů), které můžou být vzhledem k řešenému problému nedostatečně nebo nadbytečně specifikovány (z angl. underspecification, resp. overspecification)
2) MGA používá na místo operátorů křížení, které pracují s pevnou délkou chromozómu, operátory *cut* (v překl. uřízni) a *splice* (v překl. spoj)
3) MGA rozděluje evoluční vývoj do dvou fází: prvotní (z angl. primordial) fáze a juxtapoziční fáze
4) MGA používá *competitive template* (dále pouze *šablona*) pro vyzdvižení lepších stavebních bloků

Název „Messy“ (v překl. neuspořádaný) vznikl z proměnné délky řetězců, které můžou být vzhledem k řešenému problému nedostatečně nebo nadbytečně specifikovány. Například 3-bitový řetězec 111 klasické SGA může být v MGA reprezentován (s použitím zápisu podobného jazyku LISP) jako ((1 1) (2 1) (3 1)), kde každý bit je identifikován svým jménem a hodnotou. Protože v MGA je povolena proměnná délka řetězců, musí být v MGA nalezena interpretace takových řetězců, které mají příliš málo nebo mnoho bitů. Například řetězce ((1 1) (2 1)) a ((1 1) (2 1) (3 1) (1 0)) jsou oba platnými řetězci MGA v 3-bitovém problému, přestože v prvním řetězci chybí třetí bit a naopak v druhém řetězci je první bit vícekrát. V případě konfliktu prvního bitu v druhém řetězci se v MGA využívá pravidla first-come-first-serve (v překl. první přijde, první slouží) v pořadí zleva doprava. Zatímco nadbytečná specifikace může být řešena tímto relativně jednoduchým pravidlem, nedostatečná specifikace je problém mnohem složitější, a jeho řešení pomocí šablon bude diskutováno později.

Pro rekombinaci řetězců proměnné délky je opuštěno od křížení SGA (1-bodové křížení), které pracují s pevnou délkou řetězce, a je na místo toho použito operátorů cut a splice. Cut a splice operátory jsou velmi jednoduché. Cut přeruší řetězec s pravděpodobností $p_C = (l - 1)p_K$, která roste s délkou řetězce l, operátor splice spojí dva řetězce dohromady s pevnou pravděpodobností p_S. Dohromady mají operátor cut a splice zhruba stejnou možnost narušení stavebních bloků a podobnou sílu jako jednoduché křížení. Detailní teoretické úvahy musely také uvážit postupné vytlačování dobrých stavebních bloků a to vedlo k rozdělení evolučního procesu na dvě fáze.

SGA přistupuje k řetězcům po celou dobu běhu algoritmu jednotně. Na začátku je vytvořena populace (většinou náhodně). Ostatní generace mají stejnou velikost, přičemž používají selekci, křížení a ostatní genetické operátory k vytvoření nové populace z aktuální. Naproti tomu MGA dělí genetický vývoj na dvě rozdílné fáze – prvotní a juxtapoziční. V prvotní fázi je vytvořena populace tak, aby obsahovala všechny možné stavební bloky pevné délky a omezené určující délky. Následně se během několika generací, kdy dochází pouze k reprodukci bez jiných genetických operací, zvýší podíl dobrých stavebních bloků. V ten samý moment je většinou populace po určitých intervalech (generacích) zmenšována na polovinu.

S takto vyvinutou populací s velkým podílem dobrých stavebních bloků dále pracuje juxtapoziční fáze, která již nemění velikost populace a používá rekombinace, cut, splice a ostatních genetických operátorů (vedle selekce a operátoru cut a splice obyčejně není využito žádných dalších genetických operátorů).

Toto provedení požadovalo ohodnocení stavebních bloků, které si vynutilo použití šablon a souvisí s nutností vyřešit problém nedostatečné specifikace. Pokud máme l-bitový problém a řetězec má méně jak l bitů, pak jak potom nalézt objektivní funkci, která jej ohodnotí? Řešení v podobě competitive template používá šablony lokálního optima pro doplnění chybějících bitů do neúplných řetězců. Tímto způsobem pouze vyčnívající stavební bloky získají lepší hodnotu fitnes, než je fitnes šablony, a dojde k obohacení jejich počtu během prvotní fáze. V některých problémech, resp. v některých kódování, však je možné ohodnotit neúplné řetězce bez doplnění chybějících pozic z šablony.

Nejčastější forma MGA využívá tzv. level-wise processing, tzn. postupování algoritmu po úrovních, resp. se stoupající délkou stavebního bloku. Začíná úrovní 1 (délka stavebního bloku = 1), kde k ohodnocení stavebních bloků slouží náhodně vytvořená šablona. Výsledkem vnitřního cyklu pro každou úroveň, tzn. prvotní a juxtapoziční fáze, je lokální optimum, které se stane šablonou pro úroveň následující. Ukončení algoritmu se může provést například po učitém počtu úrovní, kdy nedojde ke zlepšení.

Fast Messy Genetic Algorithm (FMGA) je modifikací MGA, která urychluje prvotní fázi MGA, kde na místo redukce populace s konstantní délkou stavebních bloků provádí prostřednictvím odstraňování genů zkracování stavebních bloků v populaci o konstantní velikosti.

Více o MGA pojednává [Gol1] a [Gol2]. V této práce se budu MGA více zabývat v 6. kapitole, kde se pokusím navrhnout MGA, resp. FMGA, pro Column-Matching problém.

4 Návrh genetického algoritmu - GCMA1

V této kapitole se budu zabývat návrhem první verze genetického algoritmu, pro který budu dále v textu používat zkratku GCMA1, tzn. Genetic Column-Matching Algorithm verze 1.

4.1 Analýza složitosti problému

Vstupem do našeho algoritmu bude matice C (p, n) a matice T (s, r). Za koncové stavy stavového prostoru budeme pokládat prořezanou matici C (s, r) a pole spárovaných dvojic (sloupec matice T, sloupec matice C) délky r, ke kterému je navíc nutné udržovat informace o negování výstupů dekodéru a informace, jestli se pár povedlo či nepovedlo spárovat (pár je platný resp. neplatný).

Prořezanou matici C můžeme chápat jako přiřazení s vektorů matice C k s vektorům matice T. Protože celkový počet vektorů matice C je p, je počet všech těchto přiřazení C_p roven

$$C_P = V_S(p) = \frac{p!}{(p-s)!}$$

Dále ke každému z r sloupců matice T může být spárován jakýkoliv sloupec z n sloupců prořezané matice C, tudíž počet různých spárování sloupců je

$$C_S' = V_N(r)' = n^r$$

Uvážíme-li navíc, že každý sloupec matice T může či nemusí být negován, dostaneme

$$C_S = V_{2N}(r)' = (2\mathrm{n})^r$$

Počet konečných stavů stavového prostoru je tedy

$$C_P \cdot C_S = \frac{p!}{(p-s)!} \cdot (2\mathrm{n})^r$$

Při hledání řešení bychom v krajním případě, tzn. v případě že se povede spárovat všechny sloupce, museli porovnat všechny hodnoty matice T s odpovídajícími hodnotami v prořezané matici C. Počet těchto porovnání by byl roven

$$O_P = r \cdot s$$

V případě, že bychom sestrojili algoritmus procházející celý stavový prostor hrubou silou, měl by následující asymptotickou složitost:

$$O(C_P \cdot C_S \cdot O_P) = O\left(\frac{p!}{(p-s)!} \cdot (2\mathrm{n})^r \cdot r \cdot s\right)$$

Algoritmus, který by prošel všechny kombinace spárování a použil heuristiky využívající matici B

by měl složitost

$$O(C_S)\cdot O(r\cdot p\cdot s)=O((2\mathrm{n})^r\cdot r\cdot p\cdot s)=O((2\mathrm{n})^r)$$

Vzhledem k velikosti reálných instancí, kdy *n* a *r* se pohybuje mezi 20 až 2000, nemůže tento algoritmus dosáhnout výsledků v únosném čase.

4.2 Kódování jedince

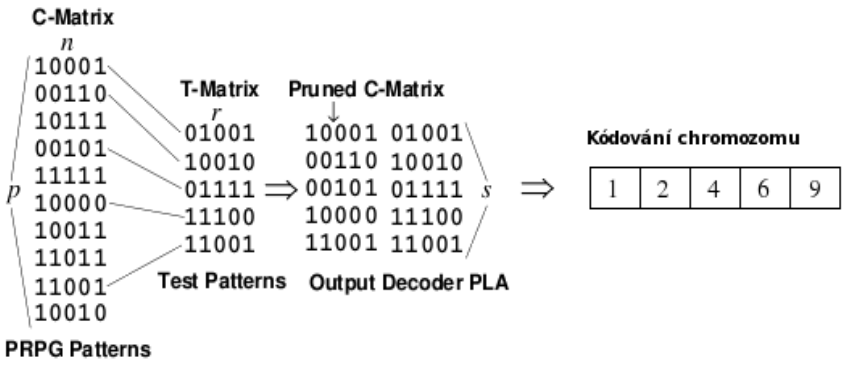
Úspěšnost genetického algoritmu stojí na vhodné volbě způsobu *kódování* (*kódování chromozómu*), což je způsob uložení informací o jedinci. V genetických algoritmech se nejčastěji pracuje s nejjednoduší formou, a to binárním polem. Toto kódování však bohužel, vzhledem ke složitosti našeho problému, není možné použít. Jedinec může reprezentovat úplné řešení (koncový stav stavového prostoru), které je popsáno v předchozí kapitole, nebo pouze řešení částečné (mezistav stavového prostoru), kdy jedinec pokrývá množinu koncových stavů. Pro reprezentaci jedince můžeme uvažovat následující varianty:

1) Pomocí úplného řešení.
2) Na úrovni řádků - prořezaná matice C.
3) Na úrovni sloupců - pole spárovaných sloupců (sloupec matice T, sloupec matice C).

Při volbě kódování je nutné uvážit především následující faktory: efektivitu genetických operátorů, dobu výpočtu fitnes a velikost stavového prostoru, které jedinec pokrývá. Protože je naším úkolem najít co největší počet sloupců matice T, které se dají spárovat se sloupci matice C (*počet platných párů, počet spárování*), musí být tomuto počtu platých párů úměrná fitnes.

Reprezentace pomocí úplného řešení by měla výhodu v rychlém výpočtu fitnes. Jedinec, který by reprezetnoval konečný stav, by byl sloučením jedince na úrovni řádků a jedince na úrovni sloupců. Musel by kódovat jak prořezanou matici C, tak spárování sloupců. Tato varianta by mohla být efektivní, pokud by byly z pohledu genetických algoritmů vhodné i oba v dalším textu popsané přístupy. Jak si ale v následujícím odstavci ukážeme, reprezentace na úrovni řádků není vhodné kódování pro nalezení efektivní metody mutace a křižení, a tak se touto reprezentací nebudu dále zabývat.

Reprezentace na úrovni řádků znamená reprezentaci jedince pomocí prořezané matice C. Tato matice může být implementována jako pole identifikátorů vektorů matice C o velikosti *s* (výška matice T), kde *i*-té pole hodnoty *j* znamená přiřazení *j*-tého vektoru matice C k *i*-tému vektoru matice T. Příklad takovéto reprezentace je na obr. 4.1. V tento moment je důležité se zamyslet nad tím, jakým způsobem by se aplikovaly genetické operátory. Protože vektory matice T nemají mezi sebou žádnou závislost (testovací vzorky mohou být předkládány testovanému obvodu v jakémkoliv pořadí), nemají závislost ani sousední hodnoty v tomto poli a vyhovující metodou pro křížení by bylo jakékoliv křížení SGA z kapitoly 3.1, tedy např. uniformní křížení. Mutace by znamenala náhodné vybrání řádku matice C a vložení do náhodného místa v poli.



Obr. 4.1 Reprezentace jedince na úrovni řádků [Fis1]

Uvažujme nyní pro jednoduchost, že fitnes je lineárně úměrná počtu platných párů a v matici T nejsou žádné neúplně určené hodnoty. Jak při křížení, tak při mutaci bude docházet k nahrazování vektorů v prořezané matici C, resp. ke změně hodnoty v chromozomu. Řekněme, že nahrazujeme jeden vektor prořezané matice C. Pak každá hodnota (0 nebo 1) ve sloupci *j* tohoto vektoru bude s pravděpodobností 1/2 různá od původní hodnoty (0 nebo 1) a pokud byl sloupec *j* součástí platného páru, pak se s pravděpodobností 1/2 tento pár zneplatní. Z toho vyplývá, že s každým měněným vektorem se z tohoto důvodu zmenší hodnota počtu spárovaných sloupců, a tedy i fitnes, v průměru o polovinu. Zároveň může docházet ke vzniku párů nových, nicméně jejich vznik je pouze prvkem náhody. Tato úvaha je samozřejmě značně zjednodušená, například zcela opomíjí don't care hodnoty, nicméně naznačuje, že jak křížení, tak mutace bude značně neefektivní, a proto jsem se rozhodl tento způsob kódování nepoužít.

Posledním typem reprezentace je reprezentace na úrovni párovaných sloupců. Takovýto jedinec představuje ve stavovém prostoru mezistav, pod který spadá $C_P = p! / (p - s)!$ koncových stavů. Z toho vyplývá největší nevýhoda tohoto přístupu, a to dlouhý výpočet fitnes. Implementací může být pole (v dalším textu označené jako pole *matches*), které sloupcům matice T přiřadí sloupce matice C. Dále, vzhledem k funkčnosti negativního párování je nutné vytvořit pole, které bude uchovávat informaci o negování výstupu dekodéru (pole *NM*). Zároveň musí jedinec obsahovat informaci o sloupcích matice T, které jsou součástí platných párů. Nabízí se zde 2 možná řešení:

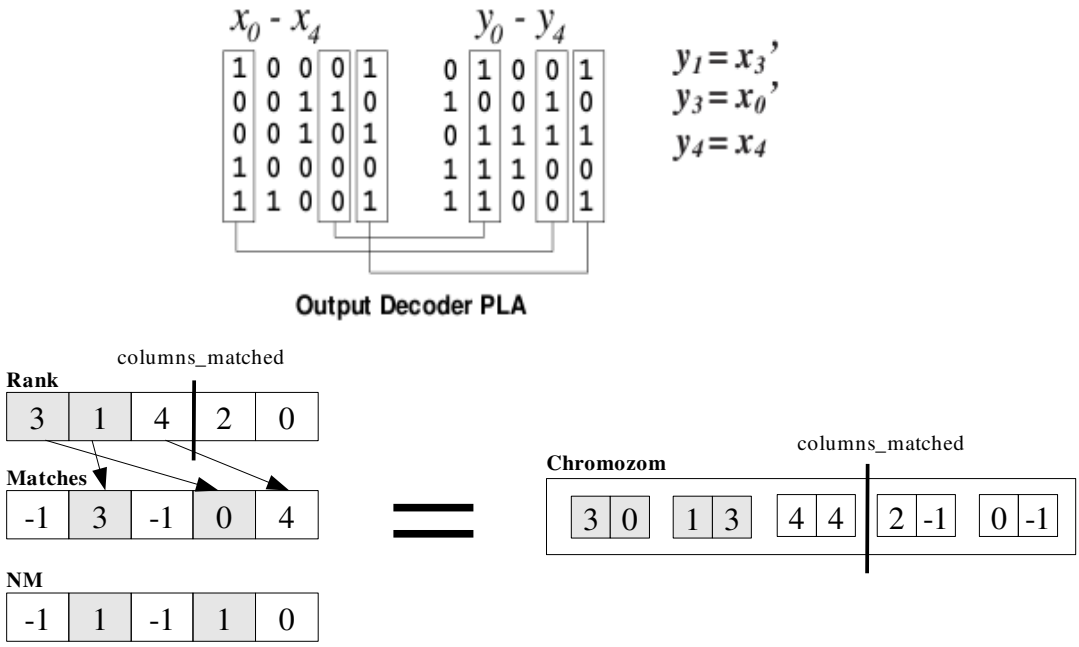
1) Pole délky *r* označující platnosti páru nebo seznam platných párů.
2) Pole délky *r* určující pořadí přiřazování sloupců matice T ke sloupcům matice C. Sloupce v levé části pole budou značit platné páry, přičemž počet platných párů určí proměnná *columns_matched*. V dalším textu budeme toto pole označovat jako pole *rank*.

Protože počet spárovaných sloupců je úměrný fitnes, je nutné tuto informaci vypočítávat spolu s výpočtem fitnes. V prvním případě počet spárovaných sloupců přímo vyplývá z použitých struktur (např. délka seznamu platných párů). Výpočet fitnes by byl triviální, něměli bychom ovšem zaručené, že po aplikaci genetických operátorů bude jedinec stále řešením. Abychom měli zaručeno, že jedinec je řešením, musely by být do algoritmu přidány další kontrolní a opravné mechanismy. Z toho důvodu jsem se rozhodl pro variantu druhou. Výpočet fitnes bude založen na heuristice využívající matici B. Postupně se budeme snažit přidávat páry v pořadí pole rank a s

každým úspěšným spárování zvětšíme hodnotu columns_matched, tj. počet párů, které se povede spárovat. Nad polem rank můžou být výhodně aplikovány genetické operátory křížení i mutace.

Aplikací operátorů křížení nad dvěmi kvalitními jedinci můžeme získat potomka, který bude mít dobré páry, resp. stavební bloky, od obou rodičů a díky tomu i lepší fitnes. Různé operátory budou diskutovány v kapitolách 4.3 a 4.4. Největší nevýhodou tohoto přístupu je složitý výpočet fitnes, nicméně při aplikaci heuristiky využívající matici B může složitost tohoto výpočtu klesnout na $O(r \cdot p \cdot s)$.

Kódování chromozómu jedince na úrovni sloupců je zachyceno na obr. 4.2. V levé dolní části obrázku jsou zobrazena všechna pole jedince, která jsou součástí chromozómu. V poli matches a NM jsou zde pro větší srozumitelnost uvedeny hodnoty -1 ve sloupcích matice T, které se nepovedlo spárovat (sloupce 0 a 2). Ve skutečnosti však budou v těchto polích konkrétní hodnoty. Důležitá je hodnota columns_matched, která se v tomto případě rovná 3 a určuje počet platných spárování. V pravé dolní části uvádím způsob, kterým budu v dalších obrázcích znázorňovat chromozóm jedince. Negativní páry jsou označeny šedě. Protože informace o negování výstupní proměnné – pole NM - se bude měnit vždy spolu s polem matches, nebudu v dalších obrázcích pro větší přehlednost toto značení používat. V horní části obrázku je Booleova funkce, kterou tento jedinec představuje. Všimněme si, že pokud prohodíme sloupce matice T v levé části pole rank před hodnotou columns_matched, pak tento nově vzniklý jedinec bude stále reprezentovat tu samou Booleovu funkci.



Obr. 4.2 Reprezentace jedince na úrovni párovaných sloupců [Fis1]

Implementace struktury pro zakódování jedince bude následující:

```
class Jedinec {
  vector<int> matches;// velikost pole=r, hodnoty=sloupce matice C (0 až n-1)
  vector<int> rank;   // velikost pole=r, hodnoty=sloupce matice T (0 až r-1)
  vector<int> NM;            // velikost pole = r, hodnoty 0 nebo 1
  int columns_matched;
  double fitnes;
};
```

`matches` - přiřazení sloupců matice C ke sloupcům matice T

Př. matches[7] = 5; // k 7. sloupci matice T patří 5. sloupec matice C

`rank` - pořadí sloupců matice T při výpočtu fitnes

- toto pole má tu vlastnost, že každý sloupec T může být v poli pouze jednou

Př. rank[0] = 7; // při výpočtu fitnes se nejdříve pokusíme spárovat 7. sloupec T

`NM` - příznaky o negování výstupů z výstupního dekodérů (sloupce matice T)

Př. NM[7] = 1; // 7. sloupec matice T je negativně párován

`columns_matched` - počet sloupců, které se při výpočtu fitnes povedlo spárovat. Párování probíhalo v pořadí podle pole rank.

`fitnes` - ohodnocení jedince závislé na hodnotě columns_matched. Jak jsem již uvedl, fitnes je úměrná počtu spárovaných sloupců, nicméně její hodnota může být závislá i na jiných vlastnostech jedince.

	SGA	GCMA1
chromozóm	binární řetězec	pole rank
abeceda	0, 1	$\{sloupce\ m.C\}\times\{0,1\}$
kardinalita	2	$2\cdot n$
kódování genu i	pole bin. řetězce, ret[i]	j = rank[i]; matches[j], NM[j]
omezení	žádné	pole rank je permutace
další informace	žádné	columns_matched

Tabulka 4.1 Srovnání SGA a Column-Matching problému

S takto navrženým kódováním můžeme nyní upravit úvahy o stavovém prostoru pro genetické algtoritmy. Koncovým bodem stavového prostoru je jedinec, jehož chromozómem je pole rank délky *r*. Abecedou použitou v chromozómu jsou sloupce matice C o počtu *n* spolu s příznaky o negování. Velkost abecedy, resp. kardinalita našeho problému je

$$k = 2 \cdot n$$

Protože používáme pro kódování pole NM, matches a rank, přitom pole rank je permutace, je velikost stavového prostoru (tzn. počet všech různých jedinců)

$$C = k^r \cdot V_R(R) = 2^r \cdot n^r \cdot r!\ ,$$

kde *n* je počet sloupců matice C a *r* je počet sloupců matice T.

Z výše uvedených vztahů vyplývá, že oproti binárnímu zakódování, kde je velikost stavového prostoru rovna 2^r, je stavový prostor Column-Matching problému s použitím výše popsaného způsobu kódování mnohem větší.

4.3 Použité metody křížení

Určit nejvhodnější křížení pro náš problém není jednoduchý úkol. Nově vznikající potomek bude mít pole matches, rank a NM od jeho rodičů. Mezi poli matches a NM je zřejmá závislost. V případě, že se jedinci změní *i*-tá hodnota v poli NM, a přitom *i*-tý sloupec tvoří platný pár, znamená to, že se tento pár zneplatní (počet spárovaných sloupců klesne o 1). Z toho vyplývá, že jestliže

chceme, aby potomek zdědil sloupec C v *i*-tém poli matches, musí rodič potomkovi předat i *i*-tou hodnotu v poli NM. Proto křížení pole matches a NM probíhá vždy společně. Pokud bychom uvažovali pouze křížení těchto dvou polí, pro která neplatí žádná omezení, mohli bychom pro křížení použít modifikace jakékoliv metody křížení SGA.

Nicméně při návrhu křížení musíme brát v potaz i pole rank. Pro pole rank platí, že se v něm každý sloupec matice T vyskytuje právě jednou. Pole rank nazýváme *permutací* a při jeho křížení se nesmí narušit tato jeho vlastnost. Pro permutace existují speciální metody křížení jako order 1 křížení a PMX.

Na křížení můžeme nahlížet jako na kopírování stavebních bloků, chápeme-li stavební bloky jako schémata s nadprůměrnou fitnes, od obou rodičů na potomka. Existence stavebních bloků v chromozómu je zaručena selekcí, která pro křížení vybírá jedince s nadprůměrným fitnes. V našem případě však nesmíme zapomenout na to, že se tyto stavební bloky vyskytují pouze v levé části pole rank. V pravé části pole rank se vyskytují taková schémata, o kterých nemáme žádnou informaci a tedy nemá význam je během křížení uvažovat.

Zároveň si před návrhem křížení musíme uvědomit určující délku stavebních bloků v poli rank. K tomuto problému můžeme zaujmout dva přístupy:

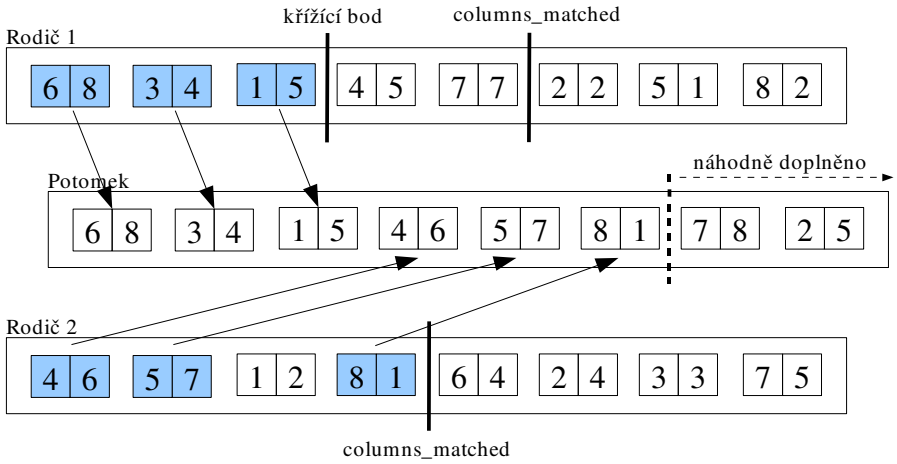
1) Určující délka by měla být co možná nejkratší. V tomto případě vycházíme z hypotézy stavebních bloků. Tím, že stavební bloky mají krátkou určující délku vlastně říkáme, že geny v těsné blízkosti mají větší praděpodobnost, že patří do jednoho stavebního bloku, než geny vzdálené. K nejméně častému rozbourávání těchto stavebních bloků dojde křížením s minimálním počtem křížících bodů. Jako nejlepší se zde jeví varianta 1-bodového křížení. S tímto přístupem si musíme uvědomit, že není možné příliš často přemísťovat geny v chromozómu (poli rank), protože jejím přemístěním zničíme řadu stavebních bloků.
2) Určující délka je rovna délce chromozómu. Zde neklademe podmínku na určující délku. Není zde rozdíl mezi blízkými geny a vzdálenými geny. Obě dvojice mohou patřit do stavebního bloku se stejnou pravděpodobností. Metodu křížení lze zvolit libovolně. Prohozením genů v levé části pole rank nezničíme žádné stavební bloky.

Na základě tohoto rozdělení jsem navrhl dva druhy křížení, které odpovídají těmto přístupům. Především proto, že do křížení bude vstupovat pouze levá část pole rank o velikosti columns_matched, jsem se rozhodl nevyužít existujících metod pro křížení permutací (PMX, order 1) a na místo toho se pokusil vytvořit modifikaci klasických metod SGA pro permutaci.

4.3.1 1-bodové křížení

Jak jsem již uvedl, do křížení bude vstupovat pouze levá část pole rank o velikosti columns_matched. Velikost této části se může u obou rodičů lišit. V levé části prvního rodiče určíme náhodně křížící bod a geny vlevo od křížícího bodu se zkopírují přímo na potomka. Poté se pokusíme doplnit co nejvíc dalších genů z levé části pole rank druhého rodiče a to tak, že budeme postupovat v pořadí pole rank. Počet genů tímto způsobem vzniklého potomka se bude pohybovat mezi columns_matched druhého rodiče a součtu pozice křížícího bodu s columns_matched druhého rodiče. Nemáme tedy zaručeno (a u velký matic to bude velice častý jev), že takto naplníme celé pole rank. Zbylé hodnoty doplníme zcela náhodně. Modifikace 1-bodového křížení pro Column-

Matching problém je na obrázku 4.3. Přerušovaná čára u potomka značí místo, kam až jsme dokázali doplnit páry od rodičů. Od přerušované čáry dále byly páry náhodně doplněny. Hodnota columns_matched potomka není okamžitě po křížení známá a určí se až během výpočtu fitnes.



Obr. 4.3 1-bodové křížení

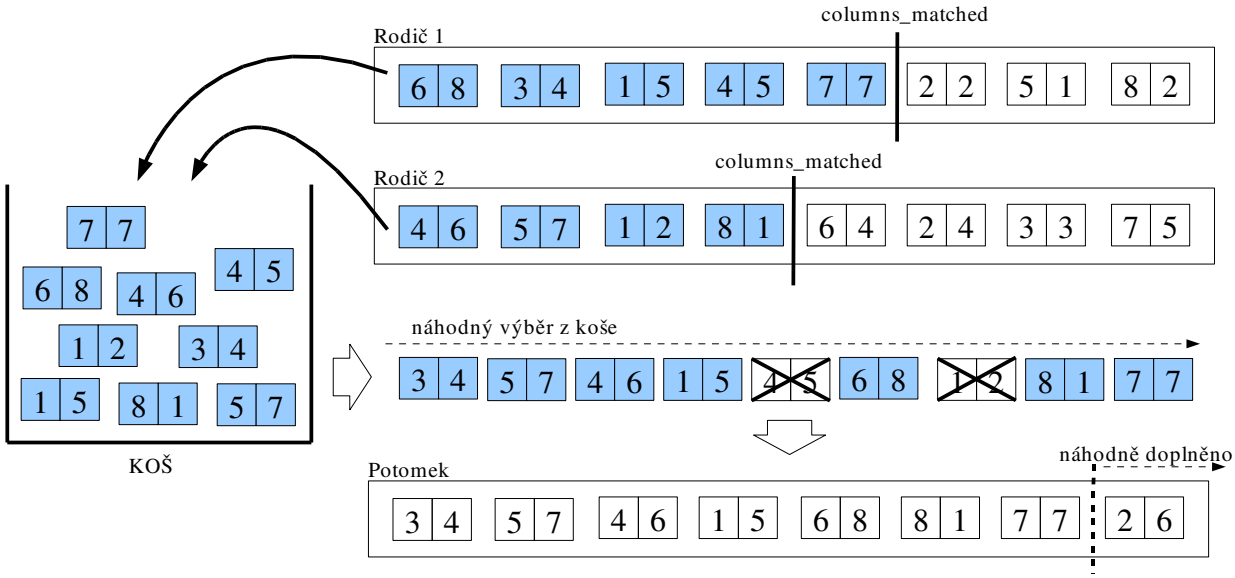
Očekávané chování

Cílem tohoto křížení je vyhovět hypotéze stavebních bloků. S přibývajícími generacemi by měl přibývat počet krátkých stavebních bloků s malou určující délkou. První z nevýhod je zakotvena v permutačních vlastnostech pole rank. Tím, že budeme od druhého rodiče postupně přidávat pouze ty geny (slouce matice T), které zatím nejsou v potomkovi, nedojde ke zkopírování souvislé části pole rank od druhého rodiče (některé geny se nepovede přidat – např. 3. gen druhého rodiče) a tím se sníží počet předaných stavebních bloků od druhého rodiče. Druhou nevýhodou tohoto křížení bude pravděpodobně velká populace z důvodu menšího počtu stavebních bloků obsažených ve jedinci. Tento důsledek bude více rozebrán v kapitole 4.8.

4.3.2 Uniformní křížení

Druhá metoda křížení neklade nároky na určující délku stavebních bloků (resp. nadprůměrných schémat). Pokusil jsem se jí navrhnout tak, aby všechny geny v levé části pole rank byly během křížení rovnocenné. Vycházím zde z myšlenky, že pokud máme kvalitního jedince s určitým počtem spárování, pak nezáleží na pořadí párů při vyhodnocování. Jinak řečeno, pokud by páry, které jsou součástí spárování byly vyhodnocovány v jiném pořadí, pak by jedinec měl stále stejné fitnes.

Průběh křížení je následující. Od obou rodičů vybereme geny z levé části pole rank do pozice columns_matched. Všechny tyto geny „naházíme“ do pomyslného koše. Potomka vytvoříme tak, že z koše postupně náhodně vybíráme geny a pokoušíme se je přidat do pole rank potomka. Gen nemůže být přidán v případě, že v poli rank již je jiný gen se stejným sloupcem matice T.



Obr. 4.4 Uniformní křížení

Očekávané chování

Od této metody očekávám, že se s přibývajícími generacemi bude v populaci zvyšovat počet schémat s nadprůměrnou fitnes. Nevýhodou této metody oproti 1-bodovému křížení je velký počet schémat v chromozómu (resp. v levé části pole rank) a s tím souvisící malý poměr mezi počtem schémat přenesených na potomka a celkovým počtem schémat v rodiči. Naopak výhodou je oproti výše uvedené metodě menší populace. Oběma zmíněným vlastnostem se budu podrobněji zabývat v kapitole 4.8.

4.4 Použité metody mutace

Mutace slouží v genetických algoritmech k tomu, aby byla populace postupem času přirozeně obměňována. V případě nedostatku mutace může dojít k situaci, kdy omezená skupina jedinců začne předčasně získávat v populaci převahu a v následku toho si začnou být vznikající jedinci z velké části podobní a vývoj fitnes se tím zastaví. V krajním případě může dojít k tomu, že mají všichni jedinci stejné geny. Tento jev se nazývá degenerace a pokud k němu dojde brzy, s největší pravděpodobností genetický algoritmus uvázne v lokáním extrému, tzn. dojde k předčasné konvergenci. Dostatečná mutace nám umožňuje se jí vyhnout.

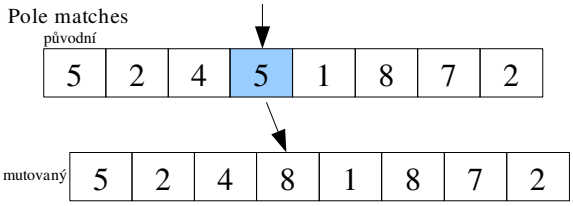
Existující metody mutace se, stejně tak jako tomu bylo u křížení, můžou rozdělit na klasické mutace (používané v SGA) a permutační mutace. Na degeneraci se můžeme koukat dvojím způsobem. Degenerovat může jak pole matches nebo NM, tak pole rank. Míru mutace můžeme ovlivnit parametry mutace p_M (pro pole matches) a p_R (pro pole rank), které obě určují s jakou pravděpodobností budou jednotlivé geny mutovány ($0 < p_M < 1, 0 < p_R < 1$).

V rámci této práce budou použity a testovány následující druhy mutace:

1) Náhodná mutace matches
2) Vložení do permutace
3) Prohození sousedů permutace

4.4.1 Náhodná mutace matches

Tato mutace je tou nejjednodušší modifikací klasické mutace ze SGA. Míra mutace je řízena parametrem p_M, který určuje pravděpodobnost mutace jednotlivého genu. Probíhá tak, že procházíme postupně pole matches od počátku až do konce a vždy, když chceme gen mutovat, nastavíme na této pozici hodnotu náhodného sloupce C.



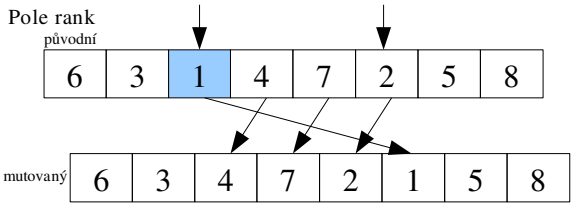
Obr. 4.5 Náhodná mutace matches

Očekávané chování

Díky dostatečné míře mutace pole matches se bude udržovat diversita tohoto pole a nebude docházek k předčasnému uváznutí v lokálním extrému.

4.4.2 Vložení do permutace

Míra mutace permutace p_{R1} má zamezit degeneraci pole rank, tudíž tomu, aby nedošlo k situaci, kdy většina jedinců má podobné pole rank. Tento parametr je nezávislý na míře mutace pole matches p_M, přičemž je mutace realizována na úrovni genu. Pro každé pole na pozici *i* se rozhoduje, jestli má být mutováno s pravděpodobností p_{R1}. V případě, že se pole má mutovat, vybere se náhodně jeho nová pozice *j*. Hodnota v poli *i* je zkopírována do pole *j*, hodnoty mezi pozicemi *i* a *j* jsou o jednu pozici posunuty směrem k *i*.



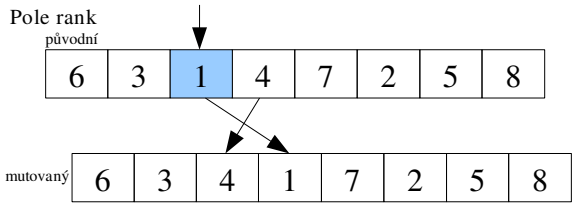
Obr. 4.6 Vložení do permutace

Očekávané chování

Tato forma mutace bude mít různý vliv na jednice v závislosti na použitém křížení. S použitím 1-bodového křížení bude tato mutace bourat řadu krátkých stavebních bloků, a tak nejlepších výsledků se pravděpodobně dosáhne s p_{R1} blížící se nule. Spolu s uniformním křížením by mohla udržovat diversitu pole rank, její míra však opět nesmí být přiliš vysoká, aby nedocházelo k příliš velkému promíchávání genů z levé a pravé části pole rank.

4.4.3 Prohození sousedů permutace

Tato mutace nemění pole rank tolik jako předchozí metoda, a proto nemá takový vliv na výsledné fitnes. Při použití této metody tedy bude nutné mít o to větší míru mutace, aby nedošlo k degeneraci. Pro každé pole rank na pozici *i* vyjma posledního se rozhoduje, jestli má být mutováno s pravděpodobností p_{R2}. V případě, že se pole má mutovat, prohodí se hodnoty v polích *i* a *i*+1.



Obr. 4.7 Prohození sousedů permutace

Očekávané chování

Tato mutace má sloužit jako alternativa příliš agresivní mutace 4.4.2 pro 1-bodové křížení. Spojení této metody s uniformním křížením nebude mít pravděpodobně velký význam.

4.5 Selekce

Posledním operátorem používaným v genetických algoritmech je operátor selekce, který rozhoduje o tom, jakým způsobem jsou vybíráni jedinci do nové populace. Důležitým pojmem, mluvíme-li o selekci, je *selekční tlak*, který určuje do jaké míry jsou zvýhodňováni kvalitnější jedinci, tzn. jedinci s větším fitnes, před méně kvalitními. Míru selekčního tlaku budeme značit jako selekční parametr p_S. Operátor selekce nijak nesouvisí s použitým kódováním, a tak může být použita jakákoliv selekční metoda zmíněná v kapitole 3.1 SGA. Pro naše účely jsem se rozhodl implementovat selekční metodu turnaj.

4.5.1 Turnaj

Selekční metodu turnaj jsem vybral z důvodu, že má strmější pravděpodobnostní funkci u lepších jedinců než u jedinců horších. Díky tomu je možné, narozdíl od lineární selekce, výraznějším způsobem prosazovat kvalitnější jedince s větším fitnes. V této vlastnosti se skrývá i velké nebezpečí, protože s větším prosazováním kvalitních jedinců vzrůstá i možnost předčasné konvergence k lokálnímu optimu. Míra selekčního tlaku je jednoduše regulovatelná a určuje jí velikost turnaje. Selekční parametr je počet účastníků, kteří se turnaje účastní ($2 \leq p_S \leq velikost\ populace$, většinou je však $p_S \ll velikost\ populace$). Nejmenšího selekčního tlaku dosáhneme s velikostí turnaje 2. V některých problémech řešených genetickými algoritmy může být i tento selekční tlak příliš velký, nicméně v našem případě již předběžné testy ukázaly, že se velikost turnaje bude pohybovat kolem 3.

4.6 Výpočet fitnes

Výpočet fitnes slouží k ohodnocení kvality jedince, resp. kvality řešení. V našem případě bude platit, že čím větší je fitnes, tím lepší je řešení. Naším cílem je navrhnutí výstupního dekodéru tak, aby byla co nejjednodušší realizace celého generátoru testovacích vzorků (TPG), tedy snažíme se minimalizovat nejen logiku výstupního dekodéru, ale i přepínací logiku (viz. kap. 1). Nejjednodušší implementace spárování nastává, pokud sloupec matice C se rovná sloupci matice T. Takové spárování nazýváme přímé a pro realizaci dekodéru není zapotřebí žádných obvodů, pouze

v případě negativního párování je nutné použít jednoho obvodu XOR. Pokud sloupec matice C není roven sloupci matice T, jedná se o nepřímé propojení a do obvodu bude přidán navíc jeden multiplexer. Výstupní proménné (sloupce matice T) nepatřící do žádného spárování budou vypočítány na konci algoritmu pomocí optimalizátoru Booleových funkcí (např. BOOM, viz. [Fis2], [BOOM]), tedy nejsou během výpočtu fitnes známy, nicméně předpokládám, že jejich implementace bude značně složitější než sloupců součástí spárování. Proto bude hlavním optimalizačním kritériem počet spárování a druhým vedlejším optimalizačním kritériem bude počet přímých spárování, tedy:

$$fitnes = počet\ spárování + \frac{počet\ přímých\ spárování}{šířka\ matice\ T}$$

Maximální fitnes, kterého jedinec může dosáhnout, je šírka matice T + 1.

Kódování jedince je navržené tak, aby výpočet fitnes co nejvíce zjednodušilo. Pole matches, NM a rank jsou v základní podobě neměnné, jedinné co se vypočítává je hodnota columns_matched. Jestliže columns_matched = *i*, tzn. počet spárovaných sloupců matice T je *i*, pak prvních *i* hodnot pole rank identifikuje sloupce T, které se povedlo spárovat. Algoritmus výpočtu fitnes pracuje podobně jako Fast Search u náhodně řízeného Column-Matching algoritmu pouze s tím rozdílem, že vybírané sloupce matic T a C pro párování nejsou voleny náhodně. Pořadí sloupců matice T je určeno polem rank a k nim příslušící sloupce matice C jsou uloženy v poli matches. Pole NM značí, jestli se má pro konkrétní sloupec T provést negativní párování. Proměnná columns_matched je na počátku nastavena na 0 a s každým povedeným spárováním je inkrementována. Jakmile se nepovede spárovat dvojici sloupců algoritmus končí. Fitnes jedince se nastaví na aktuální hodnotu columns_matched.

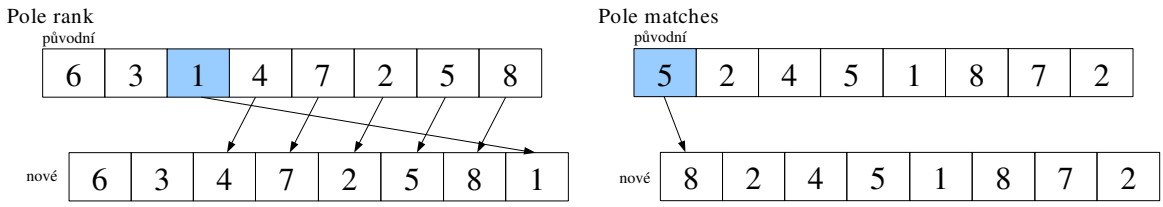
Výpočet fitness v základní podobě může být popsán následujícím pseudokódem. Pseudokód pro přehlednost neuvažuje pole NM a negativní párování.

```
Jedinec::CountFitness(C, T) {
   // inicializujeme matici B
   for (k = 0; k < C_matrix_rows; k++)
      for (l = 0; l < T_matix_rows; l++)
         B[k, l] = “1“;
   A = NULL;
   columns_matched = 0; direct_matches = 0;
   do {
      // podle priority postupne vybirame sloupce T a k nim prislusici sloupce C
      i = rank[columns_matched];
      j = matches[i];
      for (k = 0; k < C_martrix_rows; k++)
         // modify blocking matrix
         for (l = 0; l < T_martrix_rows; l++)
            if (T[l, j] != DC && C[k, i] != T[l, j])
               B[k,l] = “0“;
      A' = A;                        // provedeme zalohu prirazeni
      A = MakeRowAssignment(B);
      if (A != FAILED) {
         columns_matched++;
         direct_matches++;
      }
   } while (A != FAILED && columns_matched < T_matrix_cols);
   fitnes = columns_matched + (direct_matches / T_matrix_rows);
}
```

4.6.1 Zahazování sloupců

Výše popsaná metoda výpočtu fitnes je založena na algoritmu Fast Search. Jestliže bychom chtěli při výpočtu fitnes dosahovat většího počtu spárovaných sloupců, mohli bychom povolit během výpočtu modifikovat jedince, ať už pole matches (NM) nebo pole rank. Jednou z možností je využití algoritmu Thorough Search, který by ovšem vedl k nežádanému prodloužení doby výpočtu fitnes. Proto jsem navrhl metodu zahazování sloupců, která je kompromisem mezi Fast Search a Thorough Search.

Zahazování sloupců se odlišuje následujícím způsobem. Před každým pokusem o spárování sloupců si vytváříme záložní kopii matice B – matici B'. Pokud narazíme na sloupec *i* matice T na pozici *r* v poli rank, který se nepovede spárovat se sloupcem *j* matice C, provedeme modifikaci jedince, kterou nazývám zahození sloupce. Ta spočívá v odsunutí hodnoty *i* na konec pole rank, hodnoty mezi *r* a koncem pole rank jsou posunuty o jedno pole vpřed. Navíc, protože pár (*i* , *j*) nebude platný ani na konci pole rank, nastavíme do matches[*i*] náhodný sloupec matice C (původně platilo matches[*i*] = *j*). Na Obr. 4.8 je ukázáno zahození 1. sloupce T, protože se nepovedl spárovat pár (1, 5) (r = 3, i = 1, j = 5). Po zahození sloupce obnovíme matici B podle záložní matice B' a pokusíme se spárovat sloupec T, který se zahozením sloupce *i* dostal v poli rank na místo *r*.



Obr. 4.8 Zahození sloupce

Výpočet fitnes končí v momentě, kdy se nepovede spárovat sloupec T, který již byl jednou zahozen, tj. když počet úspěšně spárovaných sloupců + počet zahozených sloupců > velikost pole rank (šířka matice T). Metodu zahazování sloupců při výpočtu fitnes popisuje následující pseudokód:

```
Jedinec::CountFitnessDiscard(C, T) {
   // inicializujeme matici B
   for (k = 0; k < C_matrix_rows; k++)
      for (l = 0; l < T_matix_rows; l++)
         B[k, l] = “1“;
   A = NULL;
   columns_matched = 0; direct_matches = 0;
   columns_discarded = 0;
   do {
      // podle priority postupne vybirame sloupce T a k nim prislusici sloupce C
      i = rank[columns_matched];
      j = matches[i];
      B' = B;
      for (k = 0; k < C_martrix_rows; k++)
         // modify blocking matrix
         for (l = 0; l < T_martrix_rows; l++)
            if (T[l, j] != DC && C[k, i] != T[l, j])
               B[k,l] = “0“;
      A' = A;                          // provedeme zalohu prirazeni
      A = MakeRowAssignment(B);
      if (A != FAILED) {
         columns_matched++;
         direct_matches++;
```

```
      } else {
         DiscardColumn(columns_matched);     // zahozeni sloupce – uprava rank
         columns_discarded++;
         B = B';
      }
   } while ( (A != FAILED && columns_matched < T_matrix_cols) ||
                     columns_matched + columns_discarded< T_matrix_cols);
   fitnes = columns_matched + (direct_matches / T_matrix_rows);
}
```

V nejhorším případě může dojít k zahození všech sloupců matice T a následně k úspěšnému spárování všech již jednou zahozených sloupců. Proto počet vnitřních cyklů může být dvakrát větší než je nejhorší případ Fast Search. Asymptotická složitost tohoto algoritmu je však stejná jako asymptotická složitost Fast Search, tj. $O(r \cdot p \cdot s)$.

4.7 Reprodukční cyklus

Generování nové populace probíhá následujícím způsobem. V kódu je použit parametr křížení p_C, parametr selekce p_S, dva parametry pro mutaci pole rank p_{R1} a p_{R2} (pole rank) a parametr pro mutace matches p_M.

```
Populace::GenerateNext() {
   vector<Jedinec> nova_generace;
   for (i = 0; i < population_size; i++) {
      // vybereme jednoho jedince z populace
      prvni = SelectIndividual(ps);
      if ( random(0, 1) < pc ) {
         // pokud chceme jedince krizit, musime vybrat jeste jednoho
         druhy = SelectIndividual(ps);
         // provedeme krizeni
         novy = Crossover(prvni, druhy);
      } else {
         novy = prvni;
      }
      // provedeme mutaci s pravdepodobnostmi pm, pr1 a pr2
      novy.MutationMatches(pm);
      novy.MutationRank1(pr1);
      novy.MutationRank2(pr2);
      novy.CountFitness();                  // vypocteme fitnes
      nova_generace.push(novy);             // vlozime jedince do nove generace
   }
   generace = nova_generace;                // nahradime puvodni generaci novou
}
```

Na počátku je algoritmu definován také maximální počet generací *max generations* , po kterém dojde k ukončení reprodukčního cyklu. Druhou podmínkou ukončení běhu je parametr p_T . Přerušení reprodukce nastane pokud po p_T generací nedojde ke zlepšení nejlepší průměrné fitnes populace. Poslední podmínkou ukončení reprodukčního cyklu je nalezení zjevně optimálního řešení, tzn. jedince s maximálním možným fitnes = šířka matice T + 1. K této situaci však pravděpodobně nikdy nedojde.

Na konci algoritmu probíhá minimalizace Booleových funkcí u nespárovaných sloupců matice T a výpočet výsledné Booleové funkce výstupních proměnných dekodéru. Minimalizací nespárovaných sloupců se tato práce nezabývá. Může být provedena například pomocí programu BOOM, viz. [BOOM], [Fis2]. Výstupem genetických algoritmů bude soubor obsahující spárované sloupce nejlepšího řešení a neminimalizovaná funkce nespárovaných proměnných uložená v souboru typu PLA, který může být vstupem programu BOOM.
Navržený genetický algoritmus pro Column-Matching problém má následující strukturu:

```
GeneticColumnMatching(C, T) {
   Best = NULL;
   generation = 0;
   max_avg = 0;
   counter = 0;
   // vytvorime a ohodnotime prvni generaci slozenou z nahodnych jedincu
   populace.Init();
   do {
      // vytvorime dalsi generaci
      populace.GenerateNext();
      generation++;

      // zapamatujeme si nejlepsiho jedince
      if (Best.GetFitness()<=populace.BestIndividual().GetFitness())
         A=populace.BestIndividual();

      // podminky ukonceni reprodukcniho cyklu
      if (generation >= max_generations) break;
      if (populace.GetAverageFitness() < max_avg) counter++;
      else {
         max_avg = populace.GetAverageFitness();
         counter = 0;
      }
      if (max_avg >= pt) break;
      if (Best .GetFitness() == T_matrix_cols + 1) break;

   } while (1 == 1)
   Substititute_DCs(T);              // nahrazeni DC hodnot pomoci 0 nebo 1
   CompactTest(T);                   // provedeme test kompaktnosti
   ExtractMatches(C, T);             // odstranime sparovane sloupce
   // vytvorime celkovou vystupni funkci, i pro zbyle promenne
   F = Minimize( Best.RowAssingment() );     // program BOOM
   return F;
}
```

4.8 Vlastnosti populace

Z metod křížení popsaných v kapitole 4.3 vyplývají dvě základní vlastnosti genetického algoritmu, které je důležité zmínit, a to poměr schémat přenesených z rodiče na potomka a velikost populace. Vzorce, které v této kapitole uvedu nebudou dále využívány (ani během testování ani během měření závěrečných výsledků), nicméně můžou sloužit k získání představy o složitosti řešeného problému a vlastnostech genetických operací.

Pro jednoduchost budeme v následujícím textu pracovat pouze se schématy jednotného řádu (resp. stavebními bloky jednotné délky) o , nicméně si musíme uvědomit, že ve skutečnosti se v chromozomech vyskytují schémata všech řádů. Chromozómem zde myslím pouze levou část pole rank do pozice columns_matched. Máme-li chromozóm délky *l*, pak počet všech schémat řádu *o* ($o<l$) v chromozómu je

$$S_o(l)=V_o(l)=\frac{l!}{(l-o)!}$$

Klademe-li ovšem omezení na určující délku schématu d (platí $o\le d+1\le l$), pak počet takovýchto schémat v chromozómu bude pouze

$$S_o(l,d)=(l-d)\cdot V_o(d+1)=(l-d)\cdot\frac{(d+1)!}{(d+1-o)!} \qquad (4.1)$$

Přejděme nyní na výpočet počtu schémat přenesených z prvního rodiče na potomka při 1-bodovém křížení. Pozice křížícího bodu bude v průměru rovna $l/2$, a tak s ní budu také počítat. Počet schémat řádu *o* a určující délkou *d*, které se přenesou na potomka je

$$P_o(l,d)=S_o(l/2,d)$$

Poměr mezi počtem schémat přenesených z prvního rodiče na potomka a celkovým počtem schémat prvního rodiče při 1-bodovém křížení je

$$\frac{P_o(l,d)}{S_o(l,d)} = \frac{S_o(l/2,d)}{S_o(l,d)} = \frac{l/2-d}{l-d}$$

Při výpočtu poměru schémat přenesených na potomka u uniformního křížení nebereme v potaz určující délku schémat. Počet genů od jednoho rodiče bude v průměru $l/2$. Počet schémat řádu o přenesených na potomka je

$$P_o(l) = S_o(l/2)$$

Poměr mezi počtem schémat přenesených z rodiče na potomka a celkovým počtem schémat rodiče při uniformním křížení je

$$\frac{P_o(l)}{S_o(l)} = \frac{S_o(l/2)}{S_o(l)} = \frac{(l/2)!\cdot(l-o)!}{l!\cdot(l/2-o)!}$$

Vezměme například: $l = 20$, $d = 5$, $o = 3$. Poměr přenesených schémat z rodiče na potomka je u 1-bodového křížení 0,333 a u uniformního křížení 0,105.

Určení vhodné velikosti populace také není jednoduchý úkol. Bohužel existující teorie většinou počítají s kardinalitou problému 2 (chromozóm je binární pole). Pokusil jsem se tedy odvodit velikost populace tak, aby v počáteční populaci bylo zastoupení většiny schémat konkrétního nízkého řádu *o*. Následující úvaha počítá s tím, že použijeme uniformní křížení. Počet všech schémat řádu *o* v Column Matching problému je

$$T_o = V_o(r)\cdot V_o(n)\cdot 2^o = \frac{r!}{(r-o)!}\cdot\frac{n!}{(n-o)!}\cdot 2^o$$

Velikost populace, kde by každé schéma řádu *o* bylo zastoupeno právě jednou je rovno podílu počtu všech schémat v populaci a počtu schémat v jednom chromozomu, tedy:

$$\frac{T_o}{S_o(r)} = \frac{V_o(r)\cdot V_o(n)\cdot 2^o}{V_o(r)} = \frac{n!}{(n-o)!}\cdot 2^o$$

Reálnějšího počtu dosáhneme, pokud uvážíme, že do křížení vstupuje pouze levá část chromozómu délky *l*. Pak bude velikost populace rovna

$$\frac{T_o}{S_o(l)} = \frac{V_o(r)\cdot V_o(n)\cdot 2^o}{V_o(l)} = \frac{r!}{(r-o)!}\cdot\frac{n!}{(n-o)!}\cdot\frac{(l-o)!}{l!}\cdot 2^o \qquad (4.2)$$

, kde *l* je rovno počtu spárování průměrného náhodného jedince.

Výše zmíněné vzorce shledávají rozdíl mezi schematy např. ((2, 1), (3, 2)) a ((3, 2), (2, 1)), protože počítají s variacemi. Pokud ovšem používáme uniformní křížení (4.4.2), kde nezáleží na pořadí genů v levé části pole rank, pak jsou tyto schémata rovnocenná. Z toho vyplývá, že velikost populace může být při použití unifomního křížení zmenšena $o!$ -krát.

Tabulka 4.2 popisuje velikosti populací různých matic zmenšené o $o!$.

matice	r	n	l	$\frac{T_2}{S_2(r)\cdot 2!}$	$\frac{T_2}{S_2(l)\cdot 2!}$	$\frac{T_3}{S_3(r)\cdot 3!}$	$\frac{T_3}{S_3(l)\cdot 3!}$
s820	23	23	16	1 012	1 673	14 168	44 806
c1908	33	33	17	2 112	8 200	43 648	350 211
s838_1	67	67	17	8 844	143 780	383 240	26 998 694
s9234.1_1	247	247	103	121 524	702 840	19 848 920	278 468 616
s13207.1_3	700	700	460	978 600	2 267 827	455 375 200	1 608 288 728

Tabulka 4.2 Závislost velikosti populace na vstupním zadání

Vygenerujeme-li počáteční náhodnou populaci o velikosti podle vzorce 4.2, je zřejmé, že v této populaci nebudou všechna schémata řádu o zastoupena. Nicméně zbylá schémata můžou vznikat během následujících mutací. S dostatečnou mírou mutace by mohla být takováto velikost populace postačující. Pro $o=2$, pokud l je dostatečně velké ($l>r/2$) a $r\approx n$ (většina matic toto splňuje), pak nárust velikosti populace s šírkou matice T r je $O(r^2)$. Pokud je však $l\ll r$, pak nárust velikosti populace s šířkou matice T r může stoupnout až na $O(r^4)$.

Z tabulky 4.2 je zřejmé, že u velkých problémů nemůžeme splnit požadavek ani na existenci většiny stavebních bloků řádu 2. Populace by byla tak velká, že by genetické algoritmy nemohly proběhnout v únosném čase. Při volbě velikosti populace tedy musíme volit kompromisní řešení, které dovolí genetickým algoritmům spočítat výsledek v únosném čase, a přitom bude toto řešení dostatečně kvalitní.

S použitím 1-bodového křížení je situace ještě složitější. Je tomu tak především proto, že během tohoto křížení slouží jako nosné informace schémata s nízkou určující délkou (resp. stavební bloky). Těch je v jedinci mnohem méně (viz. vzorec 4.1) než celkového počtu schémat a z toho důvodu by měla být větší i populace. Nicméně stejně tak jako v případě uniformního křížení budeme během měření muset hledat kompromis mezi dostatečně kvalitním řešením, jehož kvalitu určí pravděpodobně především velikost počáteční populace a délkou běhu genetického algoritmu.

4.9 Provedené testy

Na začátku testů jsem si stanovil podmínku, že by doba běhu všech testů neměla překročit 3 hodiny. Přitom jsem se snažil, aby program běžel mezi 1-2 hodinami. V tomto směru se ukázalo být zásadní nastavení velikosti populace, přestože ostatní parametry měly na dobu běhu také vliv.

V rámci testování jsem provedl následující měření. Po několika experimentálních spuštění byl realizován úvodní test, jehož cílem bylo zjistit přibližné výsledky a chování GCMA1. Test byl proveden na deseti maticích. V druhém měření jsem se zaměřil na vliv metody zahazování sloupců. Cílem dalších testů bylo upřesnit nastavení jednotlivých parametrů. Protože testy jsou velice časově náročné, vybral jsem pro tyto testy pouze 3 matice, které vykazují větší výkyvy ve výsledcích a tedy se zdály být pro tyto testy vhodné. Testy parametrů si kladly za cíl především zjistit, jaký vliv má ten který parametr na dosažené výsledky a rámcově určit intervaly, ve kterých by se měly hodnoty jednotlivých parametrů pohybovat. Celkové výsledky jsou shrnuty v kapitole 4.10.

V testech byly použity testovací obvody ISCAS'85 [Brg85] a '89 [Brg89]. Testovací matice

poskytl Ing. Petr Fišer, Ph.D. Měření bylo provedeno na testovacím prostředí s procesorem Intel Core 2 2,80 Ghz a pamětí 1 GB.

4.9.1 Existující Column-Matching algoritmy

Protože součástí této práce je i porovnání s existujícími metodami, uvedu nejdříve výsledky, kterých dosahují náhodně řízené Column-Matching algoritmy Fast Search a Thorough Search, viz. [Fis1], a jsou rámcově popsány v kapitole 3. Pro každou matici jsem se pokusil dosáhnout co nejlepšího výsledku během jedné hodiny, a to jak pro Fast Search, tak pro Thorough Search. Tyto výsledky budou sloužit jako orientační. Přesnější srovnání s algoritmem Thorough Search bude provedeno v kapitole 4.10, resp. 5.8.

Dosažené výsledky jsou uvedeny v tabulce 4.3, v závorce uvádím dosažený počet přímých spárování. U matice s38417_1 se hodnoty s aktuální verzí programu ColMatch, který implementuje oba zmiňované algoritmy, nepovedlo naměřit.

název matice	matice C	matice T	Fast Search (1 běh)	Fast Search (1 hodina)	Thorough Search (1 běh)	Thorough Search (1 hodina)
s820	23 x 1000	23 x 25	7 (0)	20 (2)	20 (17)	22 (17)
c1908	33 x 1000	33 x 28	9 (0)	14 (0)	20 (9)	25 (9)
s420.1_1	34 x 500	34 x 42	10 (1)	20 (1)	23 (15)	29 (17)
s420.1_4	34 x 1000	34 x 34	8 (0)	24 (1)	26 (16)	30 (19)
s641_1	54 x 500	54 x 13	13 (0)	39 (0)	53 (34)	54 (38)
s838_1	67 x 1000	67 x 61	4 (0)	17 (0)	20 (12)	45 (16)
s5378_3	214 x 1000	214 x 19	42 (0)	166 (0)	212 (190)	214 (197)
s9234.1_1	247 x 1000	247 x 215	14 (0)	42 (0)	153 (77)	160 (62)
s13207.1_3	700 x 1000	700 x 197	69 (0)	166 (0)	599 (199)	624 (264)
s38417_1	1664 x 1000	1664 x 105	nelze změřit	nelze změřit	nelze změřit	nelze změřit

Tabulka 4.3 Naměřené hodnoty náhodně řízeného Column-Matching algoritmu

4.9.2 Úvodní test genetických algoritmů

Genetické algoritmy jsou typické velkým množstvím vstupních parametrů, které se liší v závislosti na vstupním zadání. Pro SGA existují experimentálně ověřené rozsahy, ve kterých by se měly hodnoty parametrů přibližně pohybovat. Nicméně Column-Matching problém je z pohledu genetických algoritmů zatím neprozkoumaný a přitom navržené kódování a genetické operátory jsou velice specifické. Nalezení optimáního nastavení parametrů bývá u takovýchto problémů často velice obtížné a vyžaduje řadu experimetů.

V testech byly použity testovací obvody ISCAS'85 [Brg85] a '89 [Brg89], testovací matice poskytl Ing. Petr Fišer, Ph.D. Během prováděných experimentů bylo nutné přistupovat různě k maticím s různě složitým výpočtem fitnes. Abychom dosáhli výsledků v rozumném čase, musel jsem u matic s nejsložitějším výpočtem fitnes výrazně zmenšovat populaci. To mě vedlo k rozdělení matic na malé, střední, velké a obrovské, které určuje doba výpočtu fitnes. Rozdělení popisuje tabulka 4.4. Při použití metody zahazování sloupců, která se během testů ukázala být nezbytná,

přesahovala doba výpočtu fitnes u nejsložitějších testovaných matic 6 sekund.

třída matice	výpočet fitnes (s)	matice
malé	< 0,1	s820, c1908, s420.1_1, s420.1_4, s641_1
střední	< 1	s838_1, s5378_3
velké	< 6	s9234.1_1
obrovské	> 6	s13207.1_3, s38417_1

Tabulka 4.4 Rozdělení vstupních matic podle velikosti

Sloupec výpočet fitnes značí dobu výpočtu fitnes jednoho jedince na jednotném testovacím prostředí.

Během všech měření jsem se snažil o dosažení výsledku přibližně během 1 – 2 hodin. Pokud výpočet překročil 3 hodiny, pak byl přerušen. Hlavním parametrem, který ovlivňuje délku běhu algoritmu je velikost populace, která tomu byla přizpůsobena. S časovým omezením je zřejmé, že především u velkých matic nebudou výsledná řešení globálním optimem. Z kapitoly 4.8 vyplývá, že pokud budeme klást požadavek existence většiny stavebních bloků délky 2 v počáteční populaci, měla by populace s šírkou matice T minimálně kvadraticky růst. Při tomto testu tedy jdeme proti těmto závěrům, protože velikost populace s narůstající složitostí problému naopak snižujeme.

parametr	1-bodové křížení	uniformní křížení
velikost populace	20 - 1000	20 - 1000
časové omezení (v minutách)	180	180
podmínka ukončení běhu p_T	5	5
parametr selekce p_S	2	2
pravděpodobnost křížení p_C	0,7	0,7
parametr mutace matches p_M	0,01	0,01
parametr mutace p_{R1} (vložení do permutace)	0	0,01
parametr mutace p_{R2} (prohození sousedů permutace)	0,1	0
zahazování sloupců	ano	ano

Tabulka 4.5 Parametry úvodního testu GCMA1

Snížení je však pochopitelné, protože se zvětšováním vstupních matic dochází k prodloužení výpočtu fitnes $O(r \cdot p \cdot s)$. Ukončení běhu reprodukčního cyklu nastane, jakmile po 5 generací nedojde k vylepšení nejlepšího průměrného fitnes populace. Testy jsou provedeny jak pro 1-bodové, tak pro uniformní křížení. Z předchozích kapitol vyplývá, že s použitím různé metody křížení se liší i některé vstupní parametry. Tabulka 4.5 popisuje vstupní proměnné pro oba druhy křížení, které byly přibližně experimentálně odvozeny z předběžných testů na maticích s820 a s838_1 a budou použity pro úvodní test.

Naměřené hodnoty úvodního testu jsou v tabulce 4.6. Ve sloupcích „GA1 (1-pnt)“ a „GA1 (uni)“ jsou naměřené celkové počty spárování pro 1-bodové a uniformní křížení algoritmu

GCMA1. V závorce je zde uveden celkový počet přímých spárování. Sloupec „pop.“ značí velikost populace, sloupec „gen.“ počet generací. Ve sloupci „Thorough S.“ jsou výsledky, kterých dosáhl algoritmus Thorough Search běžící po dobu jedné hodiny. Posledně zmíněný sloupec je pouze informační, přesné porovnání GCMA1 s algoritmem Thorough Search a náhodným generátorem bude provedeno v kapitole 4.10. Tučně jsou označeny nejlepší naměřené výsledky GCMA1.

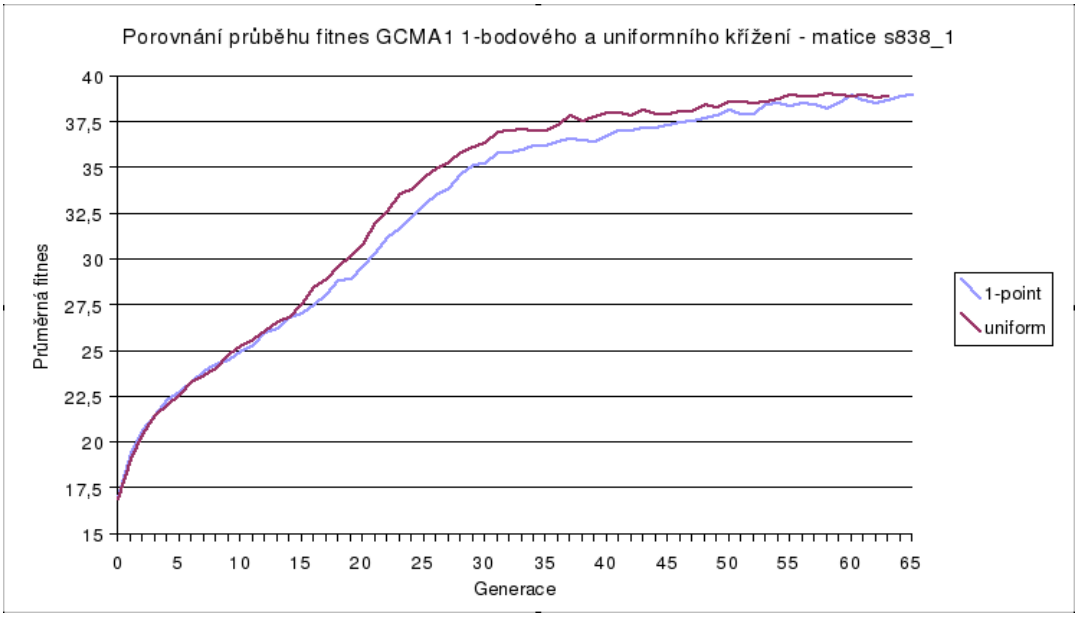
název matice	GA1(1-pnt)	pop.	gen.	čas (s)	GA1(uni)	pop.	gen.	čas (s)	Thorough S.
s820	**21 (20)**	1000	108	4520	21 (19)	1000	100	4211	22 (17)
c1908	21 (9)	1000	81	4526	**22 (4)**	1000	92	5122	25 (9)
s420.1_1	29 (14)	1000	152	8565	**29 (17)**	1000	104	5092	29 (17)
s420.1_4	**30 (15)**	1000	102	8335	29 (22)	1000	96	7231	30 (19)
s641_1	**54 (35)**	1000	105	2791	54 (26)	1000	116	2751	54 (38)
s838_1	43 (5)	250	65	4664	**44 (1)**	250	63	4322	45 (16)
s5378_3	207 (4)	500	20	3631	**209 (3)**	1000	25	7404	214 (197)
s9234.1_1	**148 (1)**	35	63	9806	137 (0)	45	31	5913	160 (62)
s13207.1_3	**568 (1)**	20	46	9941	534 (1)	100	7	7354	624 (264)
s38417_1	**887 (0)**	20	30	8459	863 (1)	100	7	7400	nelze

Tabulka 4.6 Výsledky úvodního testu GCMA1

Výsledek testu

Při srovnání 1-bodového křížení s uniformním můžeme říci, že u velkých a obrovských matic dosahovalo lepších výsledků křížení 1-bodové, zatímco u menších a středních matic si oba druhy křížení vedly přibližně stejně.

Graf 4.1 ukazuje vývoj průměrné fitnes při použití 1-bodového a uniformního křížení, které si jsou velmi podobné. Průměrná fitnes počáteční generace je přibližně 17. Vývoj populace poté postupně vcelku plynule konverguje k optimu a zastavuje se přibližně na úrovni průměrné fitnes 38.



Graf 4.1 Porovnání průběhu fitnes GCMA1 1-bodového a uniformního křížení – matice s838_1

Z výpisu poslední populace je evidentní, že algoritmus zkonvergoval k lokálnímu optimu, tzn. u všech jedinců se v levé části pole rank vyskytují stejné geny (pár sloupce T, sloupce C a příznak negování). Vzhledem k tomu, že tento výpis je vcelku rozsáhlý, je k dispozici pouze na přiloženém CD, viz. příloha C. Že se jedná o lokální optimum je zřejmé už proto, že algoritmus Thorough Search dosáhl lepšího výsledku. Naším cílem v dalších testech, kdy budeme hledat vhodné hodnoty parametrů, bude přimět genetický algoritmus, aby konvergoval k lepším řešením. Předtím však ještě zjistíme, jaký má vliv na dosažené výsledky metoda zahazování sloupců.

4.9.3 Vliv metody zahazování slopců

Cílem tohoto testu bylo spustit genetický algoritmus bez metody zahazování sloupců a tím zjistit, jaký vliv na dosažené výsledky tato metoda má. Protože metoda zahazování sloupců způsobuje prodloužení doby výpočtu fitnes, byla v tomto testu navýšena velikost populace tak, aby bylo opět dosaženo výsledků během 1-2 hodin. Ostatní vstupní parametry se oproti předchozímu testu nemění. Naměřené hodnoty jsou uvedeny v tabulce 4.7. Ve sloupci „max $GA1_1$“ jsou nejlepší naměřené hodnoty pro jednotlivé matice během úvodního testu.

název matice	GA1 (1-pnt)	pop.	gen.	čas (s)	GA1 (uni)	pop.	gen.	čas (s)	max $GA1_1$	T. Search
s820	20 (2)	3000	27	2155	20 (2)	3000	26	2008	21 (20)	22 (17)
c1908	18 (12)	2000	95	5984	18 (12)	2000	110	5972	22 (4)	25 (9)
s420.1_1	28 (12)	1500	179	9061	24 (6)	1500	141	6269	29 (17)	29 (17)
s420.1_4	25 (0)	3000	22	2851	24 (2)	3000	21	2527	30 (15)	30 (19)
s641_1	54 (4)	5000	124	10847	42 (5)	5000	35	2408	54 (35)	54 (38)
s838_1	34 (1)	1000	103	10826	32 (5)	1000	99	9319	44 (1)	45 (16)
s5378_3	197 (1)	500	52	5319	192 (1)	500	30	2699	209 (3)	214 (197)
s9234.1_1	92 (0)	140	39	6476	97 (0)	140	38	5665	148 (1)	160 (62)
s13207.1_3	269 (1)	50	45	8007	234 (1)	50	28	4495	568 (1)	624 (264)
s38417_1	284 (0)	50	83	5994	177 (0)	50	49	2098	887 (0)	nelze

Tabulka 4.7 Výsledky testu vlivu metody zahazování sloupců GCMA1

Výsledek testu

Bez použití metody zahazování sloupců byly výsledky genetického algoritmu o poznání horší, a to především u větších matic. Absence zahazování sloupců měla větší vliv na uniformní křížení. Špatné výsledky mohlo ovlivnit i to, že nebyly provedeny prakticky žádné experimenty s nastavením jednotlivých parametrů, nicméně i tak se zdá být zřejmé, že se bez této metody neobejdeme.

4.9.4 Test selekčního tlaku

Další testy se zaměří na nalezení vhodných parametrů genetického algoritmu. Testy jsou provedeny na maticích c1908, s838_1, s9234.1_1. Tyto vstupní matice se zdály být vhodné, protože projevovaly nestálé výsledky. Testy jsem se rozhodl provést tak, že se postupně pokusím nalézt optimální nastavení parametrů p_S, p_C a p_M s tím, že výsledky předchozích měření budou použity do měření následujících. Např. tento test tedy nalezne optimální p_S, který dále použijeme při hledání

optimálního p_C a p_M. U všech testů uvedu naměřené výsledky. Tam, kde je možné odvodit najéké závěry z průběhu fitnes ukážu i grafické porovnání průběhů fitnes.

První z testů parametrů se zaměřil na nalezení vhodného selekčního tlaku, resp. parametru selekce p_S. Použitá selekční metoda je ve všech případech turnaj, jehož velikost byla zvyšována od 2 do 5. Naměřené hodnoty uvádí tabulka 4.8. Nejlepší výsledky pro jednotlivé matice a druh křížení jsou označny tučně. Prvním kritériem pro porovnání výsledků je celkový počet spárovaných sloupců, druhý počet přímých spárování, a pokud i oba tyto počty jsou shodné, pak rozhoduje doba běhu algoritmu.

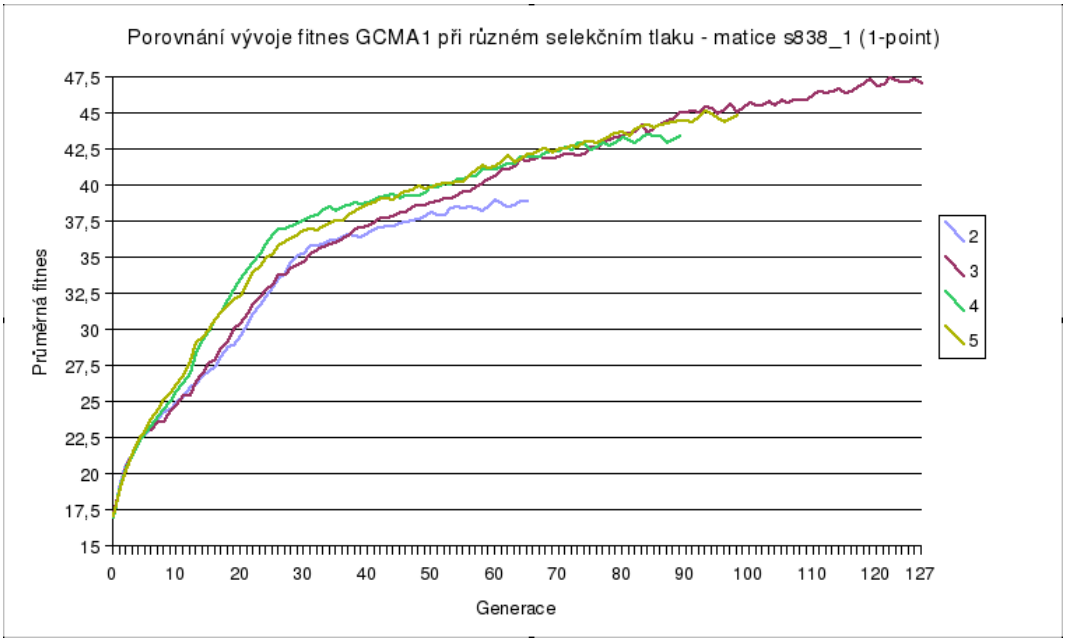
název matice	p_S	GA1 (1-pnt)	pop.	gen.	čas (s)	GA1 (uni)	pop.	gen.	čas (s)
c1908	2	21 (9)	1000	81	4526	22 (4)	1000	92	5122
c1908	3	**22 (5)**	**1000**	**76**	**4569**	21 (2)	1000	65	4034
c1908	4	21 (6)	1000	57	3306	**22 (5)**	**1000**	**72**	**4545**
c1908	5	21 (6)	1000	60	3586	21 (4)	1000	83	5423
s838_1	2	43 (5)	250	65	4664	44 (1)	250	63	4322
s838_1	3	**49 (3)**	**250**	**127**	**8841**	41 (2)	250	61	4151
s838_1	4	45 (4)	250	89	6325	42 (2)	250	61	3777
s838_1	5	48 (6)	250	98	7400	**45 (0)**	**250**	**79**	**5322**
s9234.1_1	2	148 (1)	35	63	9806	**137 (0)**	**45**	**31**	**5913**
s9234.1_1	3	**157 (2)**	**35**	**69**	**9827**	123 (0)	45	10	1759
s9234.1_1	4	142 (2)	35	40	6342	127 (2)	45	31	5515
s9234.1_1	5	153 (0)	35	44	6773	127 (0)	45	21	4410

Tabulka 4.8 Výsledky testu selekčního tlaku GCMA1

Výsledek testu

Vidíme zde, že při použití 1-bodového křížení dosahuje algoritmus nejlepších výsledků při velikosti turnaje 3. V případě uniformního křížení jsou výsledky velmi různé a nedá se z nich odhadnout nejlepší nastavení selekčního tlaku. Zároveň je zde vidět velký rozdíl v dosažených výsledcích ve prospěch 1-bodového křížení. Na základě tohoto měření a s přihlédnutím k výsledkům úvodního testu 4.9.2 jsem se rozhodl dále testovat pouze 1-bodové křížení.

Na grafu 4.2 je zobrazen vývoj fitnes pro různý selekční tlak u matice s838_1. Ačkoli první část vývoje průměrné fitnes je pro všechny křívky přibližně podobná, křivka 3 si dokázala nejdéle udržet vzrůstající tendenci. Předpokládám, že zatímco s velikostí turnaje 4 a 5 byly populace kolem 90 generace už značně zdegenerované, při použití turnaje o velikosti 3 k tomuto jevu došlo až kolem 120 generace a díky tomu jsme dosáhli lepších výsledků. Selekční tlak 2 pravděpodobně nedokázal vyvážit úbytek fitnes, který je způsoben nepovedenými mutacemi a křížením. Podle naměřených hodnot a analýzy vývoje průměrné fitnes budu dále používat $p_S = 3$.



Graf 4.2 Porovnání vývoje fitnes GCMA1 při různém selekčním tlaku – matice s838_1

4.9.5 Test poměru křížení

Parametr křížení p_C byl testován v hodnotách mezi 0,3 a 0,9 nyní už pouze pro 1-bodové křížení. Naměřene výsledky jsou v tabulce 4.9.

název matice	p_C	GA1 (1-pnt)	pop.	gen.	čas (s)
c1908	0,3	**24 (1)**	**1000**	**47**	**3100**
c1908	0,5	23 (5)	1000	84	5157
c1908	0,7	22 (5)	1000	76	4569
c1908	0,9	21 (2)	1000	64	3854
s838_1	0,3	44 (2)	250	53	3421
s838_1	0,5	45 (2)	250	77	5317
s838_1	0,7	**49 (3)**	**250**	**127**	**8841**
s838_1	0,9	48 (2)	250	84	6055
s9234.1_1	0,3	157 (1)	35	74	10914
s9234.1_1	0,5	158 (0)	35	61	8771
s9234.1_1	0,7	157 (2)	35	69	9827
s9234.1_1	0,9	**161 (0)**	**35**	**70**	**10680**

Tabulka 4.9 Výsledky testu poměru křížení GCMA1

Výsledek testu

Vhodné nastavení mutačního parametru je závislé na vstupní matici. U malé matice c1908 se setkáváme s nežádaným jevem, kde čím menší je poměr křížení, tím lepších výsledků dosahujeme. V tomto případě hrají genetické operátory na kvalitě výsledných řešení minimální roli. Jediné k čemu dochází je selekce, která vybírá lepší jedince, a náhodné nastavování nespárovaných párů metodou zahazování sloupců. Dochází tak pouze k prohledávání lokálních optim v podobě jedinců v populaci. Z těchto závěrů se odvážím tvrdit, že GCMA1 s použitím metody zahazování

sloupců není pro řešení vstupní matice c1908 vhodné.

U zbylých dvou matic (s838_1, s9234.1_1) je situace jiná. V případě s838_1 se zdá být optimální poměr křížení $p_C = 0,7$, co se týče matice s9234.1_1 dosáhli jsme nejlepších výsledků s poměrem $pc = 0,9$. Parametr p_C doporučuji volit mezi hodnotami 0,5 – 0,9. Na místo volby nižší hodnoty tohoto parametru než je 0,5 (za účelem získání lepších výsledků) doporučuji použít raději algoritmu Thorough Search.

4.9.6 Test parametru mutace matches

Parametru p_M by testován pro rozmezí 0 – 0,05, tedy od žádné do 5-ti procentí mutace. Naměřené výsledky jsou v tabulce 4.10.

název matice	p_{R1}	GA1 (1-pnt)	pop.	gen.	čas (s)
c1908	0	22 (3)	1000	55	3381
c1908	0,002	**24 (6)**	**1000**	**80**	**5160**
c1908	0,01	23 (5)	1000	84	5157
c1908	0,05	20 (6)	1000	69	3974
s838_1	0	48 (1)	250	74	5067
s838_1	0,002	48 (5)	250	103	7624
s838_1	0,01	**49 (3)**	**250**	**127**	**8841**
s838_1	0,05	41 (2)	250	52	3469
s9234.1_1	0	**166 (1)**	**35**	**72**	**10845**
s9234.1_1	0,002	159 (2)	35	72	10910
s9234.1_1	0,01	161 (0)	35	70	10680
s9234.1_1	0,05	143 (4)	35	67	9641

Tabulka 4.10 Výsledky testu parametru mutace matches GCMA1

Výsledek testu

Z tabulky 4.10 je patrné, že 5-ti procentní mutace je příliš velká. Jinak je však velikost mutace závislá na vstupní matici. Například u matice s9234.1_1 si můžeme všimnout, že jsme dosáhli nejlepšího výsledku bez mutace. Obecně je však vhodné alespoň mírnou mutaci použít. Optimální mutaci odhaduji v rozmezí $p_M = 0,001 - 0,005$.

4.10 Naměřené výsledky a porovnání s existujícími metodami

Nastavení hlavních vstupních parametrů GCMA1 bylo odvozeno během testů 4.9.4 až 4.9.6. Doporučené intervaly nastavení všech parametrů uvádi tabulka 4.11.

Během testů byly prokázány mnohem lepší výsledky s použitím 1-bodového křížení, proto byly závěrečné testy provedeny pouze s ním. Pro každou matici byl algoritmus spuštěn 3-krát, abychom získali průměrné výsledky. Velikost populace byla opět nastavena tak, abychom dosáhli řešení přibližně mezi 1-2 hodinami. Pokud výpočet přesáhl 3 hodiny, byl automaticky ukončen. Výsledky shrnuje tabulka 4.12. Průměr spárovaných sloupců „avg_{CM}“ a průměr přímých spárování „avg_{DM}“ je

použit pro porovnání s existujícími metodami, především algoritmem Thorough Search a náhodným Column-Matching generátorem (Random Search), které poběží po průměrnou dobu běhu genetického algoritmu, která je uvedena ve sloupci „avg_T (s)“.

parametr	doporučené hodnoty
metoda křížení	1-bodové křížení
velikost populace	závislé na instanci
podmínka ukončení běhu p_T	3 - 5
parametr selekce p_S	3
pravděpodobnost křížení p_C	0,5 - 0,9
parametr mutace matches p_M	0,001 - 0,005
parametr mutace p_{R1} (vložení do permutace)	0
parametr mutace p_{R2} (prohození sousedů permutace)	kolem 0,1
metoda zahazování sloupců	ano

Tabulka 4.11 Doporučené nastavení vstupních parametrů GCMA1

matice	GCMA1 (1-pnt)						Thorough Search	Random Search
	1.běh	2.běh	3.běh	avg_{CM}	avg_{DM}	avg_T (s)		
s820	22 (17)	21 (20)	22 (15)	21,6	17,3	4692	**22 (17)**	20 (4)
c1908	22 (5)	23 (4)	24 (3)	23	4	4344	**25 (9)**	16 (2)
s420.1_1	29 (15)	29 (18)	29 (17)	29	16,6	6922	**30 (16)**	23 (1)
s420.1_4	29 (19)	29 (13)	29 (10)	29	14	5632	**30 (19)**	25 (2)
s641_1	54 (27)	54 (27)	54 (24)	54	25	2879	**54 (38)**	43 (2)
s838_1	48 (2)	47 (3)	49 (5)	**48**	**3,3**	5284	45 (16)	25 (0)
s5378_3	212 (6)	212 (3)	208 (5)	210,6	4,6	9899	**214 (197)**	202 (3)
s9234.1_1	161 (2)	160 (1)	165 (0)	162	1	10705	**164 (84)**	117 (2)
s13207.1_3	591 (1)	592 (5)	590 (2)	591	2,6	9587	**627 (305)**	498 (2)
s38417_1	932 (1)	933 (1)	932 (0)	**932,3**	**0,6**	10419	nelze	808 (0)

Tabulka 4.12 Naměřené výsledky GCMA1 a srovnání s Thorough a Random Search

Počet opakování Thorough a Random Search byl nastaven tak, aby celková době běhu programů byla rovna „avg_T (s)“. Navíc, aby test byl spravedlivý, byla implementována programu Random Search metoda zahazování sloupců. Nejlepší výsledky pro jednotlivé matice jsou v tabulce 4.12 označeny tučně. Algoritmus Thorough Search dosahuje téměř pro všechny vstupní matice nejlepších řešení, vyjímku tvoří pouze matice s838_1. Zřetelný rozdíl je především v počtu přímých spárování, nicméně vzhledem k používané heuristice v Thorough Search, se tyto rozdíly daly očekávat. Opakovaně lepší dosažené výsledky (z pohledu celkového počtu spárování) u matice s838_1 naznačují, že při správném nastavení vstupních parametrů může pro některé matice algoritmus GCMA1 Thorough Search překonat.

5 Modifikace genetického algoritmu - GCMA2

Nepříliš uspokojivé výsledky výše popsaného algoritmu mě vedly k bližšímu zkoumání výpočtu fitnes a vlastností vstupních matic, především matice T. V následující kapitole popíšu úvahu, kterou jsem se dopracoval k potenciálu kvality kombinace, tzn. vlastnost kombinace párů, kterou se pokusím využít při výpočtu fitnes. Kapitola další se bude zabývat vztahy mezi sloupci matice T, na jejichž základě vypracuji druhou modifikovanou verzi genetického algoritmu, kterou budu v dalším textu označovat jako GCMA2 (Genetic Column-Matching Algorithm version 2).

5.1 Potenciál kvality kombinace

Během analýzy výpočtu fitnes jsem se zaměřil na klesání počtu jedniček v matici B. Důvodem bylo to, že malý počet jedniček v této matici ve výsledku vede k situaci, že u daného jedince nelze provést další spárování.

Pro přehlednost nejdříve definuji kombinaci párů (sloupec matice T, sloupec matice C) CM délky i ,

$CM_i = ((T_1, C_1), (T_2, C_2), ..., (T_i, C_i))$

Tato kombinace párů v podstatě odpovídá poli rank původního genetického algoritmu, pouze s tou změnou, že její délka není pevně daná.

Na počátku výpočtu fitnes je matice B naplněna jedničkami, tuto matici budu dále nazývat *plná matice B*. S téměř každým úspěšným spárováním dochází v matici B k úbytku jedniček. *Podíl jedniček* v matici B po spárování *i*-tého páru $g(CM_i)$ je poměr mezi aktuálním počtem jedniček v matici B a počtem jedniček plné matice B.

$$g(CM_i) = \frac{počet\ jedniček\ po\ spárování\ i\text{-}tého\ páru}{výška\ T \cdot výška\ C}$$

Experimentálně jsem naměřil, že provádíme-li první spárování sloupce T s jakýmkoliv sloupcem C, je podíl jedniček $g(CM_1)$ přibližně lineárně uměrný počtu neúplně určených hodnot ve sloupci T. Podíl jedniček po prvním spárování je tedy nezávislý na sloupci C a přibližně platí:

$$g(CM_1) \approx d(T_1) = 0{,}5 \cdot \frac{dc(T_1)}{výška\ T} + 0{,}5 \quad ,$$

kde $dc(T_1)$ je počet neúplně určených hodnot ve sloupci T_1 a *výška T* je výška matice T. Krajním případem je sloupec matice T, který má pouze neúplně určené hodnoty (*prázdný sloupec*), pro který platí $d(T_1) = 1$. Jeho protipólem je sloupec bez neúplně určených hodnot (*plný sloupec*), kde $d(T_1) \approx 0{,}5$. Rovnici si můžeme odůvodnit tím, že důvodem úbytku přibližně poloviny jedniček po spárování plného sloupce, je přibližně stejný počet jedniček a nul ve všech sloupcích matice C. Tato skutečnost vyplývá z použitého obvodu pro vygenerování testovacích vzorků – LFSR, který generuje pseudo náhodné vektory s přibližně stejnými počty nul a jedniček. Z rovnice navíc vyplývá, že nejsou páry (sloupec T, sloupec C_1) a (sloupec T, sloupec C_2), kde by byl jeden pár na první pohled vhodnější než druhý. Vlastnost $d(T)$ budu nazývat *průměrný diferenční podíl jedniček*, její název ozřejmím v následujícím odstavci.

V dalších experimentech jsem vytvářel náhodné kombinace a sledoval podíl mezi počtem jedniček v matici B před spárováním *i*-tého páru a počtem jedniček po spárování *i*-tého páru $h(CM_{i-1}, T_i, C_i)$. Souvislost funkce h s funkcí podílu jedniček popisují následující rovnice:

$g(CM_1) = h(CM_0 = \emptyset, T_1, C_1)$,

$g(CM_i) = h(CM_0 = \emptyset, T_1, C_1) \cdot h(CM_1, T_2, C_2) \cdot ... \cdot h(CM_{i-1}, T_i, C_i)$

Funkci h budu dále nazývat *diferenční podíl jedniček*, protože sleduje změnu podílu jedniček při jednotlivých spárováních. Ve výše zmíněném experimentu se CM_{i-1} rovná jakékoliv kombinaci spárování (náhodná kombinace). S přibývající délkou testu se průměrné hodnoty $h(CM, T, C)$ pro sloupce matice T blížily hodnotě $d(T)$, tedy nezáleželo na předchozím párování *CM* ani na sloupci matice C. Z toho vyplývá název funkce $d(T)$.

Poslední měření se zaměřila na podíl jedniček v dalších fázích výpočtu fitnes a na to, do jaké míry můžou být z tohoto pohledu některé kombinace párů lepší než jiné. Na počátku jsem definoval počet náhodných vzorků, ve kterých bude hledán minimální a maximální podíl jedniček. S délkou kombinace *i* jsem tedy hledal minimální a maximální $g(CM_i)$. Vzorky měly pevně definované sloupce T, ke kterým byly náhodně generovány sloupce C. Jediné omezení pro sloupce C bylo, že se nesmí ve vzorku opakovat (vyplývá ze závěrů v kapitole 5.2). Délka vzorku, tj. počet párů, se v testech pohybovala mezi dvěma až osmi.

Ukázka testu pro 8 párů:

```
Analyza sloupcu matice T - podil 1 v matice B

Nahodnych vzorku urcenych pro analyzu: 1000
Pocet analyzovanych sloupcu T: 8
Sloupce matice T, urcene k analyze: 15 16 17 18 19 20 21 22

Minimalni podil jednicek: 0,00296
Maximalni podil jednicek: 0,0074
Prumerny podil jednicek:  0,00501
```

	1	2	3	4	5	6	7	8
minimální $g(CM)$	0,54824	0,28096	0,12788	0,06480	0,03120	0,01404	0,00692	0,00296
maximální $g(CM)$	0,57176	0,31792	0,16836	0,08736	0,04632	0,02460	0,01352	0,00740
průměrná $g(CM)$	0,56020	0,29962	0,14976	0,07737	0,03880	0,01940	0,00998	**0,00501**
maximální / průměrná	1,02063	1,06107	1,12419	1,12911	1,19381	1,26804	1,35470	1,49700

Tabulka 5.1 – Změna podílu jedniček s přibývajícími páry

Výstup testu pro sloupce 15 – 22 matice s820 provedeném na 1000 vzorcích zachycuje tabulka 5.1. Sloupce 15 – 22 byly zvoleny proto, že mají minimální množství neúplně určených hodnot. Je zřejmé, že kombinace s maximální $g(CM)$ mají lepší předpoklady k dosažení většího počtu spárování než zbylé. Například maximální podíl jedniček pro délku kombinace 7 je 0,01352, minimální podíl jedniček pro tu samou délku kombinace je 0,00692, tedy v kombinaci s maximálním podílem jedniček je v matici B přibližně dvakrát více jedniček než v kombinaci s minimálním podílem jedniček.

Porovnání kombinací párů stejné délky *i* se stejnými sloupci T je triviální. Lepší kombinace je ta, která po spárování všech párů bude mít větší podíl jedniček, tedy $g(CM_i)$.

Pokud ovšem začneme porovnávat kombinace, které mají v párech různé sloupce T, budeme muset použít jiného přístupu. Je totiž zřejmé, že kombinace obsahující prázdné sloupce (pouze neúplně určené hodnoty) bude s předchozím přístupem lepší, než kombinace obsahující plné sloupce matice T (bez neúplně určených hodnot).

Pro porovnání dvou kombinací s různými sloupci matice T mě napadlo využít funkce $d(T)$, tedy průměrný diferenční podíl jedniček. V další úvaze předpokládám, že pro průměrnou kombinaci se sloupci $T_1, T_2, ..., T_i$ přibližně platí:

$$avg(g(CM_i)) \approx d(T_1) \cdot d(T_2) \cdot ... \cdot d(T_i) ,$$

kde $avg(g(CM_i))$ je průměrná hodnota velkého množství náhodně vybraných kombinací délky i se sloupci T_1 až T_i. Platnost tohoto předpokladu můžeme ověřit na testu podílu jedniček matice s820 pro délku kombinace 8 (tabulka 5.1 – označeno tučně). Jsou zde použity sloupce 15 – 22 mezi kterými jsou 4 sloupce bez neúplně určených hodnot, 3 sloupce s jednou neúplně určenou hodnotou a 1 sloupec s třemi neúplně určenými hodnotami. V tomto testu jsme naměřili

$$avg(g(CM_i)) = 0{,}00501 ,$$

přitom teoretický podíl jedniček průměrného jedince by měl být

$$d(T_1) \cdot d(T_2) \cdot ... \cdot d(T_8) = 0{,}5^4 \cdot 0{,}52^3 \cdot 0{,}56^1 = 0{,}00492 .$$

Kvalitu dvou kombinací párů stejné délky *i* s různými sloupci T by mohl porovnat poměr

$$\frac{g(CM_i)}{avg(g(CM_i))} = \frac{g(CM_i)}{d(T_1) \cdot d(T_2) \cdot ... \cdot d(T_i)} =$$

$$= \frac{h(CM_0 = \emptyset, T_1, C_1)}{d(T_1)} \cdot \frac{h(CM_1, T_2, C_2)}{d(T_2)} \cdot ... \cdot \frac{h(CM_{i-1}, T_i, C_i)}{d(T_i)}$$

, tedy poměr mezi reálným podílem jedniček a teoretickým podílem jedniček.

Pro porovnání dvou kombinací o různých délkách a s různými sloupci T se nabízí použití *i*-té odmocniny:

$$q = \sqrt[i]{\frac{g(CM_i)}{avg(g(CM_i))}} = \sqrt[i]{\frac{g(CM_i)}{d(T_1) \cdot d(T_2) \cdot ... \cdot d(T_i)}}$$

Koeficient *q* budu dále nazývat *poteniál kvality kombinace* párů. Měl by vyjadřovat kvalitu dané kombinace z pohledu podílu jedniček, které po spárování kombinace zůstanou v matici B, a která je nezávislá na sloupcích matice T obsažených v kombinaci a na délce kombinace. Kombinace párů se z pohledu genetických algoritmů dá chápat jako schéma řádu *i*. Potenciál kvality bude v GCMA2 součástí fitnes, přitom se bude jednat o potenciál kvality párů levé části pole rank do pozice columns_matched.

5.2 Vlastnosti sloupců matice T

Tato úvaha vznikla na základě myšlenky, využít velkého množství neúplně určených hodnot v některých maticích T. V dalším textu budu uvažovat matice $C[k,i]$ a $T[l,j]$, značení (j,i) pro pár sloupce matice T a sloupce matice C a $(j,\neg i)$ pro negativní pár. Vlastnosti a vztahy mezi sloupci jsou odvozeny z tří základních postřehů:

1) Prázdný sloupec je vždy možné realizovat jako přímé spárování.
2) Máme-li sloupec j_1 s jednou úplně určenou hodnotou v řádku l a sloupec j_2, který je součástí platného páru (j_2,i), i je sloupec matice C, a platí $T[l,j_1]=T[l,j_2]$, pak pár (j_1,i) je také platný, navíc během spárování neubyde z matice B žádná jednička, tudíž nijak neovlivní další možná spárování. Naopak pár $(j_1,\neg i)$ nemůže být platným párem.
3) Máme-li sloupec j_1 s dvěmi úplně určenými hodnotami v řádku l_1 a l_2 a sloupec j_2, který má v řádcích l_1 a l_2 také úplně určené hodnoty, je součástí platného páru (j_2,i), a platí $T[l_1,j_1]=T[l_1,j_2]$ a zároveň $T[l_2,j_1]!=T[l_2,j_2]$, pak pár (j_1,i) ani $(j_1,\neg i)$ nemůže být platným párem.

Díky bodu 1 a 2 je možné omezit výpočet pouze na některé sloupce matice T a tím zkrátit dobu běhu programu. Bod 3 umožňuje zavést do genetického algoritmu jistou logiku, která zamezí párování sloupců, které nemůžou být součástí platného páru. Předtím se však ještě pokusím shrnout všechny druhy vztahů, které mohou mezi sebou sloupce matice T mít:

1) sloupce j_1 a j_2 jsou stejné
   - pro všechny řádky l matice T platí:
     $T[l,j_1]=T[l,j_2]$
   - pokud existuje platný pár (j_1,i), je platný i pár (j_2,i), $(j_2,\neg i)$ není platný
2) sloupce j_1 a j_2 jsou negativní
   - pro všechny řádky l matice T platí:
     pokud $T[l,j_1]!=don't\,care$ a $T[l,j_2]!=don't\,care$, pak $T[l,j_1]!=T[l,j_2]$
   - pokud existuje platný pár (j_1,i), je platny i pár $(j_2,\neg i)$, (j_2,i) není platný
3) sloupce j_1 je pozitivně závislý na j_2 (podobné bodu 1)
   - pro všechny řádky l matice T platí:
     pokud $T[l,j_1]!=don't\,care$, pak $T[l,j_1]=T[l,j_2]$
   - pokud existuje platný pár (j_1,i), je platný i pár (j_2,i), $(j_2,\neg i)$ není platný
4) sloupce j_1 je negativně závislý na j_2 (podobné bodu 2)
   - pro všechny řádky l matice T platí:
     pokud $T[l,j_1]!=don't\,care$, pak $T[l,j_2]!=don't\,care$ a $T[l,j_1]!=T[l,j_2]$
   - pokud existuje platný pár (j_1,i), je platný i pár $(j_2,\neg i)$, (j_2,i) není platný
5) sloupce j_1 a j_2 se doplňují (pozitivně)
   - j_1 není závislý na j_2 ani opačně, j_1 a j_2 nejsou stejné

- pro všechny řádky l matice T platí:

  $T[l, j_1] = don't care$ nebo $T[l, j_2] = don't care$ nebo $T[l, j_1] = T[l, j_2]$

- pokud existuje platný pár (j_1, i), může být platný i pár (j_2, i), $(j_2, \neg i)$ není platný

6) sloupce j_1 a j_2 se negativně doplňují

- j_1 není negativně závislý na j_2 ani opačně, j_1 a j_2 nejsou negativní
- pro všechny řádky l matice T platí:

  $T[l, j_1] = don't care$ nebo $T[l, j_2] = don't care$ nebo $T[l, j_1] != T[l, j_2]$

- pokud existuje platný pár (j_1, i), může být platný i pár (j_2, i), $(j_2, \neg i)$ není platný

7) sloupce j_1 a j_2 jsou bez vztahu

- nemaji mezi sebou žádný z výše uvedených vztahů
- existuje takové l_1 a l_2, pro které platí:

  $T[l_1, j_1] = T[l_1, j_2]$ a $T[l_2, j_1] = T[l_2, j_2]$

  $T[l_1, j_1], T[l_1, j_2], T[l_2, j_1], T[l_2, j_2] != don't care$

- pokud existuje platný pár (j_1, i), pak (j_2, i) ani $(j_2, \neg i)$ není platný

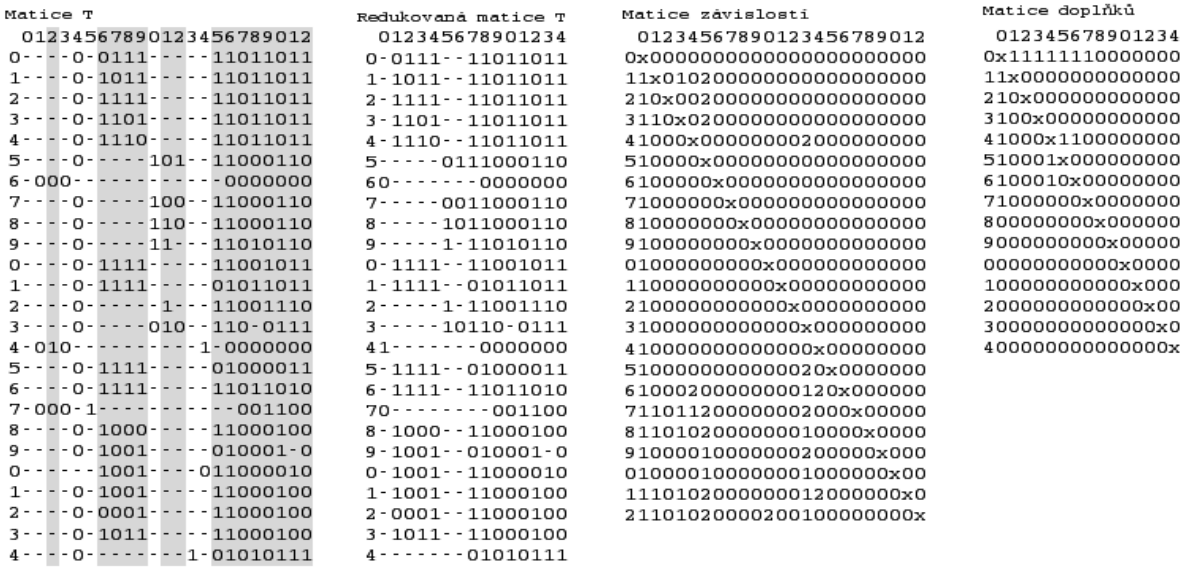
matice	šířka matice T	počet hlavních sloupců
s820	23	15
c1908	33	26
s420.1_1	34	21
s420.1_4	34	19
s641_1	54	14
s838_1	67	48
s5378_3	214	11
s9234.1_1	247	125
s13207.1_3	700	274
s38417_1	1664	509

Tabulka 5.2 Počet hlavních sloupců vstupních matic

Jak jsem již zmínil, díky výše popsaným vztahům je možné z matice T odstranit některé sloupce a tím zmenšit složitost problému. Sloupce, které mohou být odstraněny budu dále nazývat *vedlejší* a budou během běhu genetického algoritmu zastoupeny některým nebo některými zbývajícími – *hlavními* sloupci. Vedlejší sloupce budou všechny, které jsou závislé na nějakém jiném sloupci (pozitivně nebo negativně). Speciálním případem jsou prázdné sloupce, které jsou závislé na všech sloupcích, které obsahují alespoň jednu úplně určenou hodnotu. Také je nutné ošetřit případ, kdy máme nějaké stejné nebo negativní sloupce, které mají závislosti pouze navzájem mezi sebou. V ten moment určíme za hlavní sloupec ten nejmenší a zbylé budou vedlejší. Jakým způsobem se zmenší vstupní problém ukazuje tabulka 5.2.

Na obrázku 5.1 je zobrazena vstupní matice s820. Vlevo se nachází matice T a její hlavní sloupce jsou označeny šedě. Například je zde vidět prázdný sloupec 0, který je závislý na všech

ostatních sloupcích, sloupec 10, který je negativně závislý na sloupci 22 a sloupec 13, který je pozitivně závislý na sloupcích 16, 18, 20-22 a negativně závislý na sloupcích 15, 17 a 19. Odebráním vedlejších sloupců z matice T vznikne *redukovaná matice T*, která je na obrázku 5.1 druhá zleva. Veškeré závislosti mezi sloupci jsou definovány v *matici závislotí* (matice D) – obr. 5.1 třetí zleva. Je-li v poli [A, B] hodnota 1, pak sloupec matice T B je pozitvně závislý na sloupci A, jestliže je v poli [A, B] hodnota 2, pak je sloupc B negativně závislý na sloupci A, hodnota 0 znamená, že sloupec B není závislý na sloupci A. Jakým způsobem se mapují sloupce redukované matice T na sloupce matice T zachycuje pole *columnsTR_mapping* na obr. 5.2.



Obr. 5.1 Matice T, redukovaná matice T, matice závislostí, matice doplňků

columnsTR_mapping

2	6	7	8	9	11	12	15	16	17	18	19	20	21	22

Obr. 5.2 Pole columnsTR_mapping

Jádro GCMA2 bude pracovat pouze s redukovanou maticí T. Pokud bude třeba zjistit, jestli některý závislý sloupec byl spárován či ne (např. při počítání celkového počtu spárování), pak se využije pole columnsTR_mapping spolu s maticí závislostí.

Postupem vytvoření redukované matice T máme zajištěno, že se v ní nevyskytují stejné (resp. negativní) nebo závislé sloupce. Dva sloupce redukované matice T se tedy můžou buďto doplňovat (v našem případě se doplňuje např. sloupec 0 a 1 redukované matice T) nebo jsou bez vztahu. Matici vpravo na obrázku 5.1 nazývám *maticí doplňků* a její funkce je popsat doplňování sloupců v redukované matici T (0 – sloupce se nedoplňují, 1 – sloupce se doplňují pozitivně i negativně, 2 - sloupce se pozitivně doplňují, 3 – sloupce se negativně doplňují). Z obrázku 5.1 je zároveň patrné, že na příkladě matice s820 se většina hlavních sloupců v redukované matici T nedoplňuje (mají hodnotu 0). Pokud se dva sloupce redukované matice T nedoplňují , pak tyto dva sloupce nemůžou být spárovány se stejným sloupcem matice C. Pokud se dva sloupce redukované matice T doplňují, musí se držet pravidel doplňování (viz. souhrn vztahů ze začátku kopitoly - vztahy 5 a 6).

Podíváme-li se na konstrukci původního genetického algoritmu GCMA1, je tento závěr velmi zajímavý. V původním genetickém algoritmu totiž jednotlivé páry vznikaly vždy náhodně, ať

už při vytváření počáteční populace, během mutace nebo doplněním po křížení. Představme si například hledání řešení pro matici s820, která má 23 sloupců. Předpokládejme, že máme již jedince s 22 spárováními a zbývá nám spárovat poslední sloupec 19. Tento sloupec je bez vztahu ke všem zbylým čtrnácti hlavním sloupcům a navíc je bez vztahu i k šesti vedlejším (1, 3, 4, 10, 13 a 14). Sloupci 19 tedy podle předchozích úvah nemůže být přiřazeno 20 sloupců matice C z celkových 23. Pravděpodobnost, že dojde k přiřazení jednoho ze zbývajících tří sloupců je 3/23 = 0,13, což je velmi málo.

Proto se pokusím realizovat genetický algoritmus tak, aby páry jedinců neporušovaly pravidlo, že dva sloupce bez vztahu nesmí mít stejný sloupec C a zároveň, aby páry ctily pravidla doplňování. V programu to bude znamenat udržovat tuto *logickou konzistenci* pole matches a z toho důvodu modifikovat obě metody křížení, mutaci matches, vytváření náhodného jedince a výpočet fitnes.

Zavedením pravidel popsaných v této kapitole dosáhneme dvou pozitivních změn. Díky zastoupení vedlejších sloupců hlavními dojde k velkému zrychlení výpočtu fitnes a v následku udržování logické konzistence dojde k zefektivnění mutace a doplňování náhodných párů v pravé části chromozómu v rámci křížení.

5.3 Kódování jedince a genetické operátory

Kódování jedince se změní pouze minimálně. Základní genetické operace budou pracovat pouze s redukovanou maticí T. Z toho vyplývá zkrácení chromozómu (pole rank, matches, NM) na šířku redukované matice T. Určité změny se dotýkají i pole matches. Toto pole může mít, v případě, že sloupce redukované matice T se nedoplňují, vlastnosti permutace, tzn. že hodnoty v poli se nesmí opakovat.

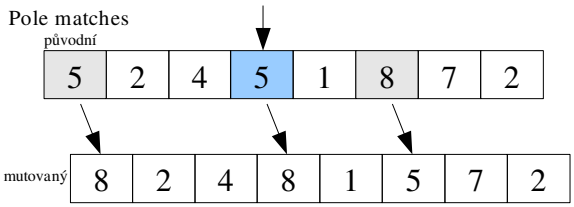
5.3.1 Křížení

Jak 1-bodové, tak uniformní křížení pracovalo tak, že postupně kopírovalo geny od rodičků k potomkovi. Přitom se před tímto kopírováním prováděla kontrola sloupce matice T obsaženého v genu, jestli potomek nemá jiný gen s tímto sloupcem. Pokud potomek gen měl, pak ke zkopírování genu nedošlo. Na závěr křížení se náhodně doplnily zbylé geny (sloupce matice T). Modifikace křížení bude samozřejmě pracovat pouze se sloupci redukované matice T. Nově bude kontrola genu spočívat i v kontrole sloupce matice C obsaženého v genu, aby jedinec udržel logickou konzistentci. Náhodné doplnění zbylých genů bude nahrazeno doplněním genů s takovými sloupci matice C, které zatím nejsou v ostatních genech obsaženy.

5.3.2 Mutace

Z mutací je nutné upravit pouze mutaci pole matches. Z genetických operátorů dojde právě zde k největší změně. Mutace je opět řízena parametrem p_M, která určuje pravděpodobnost mutace jednotlivého genu. Výběrem genu pro mutaci, je určené i pole matches, které budeme měnit - tato část se od původní mutace neliší. Následně se náhodně vybere sloupec matice C, který se bude nově nastavovat. Původní sloupec C v genu budu označovat *i*, nový sloupec *j*. Poté se provede kontrola,

jestli nastavením *j* nezpůsobíme logickou kolizi (pole matches by bylo nekonzistentní). Pokud kontrola bude v pořádku dojde k nastavení *j* a mutace tím končí. Pokud ovšem kontrola neprojde, provede se následující operace, která zaručuje, že udrží konzistenci jedince. Všechny pole pole matches, které měly původně hodnotu *i* budou mít nově hodnotu *j* . Všechny pole pole matches, které měly původně hodnotu *j* budou mít nově hodnotu *i*. Zde si musíme uvědomit, že tato operace může výrazněji změnit jedince a z toho důvodu bude pravděpodobně vhodné snížit parametr pro mutaci pole matches p_M. Na obrázku 5.3 je zobrazena mutace pole matches v případě nepovedené kontroly.



Obr. 5.3 Mutace pole matches

K obrázku 5.3, kde se pokoušíme zmutovat čtvrté pole matches (čtvrtý sloupec redukované matice T) bych rád doplnil, že sloupce 4 a 1 se zcela jistě doplňují, protože mají stejný sloupec C – 5. Stejně tak se určitě doplňují i sloupce 2 a 8 (hodnoty 2). Naopak můžeme s jistotou říci, že sloupec 4 a 6 jsou bez vztahu (nedoplňují se), protože jinak by mutace spočívala pouze ve změně hodnoty ve sloupci 4.

5.3.3 Selekce

Tím, že v redukované matici T zůstanou povětšinou z velké části plné sloupce způsobí zkrácení levé části pole rank (části do hodnoty columns_matched). Délka této části se dá velmi přibližně odhadnout. Aby po spárování prošlo přiřazení řádek v matici B, musí v každém sloupci této matice být alespoň jedna jednička. Protože má tato matice stejný počet řádek jako matice C, nesmí podíl jedniček klesnout pod hodnotu $1/výškaC$. Výška matice C, tj. počet PRPG vzorků, je v našich testech nejčastěji 1000. Předpokládejme nyní pro jednoduchost, že v redukované matici T jsou pouze plné sloupce pro které platí d(T) = 0,5. V tomto případě teoreticky klesne podíl jedniček v matici B na hodnotu 1/1000 po $\log_{0,5} 1/1000$ spárováních, což odpovídá přibližně 10 spárováním. Tento příklad byl pouze ilustrační, většinou se samozřejmě i v redukované matici T vyskytuje řada neúplně určených hodnot, nicméně ukazuje, že délka levé části pole rank se bude pohybovat v řádu desítek. U větších matic, kde počet sloupců redukované matice T se pohybuje řádově ve stovkách by pak během turnaje spolu mohly soupeřit kombinace, které spolu nesdílí ani jeden gen (sloupec T).

Metoda, často využívaná v MGA, která tomuto jevu částečně zamezí je tzv. *thresholding*. Jedná se o pravidlo, že jedinci, kteří spolu soupeří v rámci turnaje musí sdílet definované množství genů. Výběr jedince v turnaji o velikosti 2 probíhá následovně. Na počátku je do turnaje náhodně vybrán první jedinec. Následně náhodně vybereme druhého jedince a zkontrolujeme, jestli sdílí požadovaný počet genů – parametr *threshold*. Pokud sdílí dostatečné množství genů, je druhý jedinec přiřazen do turnaje a turnaj může proběhnout v podobě porovnání fitnes jedinců. Pokud dostatečné množství genů nesdíli, je náhodně vybrán další jedinec a proces se opakuje. Počet

opakování určuje parametr *shuffle_number*. Pokud ani poslední náhodný jedinec nemá patřičné množství společných genů, je selekcí vybrán první jedinec. Parametry pro thresholding jsem nastavil podle [Gol2] následovně:

$$threshold = \left\lceil \frac{l_1 \cdot l_2}{šířkaT} \right\rceil$$

$$shuffle\_number = šírkaT \quad ,$$

kde l_1 a l_2 jsou počty spárovaných sloupců jedinců, u kterých se počítají společné geny.

5.4 Změny výpočtu fitnes

Důležité změny se budou týkat výpočtu fitnes. V kapitole 5.1 popisuji potenciál kombinace, který zde chci využít. Základní složkou fitnes bude opět počet spárování. Počet spárování však nyní můžeme chápat dvojím způsobem, a to jako:

1) Počet sloupců vstupní matice T, které se povede spárovat – *celkový počet spárování*
2) Počet sloupců redukované matice T, které se povede spárovat – *redukovaný počet spárování*

Protože mají obě možnosti své výhody a nevýhody, rozhodnu jakou variantu z dvou výše uvedených použiji až během prvních testů.

První varianta znamená během výpočtu fitnes dopočítávat počet závislých sloupců na spárovaných sloupcích redukované matice T, což neznamená velký problém. Je zřejmé, že tímto způsobem dojde k prosazování těch hlavních sloupců, na kterých bude zavislých velké množství vedlejších sloupců. Díky tomu bude velmi pravděpodobné, že výsledný jedinec bude obsahovat právě tyto sloupce. Tato skutečnost však může být i na škodu, protože je možné, že některé sloupce matice T, hlavně ty, na kterých nejsou závislé žádné vedlejší sloupce, se budou jenom málokdy účastnit genetických operací, především křížení, a tím pádem bude značně narušena diverzita pole rank. Částečně by tomuto jevu mohl zabránit thresholding.

Druhá varianta nebude zvýhodňovat žádné sloupce redukované matice T. Tímto přístupem spíše udržíme diversitu pole rank a dosáhneme maximálního počtu sloupců redukované matice T, které se povede spárovat. Otázkou ovšem je, jaký budou mít výslední jedinci celkový počet spárování.

Druhou složkou fitnes bude příznak křížení, který bude nabývat hodnoty 1, pokud jedinec vznikl křížením nebo 0, pokud jedinec byl pouze vybrán z předchozí populace. Křížené jedince chci zvýhodňovat z toho důvodu, že znamenají větší posun ve stavovém prostoru a doufám, že díky tomu dojde k prohledání větší části stavového prostoru.

Poslední složkou fitnes bude potenciál kvality kombinace, do kterého budu během výpočtu uvažovat všechny sloupce redukované matice T v levé části pole rank do hodnoty columns_matched, tzn. redukovaný počet spárování. Potenciál kvality se pro většinu kombinací pohybuje v relacích $0{,}98 < q < 1{,}02$. Fitnes jsem se rozhodl počítat následovně:

$$fitnes = počet\ spárování + příznak\ křížení + potenciál\ kvality$$

Přestože jsou jednotlivé složky fitnes sčítány, bude mít největší prioritu počet spárování, který je řádově větší než příznak křížení i potenciál kvality. Jestli počet spárování bude ve skutečnosti redukovaný počet spárování nebo celkový počet spárování rozhodnu až během testů.

V původním algoritmu jsem uvažoval i počet přímých spárování. Vznik přímého spárování je pravděpodobnější u sloupců s velkým množstvím neúplně určených hodnot. Bohužel tyto sloupce jsou často vedlejší, a proto v redukované matici T většinou chybí. Z toho důvodu jsem se rozhodl počet přímých spárování z výpočtu fitnes odebrat a dosažení většího množství přímých spárování zajistit heuristikou pro optimalizaci přímých spárování na konci programu. Optimalizace přímých spárování bude více řešena v kapitole 5.5. Přesto bude během testů provedeno jedno měření (5.7.8), při kterém bude nahrazen potenciál kvality poměrem přímých spárování.

Při výpočtu fitnes budu nadále využívat heuristiky používající matici B. Vzhledem ke zkrácení chromozómu (viz. tabulka 5.1) dojde ke zkrácení doby výpočtu fitnes. Pokud *m* je počet sloupců redukované matice T, pak bude asymptotická složitost výpočtu fitnes $O(m \cdot p \cdot s)$.

5.4.1 Změny metody zahazování sloupců

Metoda zahazování sloupců dozná pouze malých změn. V původním algoritmu je zahozením myšleno odsunutí genu na konec chromozómu, tj. změna pořadí genů v poli rank, a nastavení náhodného sloupce matice C v tomto genu, tzn. změnu pole matches. Tuto druhou část (nastavení náhodného sloupce matice C) z důvodu zachování logické konzistence pole matches vynecháme. Metoda zahazování sloupců bude pracovat pouze s polem rank. Díky tomu bude mít každý výpočet fitnes přesně tolik vnitřních cylků (přiřazení řádek), kolik je počet sloupců redukované matice T.

5.5 Počítání celkového počtu spárování a přímých spárování

V předchozí kapitole jsem zmínil, že přímé spárování se budou počítat individuálně a nikoliv během výpočtu fitnes. Potom, co se vypočte jedinci fitnes, provede se i výpočet celkového počtu spárování (spárovaných sloupců matice T) a celkového počtu přímých spárování. Celkový počet spárování je pro výpočet jednoduchý. K hodnotě columns_matched, která je po výpočtu fitnes rovna počtu spárování sloupců redukované matice T, přidáme všechny vedlejší sloupce, které jsou závislé na jakémkoliv sloupci redukované matice T, který je součástí spárování. Počet přímých spárování se spočte podobně. U hlavních sloupců je existence přímého spárování pevně daná hodnotou v poli matches. U vedlejšího sloupce se provede kontrola jestli sloupec matice C, který by vytvořil přímé spárování, je mezi spárováními a pokud ano, jestli je vedlejší sloupec závislý na sloupci redukované matice T v tomto páru.

Ze způsobu počítání celkového počtu spárování vyplývá největší nevýhoda změn oproti GCMA1. Pokud bude vedlejší sloupec závislý pouze na jednom hlavním sloupci, který se nepovede spárovat, pak ani tento vedlejší sloupec nebude spárován. Přitom platí, že počet neúplně určených hodnot ve vedlejším sloupci může být mnohem menší než počet neúplně určených hodnot v hlavním sloupci a z toho důvodu je pravděpodobnost, že se povede spárovat vedlejší sloupec větší než pravděpodobnost spárování hlavního sloupce. Doufám však, že tento nedostatek převáží skutečnost, že vedlejší sloupce nebudou způsobovat ubývání jedniček z matice B.

5.6 Změny reprodukčního cyklu

Kostra reprodukčního cyklu zůstene beze změn. Jediný rozdíl bude, že s vytvořením každé další generace zkontrolujeme populaci, jestli se v ní nevyskytuje jedinec, který by měl nejvyšší doposavad nalezený celkový počet spárování a počet přímých spárování. Hledáme v podstětě jedince, který by měl nejlepší hodnoty fitnes z původního algoritmu. Zde je zajímavé si uvědomit, že budou genetické algoritmy řešit trochu jiný optimalizační problém, než je požadováno. Zatímco se chceme dobrat výsledku s co největším počtem spárovaných sloupců matice T pokud možno přímým spárováním, algoritmus bude optimalizovat počet spárovaných sloupců, který způsobí co nejmenší úbytek jedniček z matice B. Předpokládám však, že kvalitní výsledky by se tímto způsobem mohly dostavit.

Protože genetický algoritmus nijak nezohledňuje počet přímých spárování, pokusím se tento počet optimalizovat jednoduchou heuristikou. Poté, co skončí reprodukční cyklus se pokusíme u nejlepšího jedince přiřadit k vedlejším sloupcům jejich přímé protějšky z matice C, popřípadě jejich negace. Tento jedinec bude výstupem genetického algoritmu a budou na něm provedeny stejné úpravy jako původně, tzn. odstranění neúplně určených hodnot, test kompaktnosti, odstranění spárovaných sloupců a minimalizace Booleových funkcí.

5.7 Provedené testy

Na začátku testů jsem si stanovil podmínku stejnou jako při testování GCMA1, tedy že by doba běhu všech testů neměla překročit 3 hodiny a mělo by dojít k nalezení řešení mezi 1-2 hodinami.

Na rozdíl od GCMA1 bylo úkolem prvotního testu zjistit, jakým způsobem určovat fitnes, tzn. jestli použít redukovaný počet spárování nebo celkový počet spárování. Test byl proveden na 10-ti maticích a určil přibližné výsledky a chování algoritmu. Druhé měření bylo zaměřeno na vliv metody zahazování sloupců. Další testy byly stejně tak jako u GCMA1 provedeny s cílem nalezení vhodných intervalů pro jednotlivé parametry genetického algoritmu, a to na 3 testovacích maticích. Navíc byly provedeny testy stálosti dosažených výsledků, tzn. výkyvy ve výsledcích běhů programu se stejnými parametry na stejných vstupních maticích. Na závěr jsem se pokusil v rámci výpočtu fitnes nahradit potenciál kvality poměrem přímých spárování. Celkové výsledky jsou shrnuty v kapitole 5.8.

V testech byly opět použity testovací obvody ISCAS'85 [Brg85] a a '89 [Brg89]. Testovací matice poskytl Ing. Petr Fišer, Ph.D.

5.7.1 Úvodní test

Účelem prvních testů je rozhodnout, jestli během výpočtu uvažovat celkový počet spárování nebo redukovaný počet spárování. Zároveň provedu analýzu chování algoritmu. Parametry genetického algoritmu byly nastaveny po několika krátkých experimentálních měřeních, proto výsledky těchto měření nejsou optimální. Velikost populace pro jednotlivé vstupní zadání byla odhadnuta tak, aby se doba výpočtu pohybovala mezi jednou až dvěma hodinami, maximální doba

běhu je omezena třemi hodinami. Testy jsou provedeny jak pro 1-bodové, tak pro uniformní křížení, u kterých se některé vstupní parametry liší (p_{R1}, p_{R2}). Tabulka 5.3 popisuje vstupní proměnné pro oba druhy křížení, které byly přibližně experimentálně odvozeny z předběžných testů.

parametr	1-bodové křížení	uniformní křížení
velikost populace	závislý na instanci	závislý na instanci
časové omezení (v minutách)	180	180
podmínka ukončení běhu p_T	5	5
parametr selekce p_S	3	3
pravděpodobnost křížení p_C	0,5	0,5
parametr mutace matches p_M	0,01	0,01
parametr mutace p_{R1} (vložení do permutace)	0	0,01
parametr mutace p_{R2} (prohození sousedů permutace)	0,1	0
zahazování sloupců	ano	ano

Tabulka 5.3 Nastavení vstupních parametrů během úvodních testů GCMA2

Celkový počet spárování součástí fitnes

Během tohoto měření bylo fitnes vypočítáváno následovně:

fitnes = celkový počet spárování + příznak křížení + potenciál kvality

a byly použity parametry z tabulky 5.3.

Naměřené hodnoty jsou uvedeny v tabulce 5.4. Ve sloupcích „GA2 (1-pnt)“ a „GA2 (uni)“ jsou naměřené celkové počty spárování pro 1-bodové a uniformní křížení programu GCMA2. Sloupec „avg GA1“ obsahuje zaokrouhlené průměrné hodnoty naměřených výsledků GCMA1, který běžel podobně dlouhou dobu (časové omezení 3 hodinami). Ve sloupci „Thorough S.“ jsou výsledky, kterých dosáhl algoritmus Thorough Search běžící po dobu jedné hodiny. Dva posledně zmíněné sloupce jsou pouze informační, přesné porovnání GCMA2 s GCMA1, algoritmem Thorough Search a náhodným generátorem bude provedeno v kapitole 5.8.

název matice	GA2(1-pnt)	pop.	gen.	čas (s)	GA2(uni)	pop.	gen.	čas (s)	avg GA1	Thorough S.
s820	22 (7)	5000	38	5308	22 (7)	5000	34	4714	21 (17)	22 (17)
c1908	25 (1)	3000	45	6321	25 (2)	3000	51	6799	23 (4)	25 (9)
s420.1_1	29 (8)	5000	45	7362	29 (9)	5000	41	6534	29 (17)	29 (17)
s420.1_4	30 (10)	3000	42	5164	29 (12)	3000	40	5136	29 (14)	30 (19)
s641_1	54 (31)	8000	21	1170	54 (30)	8000	36	1921	54 (25)	54 (38)
s838_1	45 (3)	1000	44	7235	44 (5)	1000	51	9169	48 (3)	45 (16)
s5378_3	214 (197)	2000	19	655	214 (198)	2000	8	309	211 (5)	214 (197)
s9234.1_1	157 (73)	150	37	8916	154 (85)	150	30	6897	162 (1)	160 (62)
s13207.1_3	582 (356)	70	23	5553	581 (358)	70	16	4010	591 (3)	624 (264)
s38417_1	1046 (781)	70	25	3602	1060 (811)	70	15	2568	932 (1)	nelze

Tabulka 5.4 Výsledky měření s použitím celkového počtu spárování v rámci fitnes

Redukovaný počet spárování součástí fitnes

Během tohoto měření bylo fitnes vypočítáváno následovně:

fitnes = redukovaný počet spárování + příznak křížení + potenciál kvality

Parametry byly opět použity z tabulky 5.3.

Naměřené hodnoty jsou uvedeny v tabulce 5.5. Ve sloupci „max $GA2_1$“ jsou pro srovnání uvedeny maximální hodnoty z předchozího měření pro jednotlivé vstupní matice. Z tabulky 5.5 vyplývá, že použití celkového počtu spárovaných sloupců během výpočtu fitnes bude pravděpodobně výhodnější, protože téměř u všech maticí dosáhl tento způsob výpočtu fitnes lepších

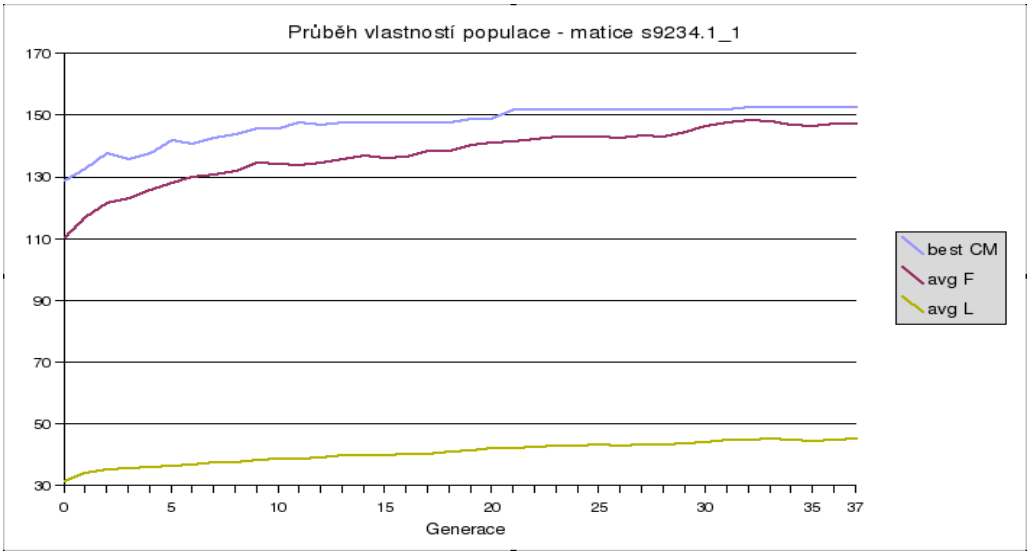
název matice	GA2(1-pnt)	pop.	gen.	čas (s)	GA2(uni)	pop.	gen.	čas (s)	max $GA2_1$	Thorough S.
s820	22 (5)	5000	21	2939	22 (7)	5000	17	2375	22 (7)	22 (17)
c1908	21 (2)	3000	14	1937	22 (2)	3000	33	4671	**25 (2)**	25 (9)
s420.1_1	28 (6)	5000	20	3549	28 (10)	5000	20	3567	**29 (9)**	29 (17)
s420.1_4	29 (10)	3000	25	3322	29 (11)	3000	32	4317	**30 (10)**	30 (19)
s641_1	54 (28)	8000	29	1621	54 (28)	8000	37	1916	54 (31)	54 (38)
s838_1	40 (3)	1000	23	4279	41 (4)	1000	47	9288	**45 (3)**	45 (16)
s5378_3	214 (196)	2000	7	300	214 (195)	2000	19	733	214 (198)	214 (197)
s9234.1_1	141 (82)	150	21	5491	146 (81)	150	19	4860	**157 (73)**	160 (62)
s13207.1_3	549 (366)	70	18	5017	559(364)	70	20	5627	**582 (356)**	624 (264)
s38417_1	1044 (808)	70	24	4431	**1078(528)**	70	37	6559	1060 (811)	nelze

Tabulka 5.5 Výsledky měření s použitím redukovaného počtu spárování v rámci fitnes

výsledků. Rozdíly nejsou příliš patrné u malých matic, resp. u matic, kde dojde ke spárování většiny sloupců, zatímco u velkých matic (např. s838_1, s9234.1_1, s13207.1_3) jsou rozdíly zřejmé. Na základě těchto výsledků jsem se rozhodl v dalších testech (vyjma testu 5.7.8) používat fitnes:

fitnes = celkový počet spárování + příznak křížení + potenciál kvality

Test se zatím nesoustředil na srovnání GCMA2 s GCMA1, náhodným generátorem, ani algoritmem Thorough Search. To bude provedeno až v kapitole 5.8.



Graf 5.1 Průběh fitnes a dalších vlastností populace GCMA2

Graf 5.1 ukazuje průběh některých hlavních vlastností populace během měření na maticích s9234.1_1 s použitím 1-bodové křížení a celkového počtu spárování při výpočtu fitnes. V tomto měření jsme dosáhli výsledku 157 spárování a 73 přímých spárování. Křivka „avg F“ popisuje nejzajímavější ukazatel, a to průměrnou fitnes. Maximální průměrné fitnes bylo dosaženo v 32. generaci. Protože v dalších pěti generacích nedošlo ke zlepšení průměrné fitnes, byl test ukončen.

Druhou sledovanou vlastností je největší celkový počet spárovaných sloupců („best CM“) mezi jedinci v populaci. Zde je zajímavé si všimnout, že maximální počet spárování, kterého bylo dosaženo také v 32 druhé generaci, je pouze 153, přitom dosažený výsledek je 157. Tato skutečnost je způsobena závěrečnou optimalizací jedince, kdy byly do spárování přidány 4 přímé spárování vedlejších sloupců.

Poslední sledovaný ukazatel „avg L“ je průměrný počet spárovaných sloupců redukované matice T. Ten zde uvádím pouze pro zajímavost. V průběhu měření vystoupal na přibližně 45 spárovaných hlavních sloupců a má podobný průběh jako průměrná hodnota fitnes.

Chování algoritmu můžou popsat také dvě sledované statistiky. Jedná se o matice, které nám můžou dát informaci o diverzitě pole matches a pole rank. V první matici S_{CM} se udržuje počet přiřazení sloupce redukované matice T ke sloupci matice C, tedy máme-li $S_{CM}(25, 36) = 250$ znamená to, že 25. sloupec redukované matice T tvořil ve 250 případech platný pár s 36. sloupcem matice C, tzn. tento pár se vyskytoval v levé části pole rank před pozicí columns_matched. Matice S_R podobným způsobem sleduje pozice sloupce redukované matice T v poli rank, tzn. pokud máme $S_R(24, 8) = 25$, pak to znamená, že 24. sloupec redukované matice T se 25-krát vyskytl na 8. pozici pole rank. Vzhledem k velikosti obou matic je přikládám pouze v elektronické podobě na přiložené CD, viz. příloha C, zde se je pokusím slovně popsat.

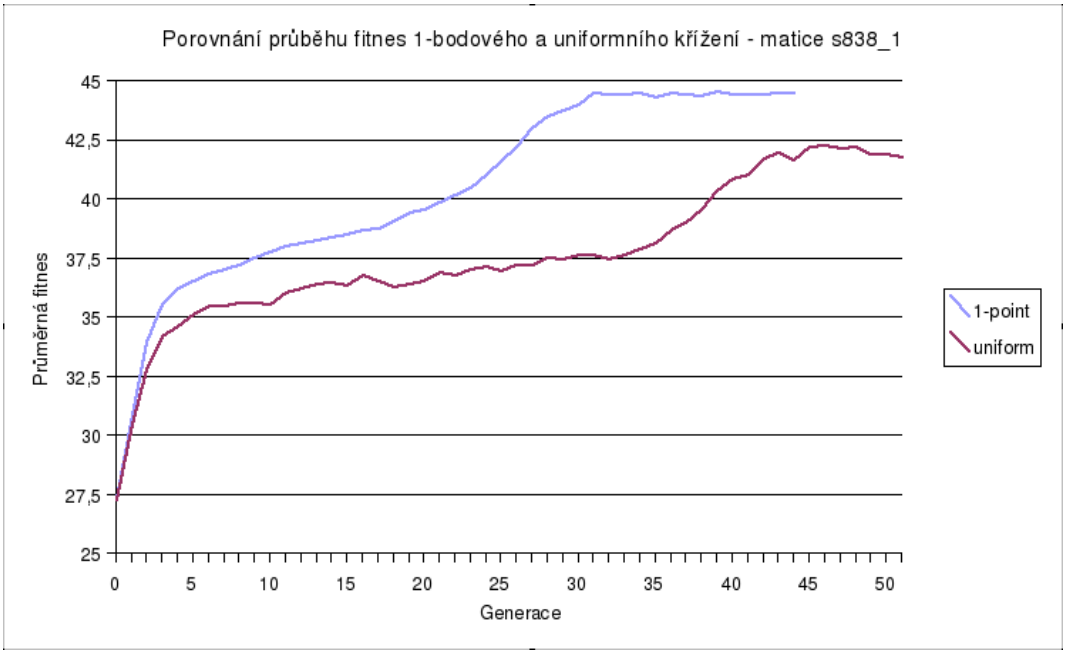
V řádcích matice S_{CM} (řádek reprezentuje sloupec redukované matice T) je možné rozlišit dva extrémy. První, který má hodnoty v řádku vcelku vyrovnané a druhý, kde se vyskytuje jedna nebo několik výrazně vyšších hodnot. V prvním případě, uvažujeme-li jeden konkrétní sloupec redukované matice T, byly rovnoměrně zkoušeny všechny kombinace párů se sloupci matice C. V druhém případě získala jedna kombinace páru v populaci převahu a tím došlo k omezení stavového prostoru, ve kterém se algoritmus dále pohyboval. Tento gen (ve smyslu sloupce redukované matice T) zdegeneroval. Zajímavé je sledovat poměr mezi nejvyšší hodnotou v řádku a průměrnou hodnotou řádku, který se pohyboval v rozmezí od 1,8 do 170. Je zřejmé, že v tomto případě došlo k postupné degeneraci populace a konvergenci k lokálnímu optimu. Lokální optimum téměř jistě nebylo optimální, protože se v matici S_{CM} přibližně v 5% stále vyskytují hodnoty 0, tzn. páry, které ani nebyly vyzkoušeny. S přihlédnutím k velikosti populace, která byla vzhledem k velikosti vstupních matic poměrně malá, se to však dalo očekávat.

V matici S_R také můžeme najít dva krajní případy řádků (opět řádek reprezentuje sloupec prořezané matice T), přitom mají souvislost s řádky matice S_{CM}. Prvním extrémním případem je řádek, který má velmi nízké hodnoty v levé části pole rank. Jedná se přesně o ty řádky, které měly vyrovnané hodnoty v matici S_{CM}. Tyto sloupce redukované matice T jenom málokdy byly součástí spárování jedince a jedná se pravděpodobně o sloupce obtížně spárovatelné. Naopak řádky matice S_R, které mají nízké počty v pravé části řádku, byly velmi často součástí spárování. Předpokládám, že reprezentují sloupce, které mají buďto velké množství neúplně určených hodnot, nebo je na nich závislých hodně vedlejších sloupců.

Porovnáním statistik S_{CM} a S_R běhu programu s použitím 1-bodového a uniformního křížení

zjistíme, že jsou si velice podobné. Znamená to, že v obou případech genetický algoritmu zkonvergoval ke stejným (vhodným) sloupcům redukované matice T.

Na grafu 5.2 je provedeno porovnání vývoje průměrné fitnes při použití 1-bodového a uniformního křížení, tentokrát pro vstupní matice s838_1. Oproti grafu 5.1 je zde patrná strmější část vývoje, která značí konvergenci k lokálnímu optimu. Na počátku křivka velmi rychle stoupne na úroveň průměrné fitnes 37 (kolem 7 generace). V druhé fázi se vývoj zpomalí, pravděpodobně dochází k postupnému navyšování počtu nadprůměrných schémat v populaci. V jeden moment získá několik schémat takovou převahu, že vytlačí ostatní schémata z populace a dojde ke konvergenci k lokálnímu optimu – strmá část (u 1-bodového křížení 22.-30. generace, u uniformního 35.-45. generace). V testech zaměřených na nalezení vhodných hodnot parametrů bychom se měli zaměřit na narovnání této strmé části, tzn. aby nárust fitnes byl plynulejší.



Graf 5.2 Porovnání průběhu fitnes 1-bodového a uniformního křížení GCMA2

Z pohledu porovnání metod křížení si můžeme všimnout menšího kolísání vývoje 1-bodového křížení. Pravděpodobně to způsobuje fakt, že během uniformního křížení dochází k většímu narušování existujících schémat, resp. s uniformním křížením dochází k větším změnám ve fitnes v porovnání s fitnes rodičů.

Ostatní běhy programu měly většinou podobný vývoj. Určitou zvláštností jsou matice s641_1 a s5378_3, kde se povedlo velmi rychle spárovat všechny sloupce matice T (stejně tak se to povedlo i algoritmu Thorough Search). Protože tato verze genetického algoritmu nemá za optimalizační kritérium počet přímých spárování, nebylo v podstatě co dále zlepšovat, brzy se zastavil vývoj fitnes, a tím pádem se ukončil i běh algoritmu.

5.7.2 Vliv metody zahazování sloupců

Během druhého měření budu zkoumat vliv metody zahazování sloupců. V tomto měření provedu testy bez metody zahazování sloupců se stejnými parametry (vyjma velikosti populace)

jako v úvodním testu. Tím, že nebudeme používat metodu zahazování sloupců dojde ke zkrácení doby výpočtu fitnes. Aby srovnání bylo regulérní, tzn. aby výsledky byly spočítány přibližně v 1-2 hodinách, navýšil jsem velikost populace na dvojnásobek.

název matice	GA2 (1-pnt)	pop.	gen.	čas (s)	GA2 (uni)	pop.	gen.	čas (s)	max $GA2_1$	T. Search
s820	22 (7)	10000	49	10859	22 (7)	10000	33	7112	22 (7)	22 (17)
c1908	23 (2)	6000	50	8049	23 (2)	6000	49	6348	**25 (2)**	25 (9)
s420.1_1	29 (9)	10000	42	10035	29 (8)	10000	40	10927	29 (9)	29 (17)
s420.1_4	30 (10)	6000	47	9070	30 (10)	6000	50	9488	30 (10)	30 (19)
s641_1	54 (32)	16000	32	3136	54 (31)	16000	28	2488	54 (31)	54 (38)
s838_1	44 (1)	2000	64	10207	38 (5)	2000	21	3011	**45 (3)**	45 (16)
s5378_3	214 (193)	4000	33	1961	214 (195)	4000	21	1342	214 (198)	214 (197)
s9234.1_1	149 (83)	600	48	6833	148 (88)	600	39	5139	**157 (73)**	160 (62)
s13207.1_3	514 (371)	140	27	4465	519 (390)	140	34	5920	**582 (356)**	624 (264)
s38417_1	1043 (810)	140	36	811	**1070 (844)**	140	20	439	1060 (811)	nelze

Tabulka 5.6 Výsledky měření bez metody zahazování sloupců GCMA2

Naměřené hodnoty jsou uvedeny v tabulce 5.6. Ve sloupci „max $GA2_1$“ jsou pro srovnání uvedeny maximální hodnoty z předchozího měření pro jednotlivé vstupní matice. U menších matic nejsou ve výsledcích prakticky žádné rozdíly. S použitím metody zahazování sloupců došlo k nalezení většího počtu spárovaných sloupců ve čtyřech případech (matice c1908, s838_1, s9234.1_1, s13207.1_3). V jednom případě (matice s38417_1 – uniformní křížení) však bylo dosaženo lepšího výsledku bez použití metody zahazování sloupců, a to dokonce ve velmi krátkém čase. Zde je dobré si uvědomit, že matice s38417_1 je největší testovaná matice, tedy má největší stavový prostor a na druhou stranu nejmenší velikost populace. Kvalita výsledku je potom závislá na počáteční náhodně vytvořené populaci a tím pádem může docházet k výkyvům ve výsledcích. Na to, jak stálé jsou výsledky genetického algoritmu bude sestrojen speciální test (5.7.7).

Přesto tento test neprokázal, stejně tak jako v GCMA1, že metoda zahazování sloupců je nutností. Je možné, že pro některé vstupní matice můžeme dosáhnout lepších výsledků bez použití metody zahazování sloupců. V dalších testech však metodu zahazování sloupců budu používat.

5.7.3 Test selekčního tlaku

Stejně tak jako u GCMA1 se další testy zaměří na nalezení vhodných parametrů genetického algoritmu, opět na maticích c1908, s838_1, s9234.1_1. Testy budou provedeny postupně pro parametry p_S, p_C, p_M a p_R.

První test se zabývá nastavením vhodného selekčního tlaku, resp. parametru selekce p_S. Použitá selekční metoda je ve všech případech turnaj, jehož velikost byla zvyšována od 2 do 5.

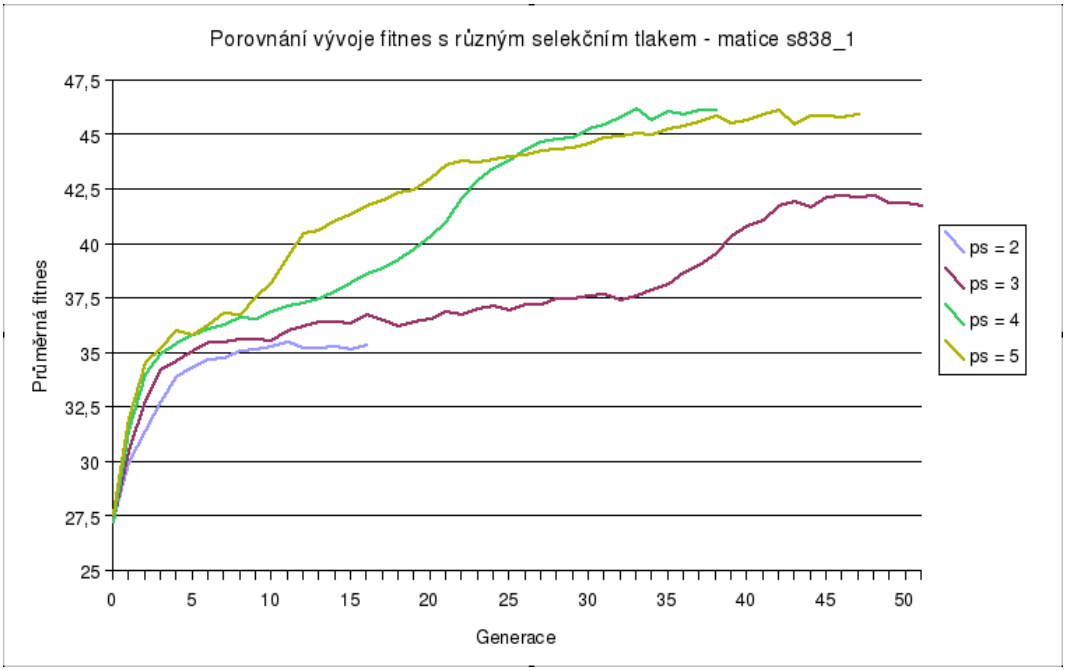
Z tabulky 5.7 je patrné, že selekční parametr má vliv na dobu běhu algoritmu. Pokud je selekční tlak příliš malý může předčasně dojít k zastavení vývoje fitnes, a to tak, že selekční tlak nevykompenzuje ztráty na kvalitě způsobené nepovedenými mutacemi a křížením. Tento jev je především patrný u uniformního křížení, kde s velikostí turnaje 2 dosahujeme o poznání horších

výsledků, viz. křivka „ps = 2“ v grafu 5.3, kde je provnán průběh fitnes pro různý selekční tlak se vstupními maticemi s838_1.

název matice	p_S	GA2 (1-pnt)	pop.	gen.	čas (s)	GA2 (uni)	pop.	gen.	čas (s)
c1908	2	24 (1)	3000	64	8465	23 (4)	3000	23	3055
c1908	3	25 (1)	3000	45	6321	**25 (2)**	**3000**	**51**	**6799**
c1908	4	**25 (1)**	**3000**	**30**	**3910**	25 (1)	3000	30	4065
c1908	5	24 (1)	3000	23	3303	24 (1)	3000	32	4510
s838_1	2	43 (3)	1000	46	7786	41 (3)	1000	16	2956
s838_1	3	**45 (3)**	**1000**	**44**	**7235**	44 (5)	1000	51	9169
s838_1	4	43 (2)	1000	29	5029	**48 (5)**	**1000**	**38**	**7101**
s838_1	5	44 (5)	1000	30	5255	47 (4)	1000	47	8843
s9234.1_1	2	159 (80)	150	60	10888	149 (80)	150	20	4665
s9234.1_1	3	157 (73)	150	37	8916	154 (85)	150	30	6897
s9234.1_1	4	**162 (77)**	**150**	**51**	**9908**	**159 (79)**	**150**	**34**	**7878**
s9234.1_1	5	159 (75)	150	41	8017	154 (84)	150	28	6889

Tabulka 5.7 Výsledky měření parametru selekce GCMA2

Naopak příliš velký selekční tlak může způsobit předčasnou konvergenci k lokálnímu optimu, což je také nežádoucí. Jako nejlepší se tedy pro naše účely jeví velikost turnaje 3 nebo 4. Pro další testy použiji selekční parametr 3 pro matice c1908 a selekční parametr 4 pro matice s838_1 a s9234.1_1.



Graf 5.3 Porovnání průběhu fitnes při různém selekčním tlaku GCMA2

Zajímavé také je všimnout si doby běhu algoritmu. U matic c1908 a s838_1 běžel algoritmus nejdéle při velikosti turnaje 3, pouze u matice s9234.1_1 při velikosti turnaje 4. Stejně tak jako u SGA a GCMA1 platí, že se zvětšováním selekčního talku, začne docházet ke zkrácení

doby výpočtu.

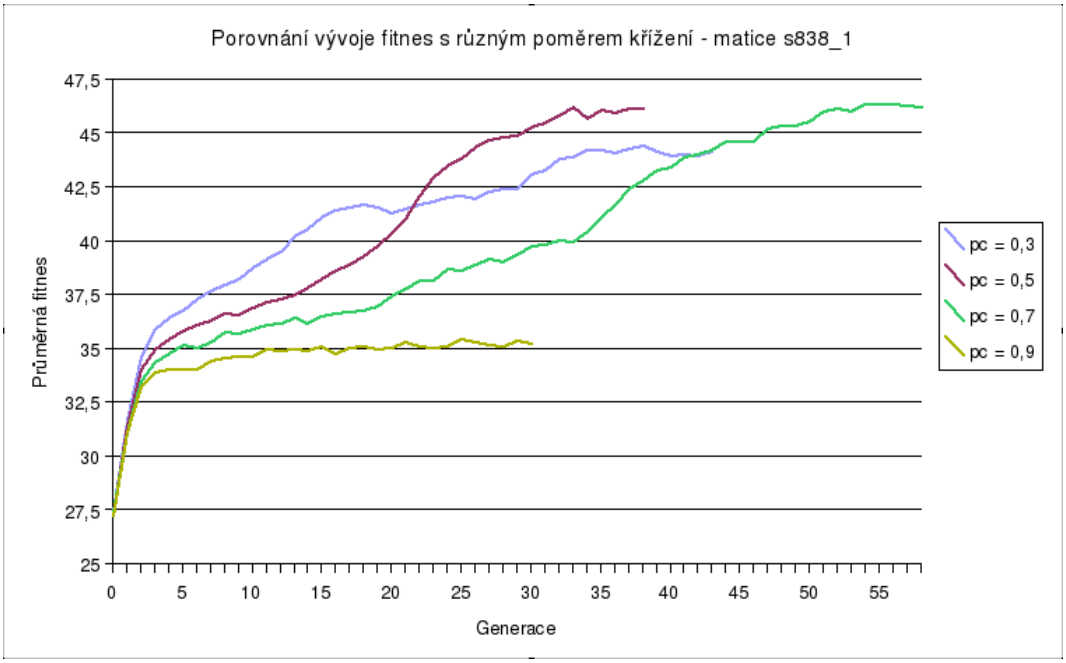
5.7.4 Test parametru křížení

Parametr křížení p_C byl testován v hodnotách mezi 0,3 a 0,9. Naměřené hodnoty uvádí tabulka 5.8.

název matice	p_C	GA2 (1-pnt)	pop.	gen.	čas (s)	GA2 (uni)	pop.	gen.	čas (s)
c1908	0,3	**25 (1)**	**3000**	**30**	**4056**	25 (1)	30000	35	4838
c1908	0,5	25 (1)	3000	45	6321	**25 (2)**	**3000**	**51**	**6799**
c1908	0,7	25 (1)	3000	43	5644	23 (0)	3000	21	2859
c1908	0,9	23 (2)	3000	38	5428	21 (2)	3000	15	1954
s838_1	0,3	43 (4)	1000	25	4603	46 (3)	1000	43	7652
s838_1	0,5	43 (2)	1000	29	5029	48 (5)	1000	38	7101
s838_1	0,7	**46 (3)**	**1000**	**42**	**7107**	**49 (3)**	**1000**	**58**	**10904**
s838_1	0,9	45 (5)	1000	41	7075	41 (5)	1000	30	5328
s9234.1_1	0,3	156 (79)	150	40	9028	**161 (77)**	**150**	**28**	**6610**
s9234.1_1	0,5	**162 (77)**	**150**	**51**	**9908**	159 (79)	150	34	7878
s9234.1_1	0,7	150 (59)	150	30	6367	155 (78)	150	28	6592
s9234.1_1	0,9	158 (79)	150	48	10430	153 (86)	150	17	4255

Tabulka 5.8 Výsledky měření parametru křížení GCMA2

Nastavení parametru křížení se liší v závislosti na vstupní matici. Se vstupní maticí s838_1 dosáhl program nejlepších výsledků s parametrem 0,7. Graf 5.4 zobrazuje průběh fitnes s použitím uniformního křížení a vstupními maticemi s838_1. Nejzajímavější je zde průběh s poměrem křížení 0,9, který je podobný jako v případě malého selekčího tlaku (viz. graf 5.3). Křížením v poměru 0,9 tedy dochází k takovému zhoršení fitnes jedinců, že ani selekční tlak s velikostí turnaje 4 tyto ztráty nedokáže vykompenzovat.



Graf 5.4 Porovnání průběhu fitnes při různém poměru křížení

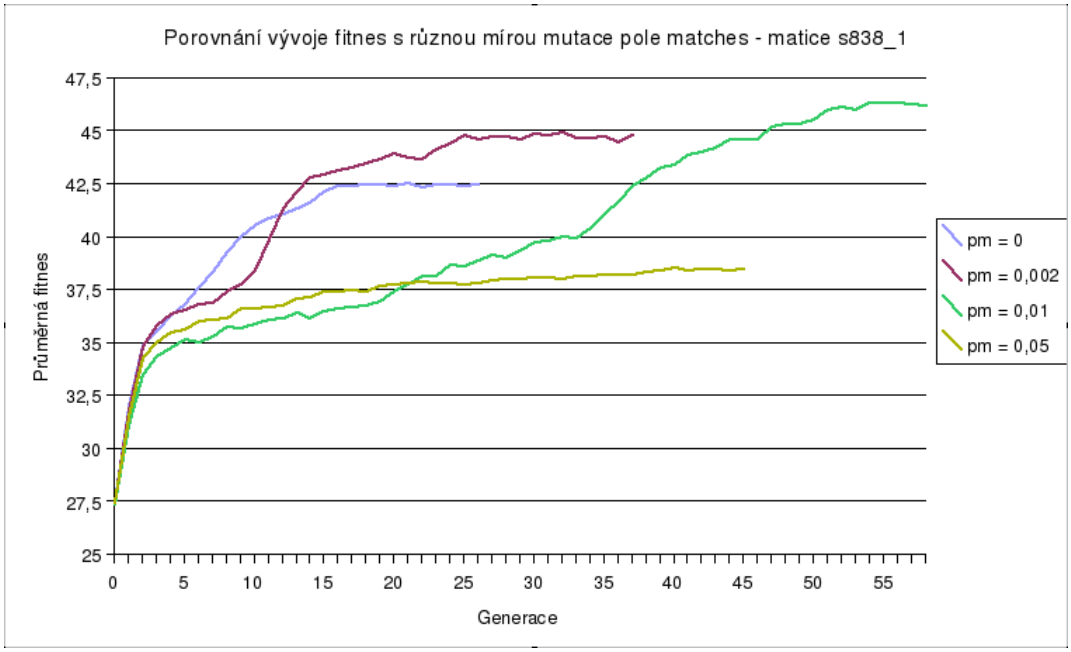
U matic c1908 a s9234.1_1 budu dále používat poměr 0,5, přestože slušných výsledků jsme dosáhli i s poměrem 0,3. Malý poměr křížení způsobí to, že na dosažené výsledky má vliv především mutace spolu se selekcí, přitom dochází k minimálnímu vyměňování kvalitních schémat mezi jedinci. Potom dojde k podobnému jevu jako v grafu 5.4 (křivka „pc = 3“), kde populace už od začátku konverguje k lokálnímu optimu. Proto je v našem zájmu, aby poměr křížení byl alespoň větší než 0,5. Optimální poměr křížení se bude pravděpodobně pohybovat mezi 0,5 - 0,7.

5.7.5 Test parametru mutace matches

Parametr mutace matches byl testován s hodnotami v rozmezí 0 – 0,05. Naměřené hodnoty zachycuje tabulka 5.9.

název matice	p_M	GA2 (1-pnt)	pop.	gen.	čas (s)	GA2 (uni)	pop.	gen.	čas (s)
c1908	0	25 (1)	3000	37	5222	**26 (3)**	**3000**	**38**	**5294**
c1908	0,002	**26 (1)**	**3000**	**34**	**4693**	24 (4)	3000	29	4456
c1908	0,01	25 (1)	3000	45	6321	25 (2)	3000	51	6799
c1908	0,05	22 (4)	3000	20	2788	23 (4)	3000	47	6319
s838_1	0	46 (2)	1000	36	6969	42 (4)	1000	26	4852
s838_1	0,002	44 (3)	1000	36	6645	45 (6)	1000	37	6455
s838_1	0,01	**46 (3)**	**1000**	**42**	**7107**	**49 (3)**	**1000**	**58**	**10904**
s838_1	0,05	40 (7)	1000	19	3608	42 (4)	1000	45	7904
s9234.1_1	0	151 (75)	150	25	6011	158 (70)	150	47	10580
s9234.1_1	0,002	159 (63)	150	45	10600	155 (65)	150	27	5991
s9234.1_1	0,01	**162 (77)**	**150**	**51**	**9908**	**159 (79)**	**150**	**34**	**7878**
s9234.1_1	0,05	155 (76)	150	36	8602	155 (70)	150	27	6300

Tabulka 5.9 Výsledky měření parametru mutace matches GCMA2



Graf 5.5 Porovnání průběhu fitnes při různém míře mutace matches GCMA2

Naměřené výsledky jasně vypovídají o tom, že 5-ti procentní mutace matches je příliš velká. To samé napovídá i graf 5.5. Dochází k tomu, co jsem již popisoval v úvodním (5.7.1) a minulém (5.7.4) testu, kdy následkem velké mutace vzniká velké množství málo kvalitních jedinců, kteří nejsou odfiltrováni selekcí a dostávají se do dalších populací. Tím dojde k zastavení vývoje fitnes. Celkem dobrých výsledků bylo dosaženo i bez použití mutace matches, přesto bych pro tento parametr doporučoval volit hodnoty mezi 0,002 a 0,01. Důvod, proč není vhodné nepoužít mutaci matches je viditelný na grafu 5.5, kde křivka „pm = 0“ od začátku strmě stoupá a algoritmus se velmi rychle uchýlí k lokálnímu optimu. Výsledky jsou pak silně závislé na počáteční náhodně vytvořené populaci. V dalších testech budu používat $p_M = 0.01$.

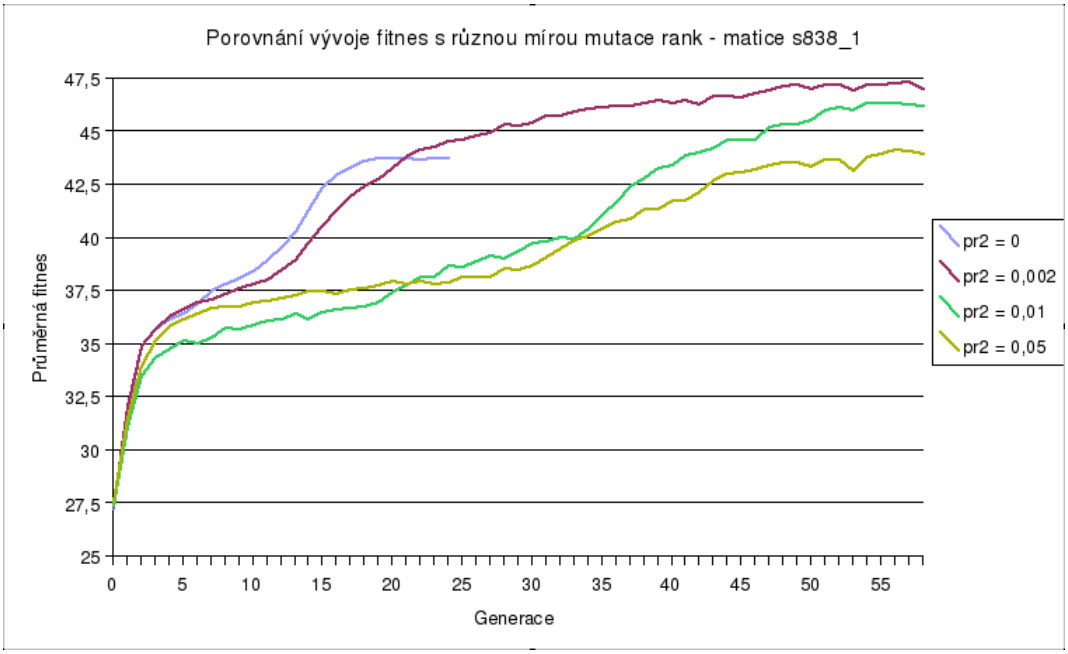
5.7.6 Test parametru mutace rank

V tomto experimentu byl testován parametr $p_{R1,}$ určující míru mutace vložení do permutace (4.4.2), pro uniformní křížení a parametr p_{R2}, který parametrizuje metodu prohození sousedů permutace (4.4.3), pro 1-bodové křížení. Hlavním účelem mutace pole rank je co nejdéle udržet diversitu pole rank. U 1-bodového křížení používám metodu mutace prohození sousedů permutace (4.4.3), protože má menší vliv na stavební bloky nízkého řádu. Testované rozmení parametru p_{R2} je 0 – 0,5. Naopak, protože mutace vložení do permutace má velký vliv na fitnes jedince, bude parametr p_{R1} u uniformního křížení testován v rozmezí 0 – 0,05. Naměřené hodnoty jsou v tabulce 5.10.

název matice	p_{R2}	GA2 (1-pnt)	pop.	gen.	čas (s)	p_{R1}	GA2 (uni)	pop.	gen.	čas (s)
c1908	0	25 (1)	3000	30	4361	0	**25 (2)**	**3000**	**41**	**5814**
c1908	0,02	24 (2)	3000	46	6467	0,002	24 (5)	3000	29	4036
c1908	0,1	**26 (1)**	**3000**	**34**	**4693**	0,01	24 (4)	3000	29	4456
c1908	0,5	24 (1)	3000	40	5591	0,05	25 (2)	3000	52	7130
s838_1	0	46 (3)	1000	47	9076	0	44 (5)	1000	24	4161
s838_1	0,02	44 (4)	1000	27	4922	0,002	47 (4)	1000	59	10903
s838_1	0,1	**46 (3)**	**1000**	**42**	**7107**	**0,01**	**49 (3)**	**1000**	**58**	**10904**
s838_1	0,5	45 (3)	1000	42	8097	0,05	46 (3)	1000	58	10927
s9234.1_1	0	154 (67)	150	43	10336	0	151 (76)	150	27	6235
s9234.1_1	0,02	160 (78)	150	21	4939	0,002	156 (77)	150	37	8425
s9234.1_1	0,1	**162 (77)**	**150**	**51**	**9908**	**0,01**	**159 (79)**	**150**	**34**	**7878**
s9234.1_1	0,5	151 (76)	150	42	10401	0,05	159 (77)	150	27	6240

Tabulka 5.10 Výsledky měření parametru mutace pole rank GCMA2

Z grafu 5.5 je zřejmý podobný vliv mutace pole matches na dosažené výsledky. Pokud mutaci neprovádíme (křivka „pr2 = 0“) dojde k rychlé konvergenci k lokálnímu optimu. Naopak u mutace 0,05 se již projevuje nízká úroveň fitnes, při které dojde k zastavení vývoje. Nejlepších výsledků jsme dosáhli s hodnotou parametru $p_{R1} = 0,01$ při uniformním křížení a $p_{R2} = 0,1$ při 1-bodovém křížení. Optimální nastavení parametrů se bude pohybovat kolem těchto hodnot.



Graf 5.6 Porovnání průběhu fitnes při různé mutaci rank GCMA2

5.7.7 Testy stability výsledného řešení

Cílem tohoto testu bylo získat představu, jak stálé jsou výsledky pro jednotlivé vstupní matice. Test byl proveden s hodnotami parametrů získané v předchozích testech. Pro každou z matic c1908, s838_1 a s9234.1_1 došlo k trojímu spuštění genetického algoritmu jak pro 1-bodové, tak pro uniformní křížení. Naměřené hodnoty uvádí tabulka 5.11. Ve sloupci „min“ je nejhorší dosažené řešení pro jednu matici, ve sloupci „max“ řešení nejlepší. Ve sloupci „avg_{CM}“ je průměrný počet spárovaných sloupců pro danou matici,

název matice	běh	GA2 (1-pnt)	pop.	gen.	čas (s)	GA2 (uni)	pop.	gen.	čas (s)	min	max	avg_{CM}
c1908	1	24 (3)	3000	32	4550	24 (3)	3000	27	3855			
c1908	2	25 (1)	3000	51	7120	26 (2)	3000	41	5745	24 (3)	26 (2)	24,5
c1908	3	24 (6)	3000	37	5452	24 (2)	3000	57	7846			
s838_1	1	46 (4)	1000	54	10373	45 (3)	1000	45	8221			
s838_1	2	44 (3)	1000	48	9209	44 (5)	1000	39	6543	44 (3)	46 (4)	44,7
s838_1	3	45 (4)	1000	60	10940	44 (7)	1000	49	8345			
s9234.1_1	1	159 (79)	300	22	11036	157 (83)	300	24	10951			
s9234.1_1	2	158 (75)	300	23	11117	160 (80)	300	24	11028	157(75)	160(80)	158,2
s9234.1_1	3	158 (80)	300	25	10995	157 (75)	300	23	10958			

Tabulka 5.11 Výsledky měření stability řešení

Bohužel výsledky nejsou stabilní, pohybují se v rozmezí 2 až 3 spárování, a to dokonce i u matice c1908, která patří mezi menší. Tento nežádoucí jev může být způsoben chybně nastavenými parametry, především příliš velkým selekčním tlakem. Pokud však uvážíme například matici c1908, kde jsme použili velikost turnaje 3, pak nejhorší výsledek je 24 spárování. Přitom během testu selekčního tlaku (5.7.3) jsme s menším selekčním tlakem (s velikostí turnaje 2) naměřili i hodnotu

pouhých 23 spárování (viz. tab. 5.6), z čehož vyplývá, že snížením selekčního tlaku sice možná výsledky více ustálíme, nicméně budou horší.

Druhé vysvětlení tohoto jevu, které je pravděpodobnější, je závislost výsledného řešení na počáteční populaci. Především u velkých matic je tato závislost celkem pochopitelná. Uvažme například matici s2934.1_1, která měla v tomto testu počáteční populaci o velikosti 300. Protože má matice T 125 hlavních sloupců a matice C má 247 sloupců je počet různých genů (pár hlavn9 sloupec matice T, sloupec matice C a příznak negování) roven $125 \cdot 247 \cdot 2 = 61750$, zatímco počet genů v počáteční populaci je pouze $125 \cdot 300 = 37500$. To znamená, že v počáteční populaci není téměř polovina genů. Vznik nových genů zajišťuje náhodná mutace pole matches (kap. 4.4.1), jak jsme ale zjistili v testu parametru mutace matches (kap. 5.7.5), tato mutace nemůže být vyšší jak 5% a není v jejích silách chybějící geny vytvořit.

S tímto tématem souvisí kap. 4.8, kde byl odvozen vztah pro velikost populace, která zajistí v počáteční populaci existenci většiny schémat řádu *o*. Díky tomu můžeme předchozí úvahu vztáhnout i na menší matice. Menší matice spojuje, že mají menší stavový prostor a větší populace. Sice je v počáteční populaci většina genů, nicméně o schématech řádu větším jak 1 se to už říci nedá. Opět doufáme, že se dobrá schémata tohoto řádu během vývoje vytvoří, a to buďto křížením nebo mutací pole matches. Pokud ovšem křížení a mutace není dostatečně efektivní, např. během křížení zaniká velké množství dobrých schémat a naopak jenom málo jich vzniká, pak jsme nuceni vytvořit velký selekční tlak, který způsobí zkrácení doby běhu algoritmu. A vzhledem ke krátkému běhu algoritmu je malý prostor pro vytváření nových schémat.

Nestabilita výsledků je tedy pravděpodobně jev, který nedokážeme odstranit nastavením parametrů a v dalších testech s ním musíme počítat.

5.7.8 Vliv potenciálu kvality

Toto měření má za úkol zjistit, jestli má potenciál kvality, který byl v předchozích měřeních součástí fitnes, velký vliv na dosažené výsledky. Potenciál kvality zde nahradím poměrem přímých spárování (počet přímých spárování ku celkovému počtu sloupců matice T), tzn. fitnes bude vypočítáváno následovně:

$$fitnes = celkový\ počet\ spárování + příznak\ křížení + poměr\ přímých\ spárování$$

Můžeme očekávat, že pokud bude poměr přímých spárování součástí fitnes, dosáhneme lepších výsledků z pohledu počtu přímých spárování.

Co se týče nastavení vstupních parametrů, předpokládám, že tato změna ve výpočtu fitnes na ně nebude mít vliv a použiji stejné hodnoty jako v předchozím měření. Protože předchozí test zjistil nestabilitu výsledného řešení, byl proveden tento test podobným způsobm, tzn. 3-krát pro matice c1908, s838_1 a s9234.1_1 pro uniformní i 1-bodové křížení. Naměřené hodnoty uvádí tabulka 5.12.

Porovnání nejlepších, nejhorších a průměrných výsledků je provedeno v tabulce 5.13. Sloupce s $GA2_7$ obsahují vlastnosti algoritmu, který při výpočtu fitnes používá potenciálu kvality. Sloupce s $GA2_8$ patří k algoritmu používající na místo potenciálu kvality poměr přímých spárování.

Z porovnání průměrného počtu spárování pro obě metody vyplývá, že potenciál kvality nepřináší výsledky s větším počtem spárování. Naopak při použití poměru přímých spárování dosahuje algoritmus o něco málo větší počty přímých spárování. Předpokládám, že rozdíly by byly

větší u matic T, které mají velké množství hlavních sloupců oproti vedlejším.

Pro závěrečné testy bude použit výpočet fitnes:

fitnes = celkový počet spárování + příznak křížení + poměr přímých spárování

název matice	běh	GA2 (1-pnt)	pop.	gen.	čas (s)	GA2 (uni)	pop.	gen.	čas (s)	min	max	avg_{CM}
c1908	1	25 (5)	3000	45	6512	25 (4)	3000	32	4316			
c1908	2	24 (4)	3000	26	3949	25 (3)	3000	35	4870	24 (3)	25 (5)	24,5
c1908	3	24 (4)	3000	38	4931	24 (3)	3000	32	4232			
s838_1	1	47 (7)	1000	60	10822	45 (7)	1000	43	7369			
s838_1	2	46 (6)	1000	58	10902	45 (7)	1000	57	9670	44 (7)	47 (7)	45,5
s838_1	3	46 (8)	1000	58	10973	44 (7)	1000	44	8097			
s9234.1_1	1	162 (76)	300	23	11107	155 (74)	300	25	11197			
s9234.1_1	2	155 (84)	300	23	10975	155 (83)	300	25	11279	155(74)	162(76)	157,5
s9234.1_1	3	159 (77)	300	25	11042	159 (79)	300	24	11074			

Tabulka 5.12 Výsledky měření vlivu potenciálu kvality

	výpočet fitnes s využitím potenciálu kvality			výpočet f. s využitím poměru přímých spárování		
název matice	min $GA2_7$	max $GA2_7$	avg $GA2_7$	min $GA2_8$	max $GA2_8$	avg $GA2_8$
c1908	24 (3)	26 (2)	24,5	24 (3)	25 (5)	24,5
s838_1	44 (3)	46 (4)	44,7	44 (7)	47 (7)	45,5
s9234.1_1	157 (75)	160 (80)	158,2	155 (74)	162 (76)	157,5

Tabulka 5.13 Výsledky měření vlivu potenciálu kvality – porovnání

5.7.9 Závislost kvality řešení a doby výpočtu na velikosti populace

Tabulka 5.14 zachycuje naměřené výsledky, při použití různých velikostí populace u matic s838_1 (uniformní křížení). Pro každou velikost populace v rozmezí 50 – 2000 byl algoritmus spuštěn dvakrát (1. běh značí sloupec „GA2a (uni)“, druhý běh je „GA2b(uni)“. V případě, že program běžel po dobu delší jak 5 hodin, byl ukončen. Toto je také důvod, proč s velikostí populace 2000 jsme dosáhli celkem nízkých výsledků. Jak je patrné z grafu 5.7 (křivka „2000“), byl v tomto případě algoritmus ukončen ještě před tím, než došlo ke konvergenci k lokálnímu optimu.

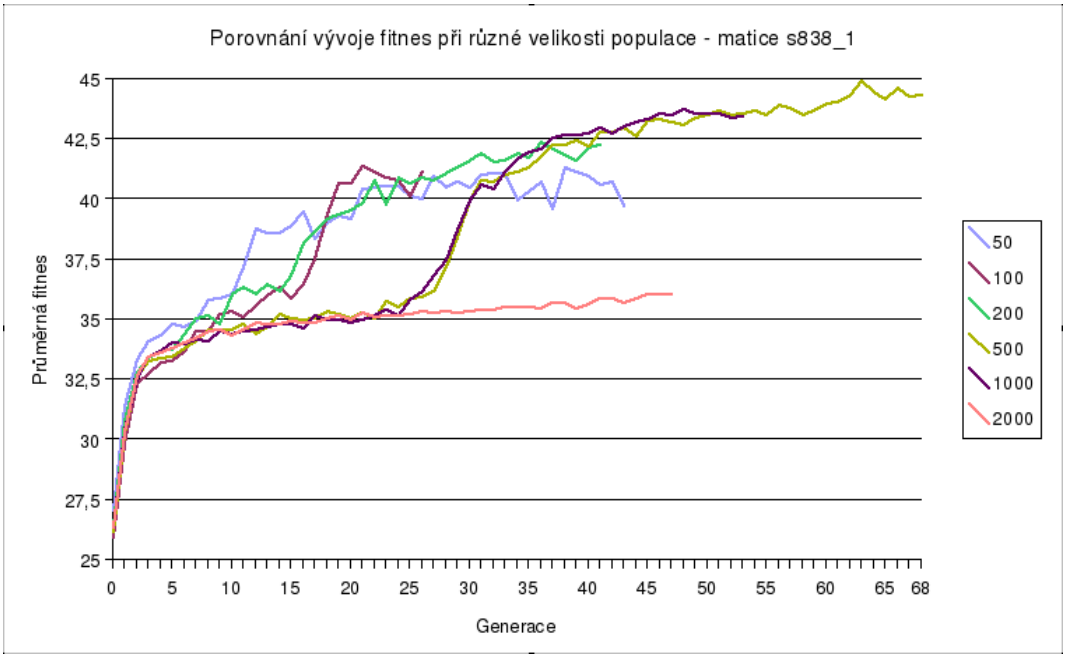
název matice	pop	GA2a (uni)	gen.	čas (s)	GA2b (uni)	gen.	čas (s)	avg_{CM}	avg gen.
s838_1	50	42 (5)	43	380	42 (4)	31	277	42	37
s838_1	100	42 (5)	26	479	40 (6)	12	224	41	19
s838_1	200	43 (4)	41	1622	43 (3)	35	1322	43	38
s838_1	500	47 (5)	68	6641	45 (7)	57	5472	46	62,5
s838_1	1000	45 (5)	53	10341	48 (10)	82	16011	46,5	67,5
s838_1	2000	44 (7)	47	18124	47 (6)	46	18345	45,5	46,5

Tabulka 5.14 Měření závislosti kvality řešení na velikosti populace GCMA2

V grafu 5.7 jsou uvedeny běhy algoritmu ze sloupce „GA2a (uni)“ tabulky 5.14. Z grafu 5.7 můžeme vyčíst, že úroveň generace, kdy začne docházet ke konvergenci k lokálnímu optimu se se vzrůstající populací zvyšuje a v závislosti na tom se zvyšuje i celkový počet generací. Tuto skutečnost vystihuje i sloupec „avg gen.“, kde je nárust populace především pro velikosti 100 – 1000 evidentní. Hodnotu „avg gen.“ pro velikost 2000 nemůžeme brát v úvahu, protože došlo k předčasnému ukončení vývoje. Protože námi navržený genetický algoritmus nejvíce zpomaluje doba výpočtu fitnes, přibližně platí, že

$$doba\ běhu\ algoritmu \approx počet\ generací \cdot doba\ ohodnocení\ populace$$

, kde doba ohodnocení populace je doba výpočtu fitnes všech jedinců v populaci. Jak jsme zjistili, počet generací s velikostí populace roste. Navíc, se zvyšující se populací lineárně roste i doba ohodnocení populace. Zvolíme-li tedy větší populaci, měli bychom očekávat vyšší než lineární nárust doby výpočtu.



Graf 5.7 Porovnání průběhu fitnes při různé velikosti populace GCMA2

Tento závěr platí pro vstupní matice s838_1. U ostatních matic může být skutečnost trochu jiná, nicméně je vysoce pravděpodobné, že výpočet se bude s narůstající velikostí populace minimálně lineárně prodlužovat.

Zároveň můžeme očekávat, že s větší velikostí populace dosáhneme lepších výsledků, viz. sloupec „avg_{CM}“ v tabulce 5.14.

5.8 Naměřené výsledky a porovnání s existujícími metodami

Vzhledem k výsledkům v testu 5.7.8, které prokázaly minimální prospěch použití potenciálu kvality, bude závěrečná verze GCMA2 pro ohodnocení kvality jedince používat následujícího vztahu:

$$fitnes = celkový\ počet\ spárování + příznak\ křížení + poměr\ přímých\ spárování \quad .$$

Doporučené intervaly pro nastavení vstupních parametrů genetického algoritmu, které byly převážně odvozeny během testů 5.7.3. až 5.7.6. shrnuje tabulka 5.15.

parametr	doporučené hodnoty
velikost populace	závislé na instanci
podmínka ukončení běhu p_T	3 - 5
parametr selekce p_S	3 - 4
pravděpodobnost křížení p_C	0,5 - 0,7
parametr mutace matches p_M	0,002 - 0,01
parametr mutace p_{R1} (vložení do permutace)	0 (1-bodové kř.), kolem 0,01 (uniformní kř.)
parametr mutace p_{R2} (prohození sousedů permutace)	kolem 0,1 (1-bodové kř.), 0 (uniformní kř.)
zahazování sloupců	ano

Tabulka 5.15 Doporučené nastavení vstupních parametrů GCMA2

Protože se během testů zjistila nestabilita dosahovaných výsledků, byly závěrečné testy provedeny pro každou matici a různou metodu křížení 3-krát. Velikost populace byla opět nastavena tak, abychom dosáhli výsledků přibližně mezi 1-2 hodinami. Pokud výpočet přesáhl 3 hodiny, byl automaticky ukončen. Výsledky pro 10 matic z úvodního testu shrnuje tabulka 5.16. Průměr spárovaných sloupců „avg_{CM}“ a průměr přímých spárování „avg_{DM}“ bude použit pro porovnání s existujícími metodami, především algoritmem Thorough Search a náhodným Column-Matching generátorem , které poběží po průměrnou dobu běhu genetického algoritmu, která je uvedena ve sloupci „avg_T (s)“.

matice	GCMA2 (1-pnt)						GCMA2 (uni)					
	1.běh	2.běh	3.běh	avg_{CM}	avg_{DM}	avg_T (s)	1.běh	2.běh	3.běh	avg_{CM}	avg_{DM}	avg_T (s)
s820	22 (15)	22 (14)	22 (11)	**22**	**13,3**	7745	22 (13)	22 (12)	22 (12)	22	12,3	5335
c1908	25 (5)	24 (4)	24 (4)	24,3	4,3	5130	25 (4)	25 (3)	24 (3)	**24,6**	**3,3**	4472
s420.1_1	29 (8)	29 (10)	29 (7)	29	8,3	5894	29 (8)	29 (11)	29 (10)	**29**	**9,6**	7278
s420.1_4	30 (14)	30 (12)	29 (16)	29,6	14	5962	29 (14)	31 (12)	30 (14)	**30**	**13,3**	4959
s641_1	54 (32)	54 (31)	54 (31)	**54**	**31,6**	1177	54 (31)	54 (33)	54 (30)	54	31,3	1406
s838_1	47 (7)	46 (6)	46 (8)	**46,3**	**7**	10899	45 (7)	45 (7)	44 (7)	44,6	7	8378
s5378_3	214(193)	214(190)	214(192)	214	191,6	875	214(194)	214(192)	214(194)	**214**	**193,3**	916
s9234.1_1	162(76)	156(81)	161(74)	**159,6**	77	8785	157(78)	157(83)	159(81)	157,6	80,6	7982
s13207.1_3	591(351)	588(353)	593(356)	**590,6**	**353,3**	6889	581(359)	583(353)	585(357)	583	356,3	6660
s38417_1	1103 (794)	1113 (785)	1096 (805)	**1104**	**794,6**	4372	1097 (787)	1075 (801)	1066 (798)	1079, 3	795,3	2981

Tabulka 5.16 Naměřené hodnoty GCMA2 – 10 matic

V tabulce 5.16 jsou tučně označeny průměrné počty spárování (resp. přímých spárování) metody křížení, kde bylo dosaženo lepšího výsledku. Zatímco u menších matic (s820 – s641_1, s5378_3) dosáhly obě metody velmi podobných výsledků, s narůstající velikostí matice se začly

projevovat rozdíly ve prospěch 1-bodového křížení. Z toho důvodu bude pro porovnání s ostatními metodami použito pouze 1-bodového křížení.

Naměřené průměrné výsledky 1-bodového křížení jsou v tabulce 5.17 porovnány s náhodným generátorem řešení. Aby srovnání bylo spravedlivé, používal náhodný generátor (Random Search) při vyhodnocování řešení metody zahazování sloupců, zastupování vedlejších sloupců matice T sloupci hlavními, také pracoval s informacemi o vztazích mezi hlavními sloupci matice T (např. nepároval jeden sloupec matice C s dvěmi hlavními sloupci matice T, které se nedoplňují, viz. kapitola 5.2) a na závěr prováděl optimalizaci přímých spárování. GCMA2 u všech vstupních matic překonal Random Search a to jak v celkovém počtu spárování, tak v počtu přímých spárování.

matice	čas (s)	GCMA2 (1-pnt)		Random Search	
		avg_{CM}	avg_{DM}	CM	DM
s820	7745	22	13,3	21	11
c1908	5130	24,3	4,3	21	3
s420.1_1	5894	29	8,3	28	8
s420.1_4	5962	29,6	14	29	11
s641_1	1177	54	31,6	53	30
s838_1	10899	46,3	7	38	2
s5378_3	875	214	191,6	214	190
s9234.1_1	8785	159,6	77	146	77
s13207.1_3	6889	590,6	353,3	548	350
s38417_1	4372	1104	794,6	1063	788

Tabulka 5.17 Porovnání GCMA2 s náhodným generátorem

Na závěr se pokusím provést souhrné porovnání GCMA1, GCMA2 a algoritmu Thorough Search. Oba genetické algoritmy používaly 1-bodové křížení. Porovnání rozšířím o dalších 11 matic, které poskytl Ing. Petr Fišer Ph.D. Výsledky GCMA1 a GCMA2 byly v případě prvních deseti matic průměrovány ze 3 běhů algoritmu, ve zbylých ze 2 běhů. GCMA1 i GCMA2 měly nastavené podobné podmínky, kdy jsem se snažil o dosažení výsledku mezi 1-2 hodinami. Bohužel, protože každá matice je velmi specifická, lze jen velmi těžko nastavit vstupní parametry (především velikost populace) tak, aby algoritmus zkonvergoval v očekávanou dobu, proto je porovnání obou verzí genetický algoritmů pouze přibližné. Průměrná doba běhu genetických algoritmů je uvedena ve sloupcích „avg_T (s)“. Porovnání s Thorough Search už můžeme považovat za přesné, protože mu byl spočítán počet opakovaní (sloupec „R“) tak, aby běžel po průměrnou dobu běhu GCMA2. Naměřené hodnoty jsou v tabulce 5.18. U každé matice jsou navíc další informativní sloupce: „n = p“ - počet sloupců matice C a T, které byly u všech matic stejné, „p“ - počet řádků matice C (pseudo-náhodných vzorků), „s“ - počet řádků matice T (deterministické vzorky), „DCs“ poměr neúplně určených hodnot v matici T a „m“ - počet hlavních sloupců matice T.

matice						GCMA1			GCMA2			Thorough Search		
nazev	r = n	p	s	DCs	m	avg_T (s)	CM	DM	avg_T (s)	CM	DM	R	CM	DM
s820	23	1000	25	0,473	15	4692	21,6	17,3	7745	22	13,3	**48850**	**22**	**17**
c1908	33	1000	28	0,450	26	4344	23	4	5130	24,3	4,3	**7750**	**25**	**10**
s420.1_1	34	500	42	0,501	21	6922	29	16,6	5894	29	8,3	**18800**	**30**	**16**
s420.1_4	34	1000	34	0,490	19	5632	29	14	5962	29,6	14	**12200**	**30**	**19**
s641_1	54	500	13	0,644	14	2879	54	25	1177	54	31,6	**10800**	**54**	**38**
s838_1	67	1000	61	0,533	48	**5284**	**48**	**3,3**	10899	46,3	7	2070	45	16
s5378_3	214	1000	19	0,913	11	9899	210,6	4,6	875	214	191,6	**720**	**214**	**197**
s9234.1_1	247	1000	215	0,818	125	10705	162	1	8785	159,6	77	**36**	**164**	**84**
s13207.1_3	700	1000	197	0,952	274	9587	591	2,6	6889	590,6	353,3	**8**	**627**	**305**
s38417_1	1664	1000	105	0,838	509	10419	932,3	0,6	**4372**	**1104**	**794,6**	nelze		
c2670_x1	233	1000	118	0,831	53	3222	190	1	**4140**	**199**	**159**	69	193	163
c2670_x2	233	1000	110	0,832	51	3461	193,5	4	**2181**	**199**	**167**	50	196	172
c7552_x1	207	1000	210	0,717	100	3906	115	1	3773	115,5	21,5	**16**	**119**	**24**
c7552_x2	207	1000	197	0,717	96	**9559**	**135**	**0,5**	5496	117,5	25,5	26	129	26
s13207_x1	700	1000	598	0,978	215	10885	649	0	7244	671	460	**6**	**673**	**497**
s13207_x2	700	1000	26	0,978	16	2017	693	3	**367**	**700**	**678**	115	700	672
s38417_2	1664	1000	500	0,979	278	8071	1175	0	**3492**	**1503**	**915**	nelze		
s838_x1	67	1000	96	0,559	47	**5003**	**42**	**4,5**	4146	40,5	3	460	41	14
s838_x2	67	1000	78	0,531	37	4348	50	6	**5036**	**52**	**4,5**	730	50	12
s9234_x1	247	1000	921	0,908	150	10063	169	1	**4220**	**178**	**73**	2	121	40
s9234_x2	247	1000	76	0,870	52	4853	218	5	3200	220	140,5	**84**	**220**	**155**

Tabulka 5.18 – Závěrečné porovnání GCMA1, GCMA2 a Thorough Search

Nejlepší naměřené hodnoty jsou v tabulce 5.18 označeny šedě. Nejčastěji dosáhl nejlepších výsledků algoritmus Thorough Search. GCMA2 jej dokázalo co do celkového počtu spárování překonat přibližně ve 30-ti procentech vstupních matic. V některých případech byly nejlepší řešení naměřeny i v případě GCMA1, celkově však tento algoritmus vykazoval v porovnání s Thorough Search a GCMA2 horší výsledky. Navíc GCMA1 velmi zaostával v počtu přímých spárování, to je však jednoznačně způsobené heuristikami pro navýšení počtu přímých spárování v GCMA2 i Thorough Search.

Ze závěrečného měření můžeme vyslovit závěr, že pro řešení Column-Matching problému je vhodnější použít algoritmus Thorough Search, pro některé vstupní matice však může dosáhnout lepších řešení GCMA1 i GCMA2. Jaké matice to však jsou není z výsledků patrné.

Volba mezi GCMA1 a GCMA2 je také závislá na vstupní matici. GCMA2 však znamená větší jistotu dosažení kvalitních výsledků.

5.9 Optimalizace GCMA2

Poslední vylepšení, které bylo v GCMA2 v rámci této práce implementováno, je optimalizace, která umožňuje zkrácení doby výpočtu. Po ukončení výpočtu fitnes máme zaručeno, že po spárování všech platných párů jedince, a s tím spojené úpravě matice B, se povede provést přiřazení řádek matice T k řádkům matice C. Pokud však změníme *j*-tý gen chromozómu, kde *j* < *columns_matched*, pak úroveň počtu spárování, po kterém se zaručeně povede provést přiřazení řádek, klesne na hodnotu *j – 1*. Jestliže bychom chtěli vypočítat fitnes tohoto jedince, mohli bychom pro urychlení prvních *j – 1* cyklů vynechat kontrolu přiřazení řádek. Optimalizace tedy spočívá v tom, že součástí informací jedince je i proměnná, ve které je uložen počet spárování, během kterých není nutné provádět přiřazení řádek. Zároveň je zaručeno, že se po spárování tohoto počtu párů povede provést přiřazení řádek.

Optimalizace je mnohem efektivnější při použití 1-bodového křížení. Po uniformním křížení, kde potomek postupně získává páry náhodně od obou rodičů, je totiž zaručená úroveň spárování vždy velmi nízká. Přesto však dochází k optimalizaci v momentě, kdy jedinec není křížen, tzn. je kopírován z předchozí generace. Testy dosaženého zrychlení byly provedeny pouze pro 1-bodové křížení, a to tak, že byl algoritmus spuštěn se stejnou velikostí populace jako během závěrečných testů. Zrychlení *a* se získalo podle následujícího vztahu:

$$a = \frac{trvání\ 1\ generace\ záv.\ testu}{trvání\ 1\ generace\ optimalizace} = \frac{počet\ generací\ optimalizace}{počet\ generací\ záv.\ testu} \cdot \frac{doba\ záv.\ testu}{doba\ optimalizace}$$

Musíme si uvědomit, že zrychlení je závislé i na ostatních parametrech genetického algoritmu, a to u 1-bodového křížení především na nastavení parametrů p_C a p_M, tedy podílu křížení a mutace matches. Během tohoto testu byly nastaveny hodnoty $p_C = 0,6$ a $p_M = 0,01$. Naměřené výsledky popisuje tabulka 5.18.

matice	*a*	matice	*a*
s820	1,661	s838_1	1,204
c1908	1,208	s5378_3	1,56
s420.1_1	1,214	s9234.1_1	1,192
s420.1_4	1,238	s13207.1_3	1,051
s641_1	1,055	s38417_1	1,092

Tabulka 5.19 – Zrychlení optimalizace při použití 1-bodového křížení v GCMA2

Protože zrychlení není zas tak výrazné, nebylo provedeno porovnání optimalizované verze GCMA2 s ostatními Column-Matching metodami. Stejnou optimalizaci je možné provést i v případě GCMA1.

6 Messy Genetic Algorithm pro Column-Matching problém

Messy Genetic Algorithm (MGA) byl uveden v kapitole 3.4. MGA se od klasických genetických algoritmů liší v následujícím:

1) MGA používá proměnnou délku řetězců (chromozómů), které můžou být vzhledem k řešenému problému nedostatečně nebo nadbytečně specifikovány (z angl. underspecification, resp. overspecification)
2) MGA používá na místo operátorů křížení, které pracují s pevnou délkou chromozómu, operátory *cut* (v překl. uřízni) a *splice* (v překl. spoj)
3) MGA rozděluje evoluční vývoj do dvou fází: prvotní (z angl. primordial) fáze a juxtapoziční fáze
4) MGA používá *competitive template* (dále pouze šablona) pro vyzdvižení lepších stavebních bloků

Tyto 4 odlišnosti byly více diskutovány v kapitole 3.4. Zároveň byl zmíněn pojem level-wise processing, což je postupování algoritmu po úrovních, které značí délku stavebního bloku.
MGA může být popsáno následujícím pseudokódem:

```
MGA {
  k = 1;                              // zacina se delkou stavebniho bloku 1
  template = RandomTemplate();        // na pocatku je template vytvorena nahodne

  do {
    // vytvoreni nahodne populace stavebnich bloku delky k
    InitPopulation(k);
    // ohodnoceni bloku na zaklade competitive template
    Evaluation(template);

    // prvotni faze
    do {
      Selection();                    // rekombinace
      ReducePopulation();             // zmenseni populace
    } while ( PrimordialCondition() );

    // juxtapozicni faze
    do {
      CutAndSplice();
      Selection();
      Evaluation(template);
    } while { JuxtapositionalCondition() };

    // sablonou pro dalsi uroven se stava nejlepsi nalezene reseni teto urovne
    template = population.Best();
     k = k + 1;

  } while ( MGACondition() );
}
```

Ukončení cyklů v pseudokódu (ukončení prvotní, juxtapoziční fáze a celého algoritmu) je provedeno symbolicky jako PrimordialCondition(), JuxtapositionalCondition() a MGACondition(). Celý MGA bývá ukončen pokud po několik vnitřních cyklů nedojde ke zlepšení nejlepšího nalezeného řešení. Prvotní fáze je obyčejně ukončena tím, že se populace zmenší na předem vypočítanou velikost, kdy by v populaci mělo být dostatečné zastoupení dobrých stavebních bloků. Ukončení juxtapoziční fáze je určeno podle očekávaného času konvergence k lokálnímu optimu (pokud jsme schopni toto odvodit). Přesným určením těchto podmínek pro problém s kardinalitou 2

se zabývá např. [Gol2].

Nýní se pojďme zabývat otázkou, jestli je možné převést MGA na Column-Matching problém. První odlišností je velká kardinalita našeho problému rovna $2 \cdot n$, kvůli které není možné volit binárního zakódování. Nabízí se použití stejného kódování jako v genetickém algoritmu zkoumaném v kapitole 4, resp. 5, tedy pomocí polí rank, matches a NM. S tím souvisí další odlišnost, a to ta, že pole rank je permutací. Zde se nabízí jednoduché řešení. Naším úkolem je udržovat permutační vlastnost pole rank, proto musíme prověřit všechny operace MGA, kde dochází ke změnám v chromozómech, resp. stavebních blocích. K našemu prospěchu dochází v MGA ke změnám ve stavebních blocích pouze při inicializaci náhodné populace a při aplikaci operátoru cut a splice. Vytvořit stavební blok, ve kterém se nebudou opakovat geny je jednoduché a kolizní geny, které vzniknou během operací cut a splice můžeme řešit stejně jako nadbytečnou specifikaci, tzn. uvažovat pouze ty, které se v chromozómu vyskytují nejvíce vlevo. Toto řešení je sice jednoduché, nicméně skrývá se za ním nebezpečí v podobě menšího počtu genů, které potomek získá od druhého rodiče (většina genů nebude uvažována, protože budou způsobovat kolizi). Složitější řešení je uvedeno v [Knj1], ve kterém je navržen FMGA pro permutace, využívající zakódování pomocí reálných čísel. Na základě výše uvedeného můžeme říci, že vysoká kardinalita ani permutační vlastnosti našeho problému nebrání vytvoření MGA pro Column-Matching problém.

V rámci návrhu MGA je nutné také určit, jakým způsobem řešit nedostatečnou specifikaci chromómu a z ní vyplývající nutnost ohodnocovat stavební bloky, resp. vypočítávat jejich fitnes. Jak jsem již uvedl v kapitole 3.4, ohodnocení stavebních bloků může být v MGA řešeno dvojím způsobem:

1) problém, resp. kódování, umožnuje ohodnocení nedostatečně specifikovaného chromozómu, aniž bychom znali chybějící (nespecifikované) geny
2) nespecifikované geny jsou doplněny z šablony, která je lokálním optimem z předchozí úrovně běhu MGA

První varianta bývá méně častá. V našem případě je zřejmé, že fitnes stavebního bloku nemůže vycházet z počtu spárování, protože pak by fitnes bylo úměrné délce stavebního bloku a stavební bloky stejné délky by nedokázalo porovnat. Porovnání stavebních bloků (schémat) různé délky bylo řešeno v kapitole 4.2, kde byl odvozen potenciál kvality kombinace na základě podílu jedniček v matici B, které v matici zůstanou po spárovávní. Bohužel v testech GCMA2 (kap. 5.7.8) bylo odhaleno, že potenciál kvality nepřináší lepší výsledky, a proto by jeho použití pravděpodobně nebylo vhodné ani v MGA.

Standardní řešení využívá šablony, ze které jsou doplňovány nespecifikované geny. Předpokládejme, že bychom vypočítávali fitnes stejným způsobem jako v genetických algoritmech z kapitol 4 a 5, tzn. fitnes by bylo primárně závislé na celkovém počtu spárování. Algorimus výpočtu by vybíral páry podle pořadí pole rank a prováděl by testy na přiřazení řádek. Pokud by algoritmus přiřazení řádek skončil chybou, došlo by ukončení výpočtu. Bylo by možné využít i modifikace metodou zahazování sloupců, kdy by se v případě chyby během přiřazení řádek odsunul gen na konec chromozómu a dále by se testoval gen následující.

Představme si nyní, jakým způsobem by vypadal stavební blok délky 2. Příklad takového stavebního bloku by ve formátu jazyka LISP mohl vypadat takto: ((5 2 5 0) (3 6 4 1)) a zastupoval

by pár (2, 5) pozitivně spárovaný na 5. pozici pole rank a pár (6 4) negativně spárovaný na 3. pozici pole rank. Zde je zřejmé, že výsledek ohodnocení takového stavebního bloku by byl závislý především na umístění genů v poli rank (v našem případě se jedná o pozice 5 a 3). Pokud by například tyto pozice byly větší, než hodnota columns_matched šablony, pak by ohonocení bylo zcela nevypovídající, protože by téměř určitě bylo fitnes stavebního bloku stejné jako fitnes šablony. Naopak pokud by byly pozice v poli rank nízké, měl by stavební blok s velkou pravděpodobností značně horší fitnes než šablony. Domnívám se tak na základě zkušeností z testů genetických algoritmů v kapitole 4 a 5 a s vědomím, že šablona je lokálním optimem.

Druhou možností, jak by mohl stavební blok vypadat je, že by v něm chyběla informace o pozici v poli rank. Příklad takového stavebního bloku by byl např. ((2 5 0) (6 4 1)), který by reprezentoval páry (2, 5) pozitivně párovaný a (6, 4) párovaný negativně. Stavební blok by mohl být při výpočtu fitnes předsunut na první pozice pole rank. Zde vyvstává otázka, jestli takovéto ohodnocení bude vůči všem stavebním blokům rovnocenné. Je zřejmé, že fitnes stavebních bloků krátké délky bude silně závislé na aktuální šabloně.

Během návrhu MGA pro Column-Matching problém vznikly nové otázky, které vyžadují dlouhodobější analýzu a experimenty. Možné řešení by mohlo být s použitím stejného způsobu kódování jako v případě genetických algoritmů z kapitol 4 a 5, tedy pole rank, matches a NM. Speciální důraz by měl být kladen na nalezení správného způsobu ohodnocování stavebních bloků. Nejvhodnější se zdá být varianta stavebních bloků, kde není specifikována pozice v poli rank, které budou při výpočtu fitnes předsouvány před aktuální šablonu. Více informací o MGA (FMGA) můžete nalézt v [Gol1], [Gol2]. Modifikace FMGA pro permutační problémy je popsána v [Knj1].

7 Závěr

Hlavním cílem této práce bylo prozkoumat možnosti genetických algoritmů při návrhu výstupního dekodéru, který je součástí generátoru testovacích vzorků pro mixed-mode BIST. Za tímto účelem byly navrženy, implementovány a otestovány 2 algoritmy (GCMA1 a GCMA2), vycházející z nejjednoduššího modelu genetických algoritmů, a to SGA. Přestože oba algoritmy vycházely ze SGA, ukázalo se jako nejvhodnější použít kódování s velmi odlišnými vlastnostmi.

Zcela zásadní rozdíl je v kardinalitě obou kódování. Zatímco kardinalita kódování SGA, tedy binárního pole, je rovna 2, kardinalita obou verzí GCMA je $2 \cdot n$, kde n je počet sloupců vstupní kódové matice, který se v reálných problémech pohybuje mezi 20 – 2000. Z takto vysoké kardinality vyplývá, v porovnání se SGA, i obrovský nárust stavového prostoru. Druhý velký rozdíl spočívá ve vlastnostech navrženého chromozómu, kvůli kterým není možné nad ním aplikovat standardní genetické operátory. Proto byly navrženy zcela specifické operátory křížení a mutace. Pro GCMA byla navíc implementována další, velmi specifická metoda zahazování sloupců, která mění chromozóm během výpočtu fitnes. Při samotném výpočtu fitnes byla využita heuristika popsaná ve [Fis1], využívající blokující matici B se složitostí $O(r \cdot p \cdot s)$, která určila charakteristickou vlastnost GCMA, a to dlouhou dobu výpočtu fitnes.

Časově nejnáročnější částí této práce bylo jednoznačně testování. U největších instancí problému, kde doba výpočtu fitnes přesahovala 5 sekund, bylo zřejmé, že výsledná řešení budou kompromisem mezi jejich kvalitou a dobou běhu algoritmu. Proto byly při testech nastaveny takové podmínky, že doba běhu algoritmu nesmí přesáhnout 3 hodiny, přitom výsledky by měly být dosaženy mezi 1-2 hodinami. Během testů jsem se snažil najít optimální hodnoty parametrů genetického algoritmu. Ty se však ukázaly být velmi závislé na instanci problému, proto byly alespoň odhadnuty doporučené intervaly pro nastavení jednotlivých parametrů.

Zatímco jednodušší verze GCMA1 využívala jedinou heuristiku, a to metodu zahazování sloupců, pro GCMA2 jich bylo implementováno hned několik. Na základě analýzy vstupních matic byly definovány vztahy mezi sloupci v testovací matici T. Sloupce v této matici byly rozděleny na hlavní a vedlejší. Vedlejší sloupce byly v průběhu celého algoritmu zastupovány sloupci hlavními a díky tomu se značně snížila délka chromozómu, resp. doba výpočtu fitnes. Vedle toho byla zavedena pravidla, která vylučovala takové geny, které nemohly vést k lepším řešením. Tato pravidla si vyžádala řadu změn genetických operátorů. Poslední změnou oproti GCMA1 bylo použití vlastnosti potenciálu kvality stavebního bloku při výpočtu fitnes, která však, jak se během testů ukázalo, neměla žádný přínos. Závěrečná verze GCMA2 s potenciálem kvality nepočítá, naopak byl GCMA2 optimalizován tak, že došlo k mírnému zkrácení doby výpočtu.

Výsledky obou verzí GCMA byly porovnány s existujícím algoritmem Thorough Search (viz. [Fis1]) a náhodným generátorem řešení. Náhodný generátor byl při srovnání vybaven stejnými heuristikami (např. metoda zahazování sloupců), kterých využívala i daná verze GCMA. Při srovnání s generátorem náhodných řešení byla prokázána účinnost genetických operátorů. Výsledky však napověděly, že lepší metodou pro řešení Column-Matching problému je Thorough Search, přestože u některých instancí problému může jak GCMA1, tak GCMA2 dosáhnout lepších výsledků. Volba GCMA1 nebo GCMA2 je také závislá na vstupní matici. GCMA2 však znamená větší jistotu dosažení kvalitních výsledků.

Na závěr jsem se pokusil nastínit problémy spojené s návrhem MGA, resp. FMGA, pro Column-Matching problém. Během návrhu však vznikly nové otázky, které si vyžadují dlouhodobější analýzu a experimenty.

Přestože se navržené algoritmy GCMA1 a GCMA2 pouze přiblížily výsledkům algoritmu Thorough Search, neznamená to, že genetické algoritmy nejsou pro Column-Matching problém vhodné. Pravděpodobně se však vždy budeme potýkat s vysokou kardinalitou kódování tohoto problému a dlouhou dobou výpočtu fitnes. Na tuto studii je možné navázat několika směry, které se zde pokusím shrnout:

- Realizovat MGA, resp. FMGA. V tomto případě bude pravděpodobně velmi náročný návrh algoritmu, který bude často vyžadovat hodně nestandardní řešení a vyžádá si řadu experimentů. Možné je také hledat inspiraci v Ordering Messy Genetic Algorithm (OMEGA), viz. [Knj1]
- Začlenit do vývojového procesu metodu simulovaného ochlazování, které by regulovalo některé vlastnosti genetického algoritmu (např. míra mutace, selekční tlak) a umožnilo by algoritmu vyváznout z lokálního extrému.
- Pokusit se identifikovat přínosné metody a operace, kterých využívá GCMA1 a GCMA2 a na základě toho sestrojit další verzi GCMA. Jedna z možností je například implementovat do GCMA1 pravidla urdžující logickou konzistenci chromozómu. Tato verze by mohla dosáhnout lepších výsledků než GCMA2, protože by nepoužívala zastupování vedlejších sloupců hlavními.

V neposlední řadě je možné využít poznatků z kapitoly 4.2, která se zabývala vlastnostmi sloupců vstupní testovací matice, pro vylepšení současné verze algoritmu Thorough Search. Nabízí se zde následující možnosti:

- Provést optimalizaci výpočtu fitnes. Tak jak je algoritmus popsán ve [Fis1], po vytvoření nového páru se provádí kontrola přiřazení řádek. Na základě doplňování sloupců bychom však mohli o záporném výsledku rozhodnout i bez provádění této kontroly. Dosažené zrychlení (zpomalení) by bylo pravděpodobně závislé na vstupní matici a je tedy obtížné jej odhanout.
- Aplikovat heuristiku pro vybírání sloupců matice T. Nejdříve by se spočítaly závislosti mezi sloupci matice T. Vyběr sloupců by byl řízen sestupně podle počtu závislých sloupců, které by byly párovány stejně jako vybraný sloupec. V případě rovnosti počtu závislých sloupců by o výběru rozhodoval počet neúplně určných hodnot ve sloupci. Tato heuristika by mohla dosáhnout kvalitního výsledku hned při prvním běhu.

8 Použité zdroje

[Brg85] F. Brglez, H. Fujiwara. A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortan, Proc. of International Symposium on Circuits and Systems, pp. 663-698, 1985

[Brg89] F. Brglez, D. Bryan and K. Kozminski. Combinational Profiles of Sequential Benchmark Circuits, Proc. of International Symposium of Circuits and Systems, pp. 1929-1934, 1989

[Chow1] Chowdhury M, Yun Li: Messy Genetic Algorithm Based New Learning Method for Structurally Optimised Neurofuzzy Controllers

[Fis1] Fišer P.: Column-Matching Based Mixed Mode Bist Technique, Ph.D. Thesis, ČVUT Praha, 2007

[Fis2] Fišer P.:Minimization of Boolean Functions, MSc. Thesis, ČVUT Praha, 2002

[Gol1] Goldberg D.E., Deb K., Kargupta H., Harik G.: Rapid, Accurate Optimization of Difficult Problems Using Fast Messy Genetic Algorithms, University of Illinois

[Gol2] Goldberg D.E, Deb K., Korb B.: An Investigation of Messy Genetic Algorithms, University of Alabama, 1990

[Gol3] Goldberg D.E, Sastry K.: A Practical Schema Theorem for Genetic Algorithms Design and Tuning, Illinois Genetic Algorithms Laboratory, 2001

[Har1] Harik G.R.: Learning Gene Linkage to Efficiently Solve Problems of Bounded Difficulty Using Genetic Algorithms, Doctoral Thesis, University of Michigan, 1997

[Knj1] Knjazew D., Goldberg D.E.: OMEGA – Ordering Messy GA: Solving Permutation Problems with the Fast Messy Genetic Algorithm and Random Keys, Illinois Genetic Algorithms Laboratory, 2000

[Mer1] Merkle L.D., Gates H.G. Jr., Lamont G.B., Scalability of an MPI-Based Fast Messy Genetic Algorith, New York, 1998

[Smi1] Smith R.E.: Genetic and evolutionary systems

[Sta1] Šťáva M.: Vestavěné dignostické prostředky, Diplomová práce, ČVUT Praha, 2004

[Whi1] Whitley D.: A Genetic Algorithm Tutorial

[BOOM] http://service.felk.cvut.cz/vlsi/prj/BOOM/ - BOOM – The BOOlean Minimizer

[WIKI1] http://en.wikipedia.org/wiki/Genetic_algorithms - Genetic Algorithm – Wikipedie

[GA] http://cs.felk.cvut.cz/~xobitko/ga/ – Introdution to genetic algorithms with Java applets, Marek Obitko, ČVUT, 1998

[ILLINOIS] http://www.illigal.uiuc.edu/web/ - Illinois Genetic Algorithms Laboratory

A Seznam použitých zkratek

BIST - Build-In Self-Test
CUT - Circuit Under Test - testovaný obvod
TPG - Test Pattern Generator - generátor testovacích vzorků
PRPG - Pseudo-random Pattern Generator - generátor pseudo-náhodných vzorků
LFSR - Linear Feedback Shift Register
ATPG - Automatic Test Pattern Generator
OD - Output Decoder - výstupní dekodér
SGA - Simple Genetic Algorithm
MGA - Messy Genetic Algorithm
FMGA - Fast Messy Genetic Algorithm
BOOM - Boolean Minimizer
GCMA1 - Genetic Column-Matching Algorithm, první verze gen. algoritmu
GCMA2 - Genetic Column-Matching Algorithm, modifikovaná verze gen. algoritmu

B Nastavení parametrů programu

Vstupem programu je soubor s parametry pro genetický Column-Matching algoritmus. Jednotlivé parametry jsou v souboru definovány ve tvaru:
[název parametru] [hodnota]

V názvu a hodnotě parametru nesmí být mezera ani žádný jiný prázdný znak. V souboru nemůžou být komentáře. Všechny parametry jsou povinné a jejich pořadí je pevně dané. Zde je uvedeno pořadí a význam všech parametrů:

matrix-C – relativní cesta k matici C
matrix-T – ralativní cesta k matici T
estimate-population-size(0/1) – odhad počáteční populace na základě vz. 4.2 (1 = ano, 0 = ne)
max-duration(min) – maximální doba běhu programu v minutách
population-size – velikost populace
max-generations – maximální počet generací
bad-generations-to-terminate(pt) – podm. ukončení, počet gen. během kterých musí dojít k zlepšení
selection-parameter(ps) – parametr selekce, tzn. velikost turnaje
use-threshold – definuje, jestli GA má použit thresholding
crossover-method – metada křížení, 1 = 1-bodové křížení, 0 nebo 2 = uniformní křížení
crossover-probability(pc) – pravděpodobnost křížení
use-discarding – použití metody zahazování sloupců (1 = ano, 0 = ne)
mutation-matches-probability(pm) – podíl mutace pole matches
mutation-rank1-probability(pr1) – podíl mutace pole rank – vložení do permutace
mutation-rank2-probability(pr2) – podíl mutace pole rank – prohození sousedů permutace

Ukázku vstupního souboru s parametry najdete na přiloženém CD (data/parameter_file).

C Obsah přiloženého CD

Soubory:

index.html	- výchozí HTML stránka projektu
readme.txt	- základní informace o GCMA
install.txt	- postup instalace GCMA1, resp. GCMA2
data/parameter_file	- ukázka vstupního souboru pro GCMA1 i GCMA2
data/vypis.out	- výpis části populaci, o které se pojednává v kapitole 4.9.2
data/scm.stat	- statistika Scm, na kterou se odkazuji v kapitole 5.7.1
data/sr.stat	- statistika Sr, na kterou se odkazuji v kapitole 5.7.1

Adresáře:

bin	- přeložené binární programy gcma1 a gcma2 (přeloženo na OS Linux Fedora 6)
data	- různé
html	- HTML dokumenty
matrix	- testovací matice použité v diplomové práci
src	- zdrojové kódy programu GCMA1 a GCMA2
tests	- provedené testy
text	- obsahuje PDF verzi diplomové prace