

# Evoluční návrh elektronických obvodů

Design electronics circuits by  
means of evolution method

Roman Strakoš

---

Bakalářská práce  
2005



Univerzita Tomáše Bati ve Zlíně  
Fakulta technologická

---

***Vložit oficiální zadání bakalářské práce***

Souhlasím s tím, že s výsledky mé práce může být naloženo podle uvážení vedoucího bakalářské práce a ředitele ústavu. V případě publikace budu uveden jako spoluautor.

Prohlašuji, že jsem na celé bakalářské práci pracoval samostatně a použitou literaturu jsem citoval.

Ve Zlíně, 01. 09. 2005

.....

podpis

## RESUMÉ

Hlavním cílem této bakalářské práce je potvrdit, že evoluční algoritmy je možné použít na řešení a optimalizaci návrhu elektronických obvodů. V první části této práce pronikneme hlouběji do tajů Genetického programování, kde si blíže popíšeme jeho princip. V druhé části se pak seznámíme s Analytickým programováním a v třetí části se zaměříme na Gramatické algoritmy, které jsou nedílnou součástí této obsáhlé skupiny. V praktické části pak budou prezentovány vybrané výsledky ze simulací a ty pak vyhodnoceny.

The main point of this bachelor thesis is confirm, that evolution algorithms is possible applicated to solving and optimalisation design of electronics circuits. In first part of thesis we are going to deeper to mystery of Genetic programming. In second part we are going to better understanding to Analytical programming. And in third part of this thesis we are going to focus to Gramatical algorithms, which are inseparable part group of Evolution algorithms. And finally in practise part will be presented selected results from simulations and evaluated.

# OBSAH

|                                                                             |           |
|-----------------------------------------------------------------------------|-----------|
| <b>ÚVOD .....</b>                                                           | <b>6</b>  |
| <b>1 SEZNÁMENÍ SE S PROBLEMATIKOU GENETICKÉHO PROGRAMOVÁNÍ .....</b>        | <b>7</b>  |
| <b>1.1 Genetické programování v porovnání s genetickými algoritmy .....</b> | <b>7</b>  |
| <b>1.2 Prvky genetického programování .....</b>                             | <b>7</b>  |
| 1.2.1 Účelová funkce v genetickém programování .....                        | 7         |
| 1.2.2 Reprezentace jedinců.....                                             | 8         |
| 1.2.3 Požadavky na prvky stromové reprezentace .....                        | 8         |
| <b>1.3 Algoritmus genetického programování.....</b>                         | <b>9</b>  |
| 1.3.1 Generování počátečních stromových struktur .....                      | 10        |
| 1.3.2 Rekombinační operátory v genetickém programování.....                 | 10        |
| 1.3.3 Složitost generování programů .....                                   | 11        |
| <b>2 SEZNÁMENÍ SE S PROBLEMATIKOU ANALYTICKÉHO PROGRAMOVÁNÍ.....</b>        | <b>12</b> |
| <b>2.1 Úvod.....</b>                                                        | <b>12</b> |
| <b>2.2 Symbolická regrese .....</b>                                         | <b>12</b> |
| <b>2.3 Analytické programování.....</b>                                     | <b>12</b> |
| 2.3.1 Původ názvu Analytické programování.....                              | 13        |
| 2.3.2 Hlavní principy analytického programování.....                        | 13        |
| 2.3.3 Tři verze analytického programování.....                              | 15        |
| 2.3.4 Analytické a genetické programování – podobnosti a rozdíly.....       | 15        |
| <b>3 GRAMATICKÁ EVOLUCE .....</b>                                           | <b>16</b> |
| <b>4 APLIKACE EVOLUČNÍCH ALGORITMŮ NA ŘEŠENÍ VYBRANÝCH PROBLÉMŮ.....</b>    | <b>18</b> |
| 4.1 Řešení úlohy „Světelná signalizace“ pomocí SOMA algoritmu.....          | 20        |
| 4.2 Řešení úlohy „Tepelná regulace“ pomocí SOMA algoritmu .....             | 29        |
| 4.3 Řešení úlohy „Řízení vlakových výhybek“ pomocí SOMA algoritmu.....      | 38        |
| <b>ZÁVĚR.....</b>                                                           | <b>49</b> |
| <b>SEZNAM POUŽITÉ LITERATURY .....</b>                                      | <b>50</b> |
| <b>SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....</b>                              | <b>51</b> |
| <b>SEZNAM OBRÁZKŮ.....</b>                                                  | <b>52</b> |
| <b>SEZNAM TABULEK .....</b>                                                 | <b>53</b> |

## ÚVOD

Náplní této bakalářské práce je potvrzení, že Evoluční algoritmy jsou vyjímečně silný a vhodný nástroj na řešení složitých problémů. Pro svou práci jsem si vybral pak Design elektronických obvodů, kde chci prokázat, že i v tomto směru aplikací mají Evoluční algoritmy dveře otevřeny.

Metody a postupy, které se využívají v této době při vytváření složitých elektronických obvodů jsou natolik složité, že design některých elektronických obvodů může trvat řádově až několik týdnů až dnů. Při aplikaci této metody můžeme získat vhodný výsledek již během několika málo minut (záleží ovšem na složitosti řešeného problému). Příklady, které jsou v této práci uvedeny byli vyhodnoceny vždy do 20 minut, při 50ti simulacích. Což jak víme není dostatek času ani na klidný oběd.

Existuje mnoho metod jak navrhovat elektronické obvody od numerických až po grafické. Pro svou bakalářskou práci jsem si však vybral metodu novou, která není zatím příliš využívána pro tento účel.

Vzhledem k tomu, že jsem delší dobu pracoval ve firmě ONSEMICONDUCTOR CZ, kde jsme se touto problematikou zabývali, mohu jen potvrdit, že takto silný nástroj ve všeobecné praxi chybí a rád bych se proto v příští práci spíše zaměřil na vytvoření všeobecného návodu jak využít např. Evoluční algoritmy v optimalizaci sériové výroby.

Všechny simulace byly vytvořeny v programu Mathematica. Zapojení do blokového zapojení, schématického zapojení a ověření funkčnosti zapojení bylo provedeno v programu ElectronicWorkbench.

## 1 SEZNÁMENÍ SE S PROBLEMATIKOU GENETICKÉHO PROGRAMOVÁNÍ

Genetické programování ( 1 ) je nástroj pro automatické vytváření programů řešících složité problémy.

### 1.1 Genetické programování v porovnání s genetickými algoritmy

Genetické programování vychází v principu z genetických algoritmů ( 1 ), od nichž se liší ve dvou směrech:

- a) Účelem, pro nějž se používají. Na rozdíl od genetických algoritmů, jejichž základní aplikací, je hledání hodnot nezávislých proměnných, pro něž jistá účelová funkce nabývá svého optima, je genetické programování určeno pro automatickou syntézu programů, algoritmů či rozhodovacích postupů.
- b) Reprezentací jedinců představujících potenciální řešení problému, který chceme automaticky hledaným postupem řešit. Na rozdíl od genetických algoritmů, které reprezentují jedince pomocí chromozomů ve tvaru lineárních řetězců, genetické programování používá na místě chromozomů nejčastěji stromové struktury ( 1 ).

### 1.2 Prvky genetického programování

Podobně jako v genetických algoritmech i zde budou operace „selektce“ a „změna“, které vytváří nové jedince. Schéma genetického programování ( 1 ) bude tedy obdobné algoritmu obecně používanému téměř všemi druhy evolučních výpočetních postupů. Méně evidentní je však operace vyhodnocení jedinců. Volba účelové funkce, jež ohodnocuje kvalitu jednotlivých potenciálních řešení, je jedním z klíčových problémů genetického programování.

#### 1.2.1 Účelová funkce v genetickém programování

Předpokládejme, že máme mechanismus tvorby potencionálních řešení problému, který bude generovat funkce jako vhodné (a algebraicky přípustné) pomocí kombinací operací +, -, \*, / proměnné a popř. několika konstant. Bezprostředním úkolem je ohodnocování takto vzniklých výrazů a v kontextu s genetickými postupy použití tohoto ohodnocení jako měřítka kvality (fitness) každého jedince ( 1 ).

### 1.2.2 Reprezentace jedinců

Hledaným řešením v genetickém programování jsou výrazy, programy či algoritmy. Ty lze zapisovat ve tvaru interpretovaných stromových struktur.

Takový strom je konstruován ze dvou množin symbolů:

- a) z množiny „T“ terminálních symbolů představujících ve stromu listové uzly
- b) z množiny „F“ neterminálních symbolů (funkcí) užitých ve stromu jako vnitřní (nelistové uzly)

Z hlediska modelovaného výrazu zobrazují neterminální prvky funkce či operace, terminální symboly pak vstupní proměnné, konstanty nebo výstupy senzorů. Terminály mohou být i funkce bez argumentů mající nějaký vedlejší efekt.

Během evoluce působí na stromy reprezentující jednotlivé výrazy aplikační genetické rekombinační operátory. Ty většinou spočívají v určitých modifikacích částí stromů spolu s jejich kombinováním či „převěšováním“ podstromů ( 1 ).

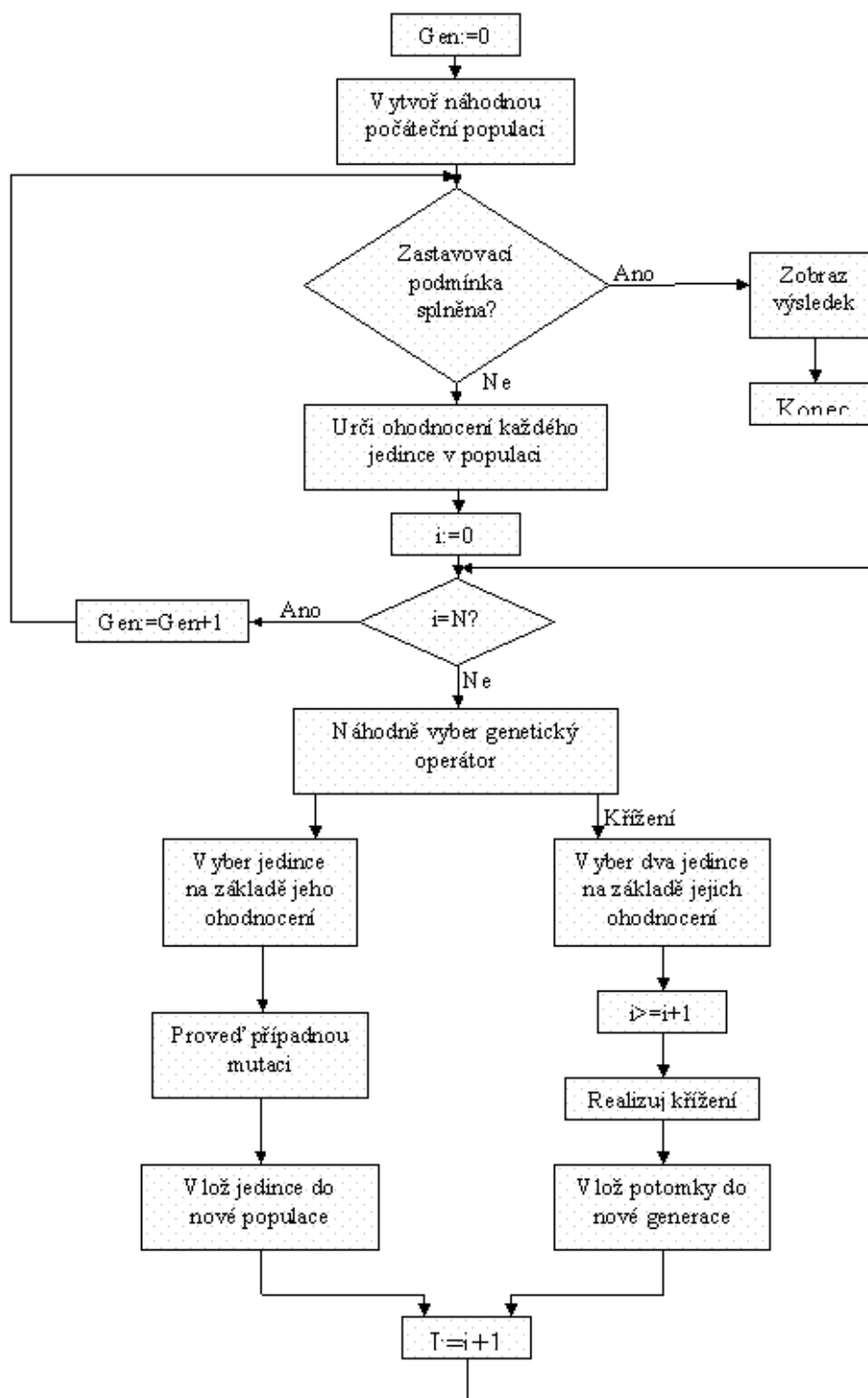
### 1.2.3 Požadavky na prvky stromové reprezentace

Abychom předešli případným problémům stromové reprezentace musíme nadefinovat několik požadavků:

- a) uzavřenost množiny funkcí a množiny terminálů – je nutné, aby výstup libovolné funkce a terminálu mohl figurovat jako kterýkoliv argument jakékoliv jiné funkce
- b) úplnost a postačitelnost množiny funkcí a množiny terminálů – množina funkcí a terminálů musí být volena adekvátně k řešené úloze. (Pokud by např. v úloze o konstrukci logické funkce obsahovala množina „F“ pouze operace AND a OR a neobsahovala by operaci NOT, pak nelze úlohu vyřešit.) ( 1 )

### 1.3 Algoritmus genetického programování

I v genetickém programování jsou populace jedinců ohodnocovány. K ohodnocení se nejčastěji používá řízená selekce a rekombinační operátory, tedy prvky a akce velmi podobné akcím v genetických algoritmech. ( 1 )



Obr. 1. Algoritmus genetického programování

### 1.3.1 Generování počátečních stromových struktur

Je známo, že vhodně vygenerovaná počáteční populace může výrazně zefektivnit celý evoluční proces a kvalitu dosaženého řešení, zvláště pak to platí pro genetické programování.

Náhodným postupem vygenerujeme počáteční populaci obsahující  $\text{PopSize} = 1000$  stromů – analytických předpisů. Dá se očekávat, že některé z těchto stromů budou nesmírně hluboké a budou tak představovat složité, avšak do značné míry nesmyslné výrazy. Jiné stromy budou degenerované a budou tvořeny jediným uzlem – terminálním symbolem. Musí se proto přistoupit k určitému usměrnění. První komponent tohoto usměrnění se týká hloubky generovaných stromů, tedy složitosti na počátku generovaných výrazů. V současnosti existují dvě metody:

- a) růstové generování stromu – je neřízené a vytváří stromy s nepravidelným vzrůstem, kdy jednotlivé větve jsou různě vysoké
- b) úplné generování stromu – vytváří stromy pravidelně (rovnoměrně rozložené)

J. Koza (1) ve své práci pak doporučuje metodu označovanou „lineárně půl na půl“ (ramped half and half). Kde pak v každé skupině s danou hloubkou stromu je vždy polovina vytvářena úplným a polovina růstovým generováním.

### 1.3.2 Rekombinační operátory v genetickém programování

- a) Křížení – do operace vstupují dva jedinci a výsledkem je jeden nebo dva potomci. Standardně se postupuje tak, že v každém z rodičovských chromozomů se náhodně vybere po jednom uzlu a podstromy vycházející z těchto uzlů se vzájemně vymění.
- b) Mutace – probíhá tak, že se ve zvoleném jedinci náhodně vybere jeden uzel a podstrom z tohoto uzlu vycházející se zcela nahradí nově vygenerovaným stromem, tedy podobným způsobem jako při tvorbě stromů v počáteční populaci.
- c) Permutace – je operace, která míchá pořadím argumentů náhodně zvoleného vnitřního uzlu
- d) Inverze – je operace, která má dosáhnout takového uspořádání genů, aby geny, které tvoří stavební bloky, byly co možná nejblíže u sebe.

- e) Editace – tato operace se používá pro automatické zjednodušování vyvíjených stromů. Editace aplikuje předem definovaná pravidla na redukci kódu. Pravidla mohou být úlohově nezávislá i speciálně navržená pro řešenou úlohu. Důležitým efektem, je to, že zhušťuje zápis reprezentovaného řešení.
- f) Zapouzdření – je operace, která slouží k automatickému definování stavebních bloků, celá operace spočívá v tom, že náhodně vybereme vnitřní uzel stromu a jeho podstrom prohlásíme za novou dále nedělitelnou funkci.
- g) Decimování – používá se při řešení problémů, u kterých může dojít po vygenerování počáteční populace k situaci, kdy většina jedinců reprezentuje zcela bezcenné řešení a pouze několik málo jedinců je použitelných.

### 1.3.3 Složitost generování programů

Je velice důležité, aby složitost generovaného programu byla co nejjednodušší:

- a) program který je jednodušší z hlediska složitosti zápisu, je i snáze interpretovatelný, a tím i přijatelnější pro uživatele
- b) je snaha zamezit generování potenciálních řešení, která jsou příliš rozsáhlá, a tak vyžadují i více mnohdy zbytečných výpočtů, potřebných pro jejich ohodnocení

Neúměrný nárůst velikosti zpracovávaných stromových struktur je totiž typickým problémem genetického programování. To vyplývá z podstaty genetického programování jakožto metody, která na rozdíl do klasických genetických algoritmů pracuje s jedinci proměnné délky a struktury. Konkrétně dochází k tomu, že generovaná řešení jsou nejenom různě složitá, resp. jednoduchá, ale mohou obsahovat i takové struktury (části kódu), které nepřispívají k ohodnocení daného řešení ( 1 ).

## 2 SEZNÁMENÍ SE S PROBLEMATIKOU ANALYTICKÉHO PROGRAMOVÁNÍ

### 2.1 Úvod

Analytické programování je nová metoda, která umožňuje řešení nejrůznějších problémů z oblasti symbolické regrese. Symbolická regrese byla poprvé uvedena v ( 1 ) a ( 2 ).

### 2.2 Symbolická regrese

Termín symbolická regrese (SR) reprezentuje proces, v kterém zaznamenaná data odpovídají matematické definici jako např.  $x^2+C$ ,  $\sin(x)+1/ex$  atd. Mezi matematiky je tento proces dobře znám a může být použit, když proces obdrží neznámá data. Po dlouhou dobu byla SR pouze doménou lidí, ale před několika desítkami let nastala éra počítačů. K dnešní době jsou používány dvě metody, které mohou být použity pro výpočet SR pomocí počítače. První z nich je genetické programování a druhá gramatické programování.

Myšlenka, jak řešit různé problémy použitím SR prostřednictvím evolučních algoritmů, byla představena v ( 1 ), kde autor využil genetický algoritmus pro genetické programování. Genetické programování je základem symbolické regrese, která by byla provedena evolučními algoritmy místo člověka. Schopnost řešit skutečné složité problémy byla potvrzena mnohokrát, a proto se dnes genetické programování dobře aplikuje např. při návrhu složitých integrovaných elektronických obvodů.

### 2.3 Analytické programování

Analytické programování ( 4 ) bylo inspirováno numerickými metodami v Hilbertových prostorech ( 4 ) a genetickým programováním. Principy analytického programování jsou někde mezi těmito dvěma filozofiemi. Z genetického programování byla převzata myšlenka o evolučním vytváření symbolických řešení, zatímco z Hilbertových prostor byla adaptována myšlenka funkčních prostorů a postavena z výsledných funkcí ve smyslu hledání procesu obvykle pomocí numerických metod jako jsou Ritz nebo Galerkin. Analytické programování

je založeno podobně jako genetické na souboru funkcí, operátorů a takzvaných terminálů, které jsou obvykle konstanty nebo nezávislé proměnné. Stručně řečeno, v analytickém programování se jedinec skládá z nenumerných výrazů (operátorů, funkcí, ...), jak bylo popsáno výše, které jsou v evolučním procesu reprezentovány jejich celočíselnými indexy. Tento index poté slouží jako ukazatel na soubor výrazů a analytické programování je použije ke sloučení výsledného funkčního programu pro vyhodnocení funkce.

### 2.3.1 Původ názvu Analytické programování

Až doteď existovalo pro symbolickou regresi pouze genetické programování. Protože analytické programování umožňuje používat jakékoliv evoluční algoritmy pro symbolickou regresi, význam krycích názvů pro všechny tyto evoluční algoritmy v rámci analytického programování je zcela jasný. Název analytické programování představuje použití evolučních algoritmů pro nalézání analytického řešení bez ohledu na použitý algoritmus. Pokud by se neuvedl název analytické programování, může se použít název podle použitého algoritmu tj. SOMA programování, DE programování, SA programování atd. Každý vidí, že by to bylo zmatené a komplikované. Toto je proto hlavní důvod, proč byl zvolen název analytické programování. ( 4 )

### 2.3.2 Hlavní principy analytického programování

Jádro analytického programování je založeno na souboru funkcí, operátorů a tzv. terminálů, které jsou obvykle konstanty nebo nezávislé proměnné. Toto nastavení proměnných je obvykle smíchané. Hlavní cíl analytického programování je slučovat vhodný program (matematický vzorec), který by byl co nejvíce vhodný pro vstupní data. Z tohoto důvodu byla manipulace s izolovanou množinou přijata do analytického programování.

Manipulace s izolovanou množinou (MIM) byla navržena v ( 3 ). V analytickém programování je MIM použita k vytvoření celočíselných indexů, které jsou použity v evolučním procesu jako alternativní jedinec, ovládaný v evolučním algoritmu metodou manipulace s celými čísly. Jedinec je v populaci vytvořen automaticky jako celočíselný jedinec jehož parametry rozsahu nemůžou přesáhnout mohutnost použitého souboru funkcí a terminálů.

Protože jsou zde dva hlavní soubory: soubor funkcí a terminálů a index tohoto souboru, pak zde máme pro úspěšnou symbolickou regresi jedinečné mapování mezi těmito soubory. Společně zde jsou s mapováním také speciální bezpečnostní podprogramy, které nás ujistí, že aktuálně vygenerované řešení nebude:

- a) patologickou funkcí (bez argumentů, smyčka)
- b) funkce s imaginární nebo reálnou částí (pokud není očekávaná)
- c) nekonečná funkce (dělení nulou)
- d) zamrzlá funkce (extrémní čas na získání její hodnoty až hodiny)
- e) atd.

Struktura souboru (nazývejme ji HFP – hlavní funkční prostor) je dělena na podsoubory podle počtu argumentů každé funkce. Např. podsoubor  $HFP_{all}$  představuje všechny funkce, operátory a terminály, zatímco  $HFP_{3arg}$  představuje všechny funkce, operátory jejichž počet argumentů je max. 3 zahrnující terminály, atd. nejmenší podsoubor je  $HFP_{0arg}$ , který představuje pouze terminály. Struktura tohoto podsouboru společně s unikátním mapováním v analytickém programování zajistí, že nebudou generovány žádné patologické funkce. Analytické programování je série operací, které vytvoří výslednou funkci z HFP shodně s jedincem z populace, který slouží jako celočíselný index pro HFP jehož vlastnost je jedinečná.

Analytické programování je v podstatě skupina po sobě jdoucích mapujících funkcí. Naneštěstí tento základní přístup může generovat patologické funkce. Z tohoto důvodu byla vytvořena některá vylepšení jako podmnožina HFP, které zajistí, že nebudou generovány patologické funkce. Hlavní myšlenka vylepšení vychází z filozofie, že když je požadováno více argumentů, než je možných (konec jedince se blíží), bude tato funkce nahrazena jinou se stejným indexovým ukazatelem z podmnožiny s nižším počtem argumentů. To samé se může stát, když například poslední argument pro funkci s jedním argumentem nebude terminál (funkcí s nula argumenty - počítám že terminál = konstanta).

HFP nemusí být nutně vytvořen z čistě matematických funkcí, ale také mohou být použity jiné uživatelsky definované funkce jako např. nějaké logické funkce. Funkce, které reprezentují nějaké prvky elektrických obvodů atd. ( 4 )

### 2.3.3 Tři verze analytického programování

V dnešní době existuje analytické programování ve třech verzích. Všechny tři verze potřebují stejný soubor funkcí, terminálů atd. pro programovou syntézu, podobně jako je použita v ( 1 ) První základní verze byla popsána v předchozí části a nazýváme ji základní analytické programování.

Druhá verze (meta analytické programování) je modifikací ve smyslu odhadu konstant. V porovnání se základním analytickým programováním, které manipuluje se stejným HFP jako genetické programování, v meta AP je generována pouze jedna nespecifikovaná konstanta „K“ místo náhodně generovaných konstant ( 3 ), která je po složení indexována K1, K2, ..., KN, ve vzorci a poté všechny KN jsou odhadnuty stejnou nebo různou evoluční metodou. Protože evoluční algoritmy pracují pod evolučními algoritmy (tj. EA master -> program -> K indexace -> EA slave -> odhad konstant) je tato verze nazývána analytické programování s meta evolucí – meta AP.

Poslední modifikace analytického programování je APnf, která je založena na APmeta, je taková, že místo jiného evolučního algoritmu je konstanta „K“ odhadnuta vhodnou nelineární metodou. Všechny tři modifikace byly použity v různých úrovních testování. ( 3 )

### 2.3.4 Analytické a genetické programování – podobnosti a rozdíly

Jelikož bylo analytické programování částečně inspirováno genetickým programováním jsou mezi nimi logicky jisté rozdíly a podobnosti. Za hlavní podobnost může být považován fakt, že analytické programování, stejně jako genetické, je schopno vytvořit symbolickou regresí. Z hlavního hlediska to znamená, že výstup z analytického programování je shodný se všemi důležitými simulacemi stejně jako výstup z genetického programování. Další podobnost je, že APbasic pracuje se stejným souborem (HFP) jako GP. Hlavní rozdíl mezi analytickým programováním je např. kódování jedinců (AP používá DSH, zatím co GP používá přímou reprezentaci v LISP ( 3 ), rozdílná struktura HFP (jenom jedna konstanta „K“) pro APmeta a APnf, použití nelineárního přidělování, nebo jiného evolučního algoritmu pro odhadování konstant, bezpečnostní procedury které zaručují syntézu nepatologických funkcí apod. ( 5 )

### 3 GRAMATICKÁ EVOLUCE

Jednou z nenovějších metod spadající do evolučních výpočetních technik je tzv. gramatická evoluce (2)(3). Jde o metodu, která má mnohé společné jak s genetickými algoritmy, tak s genetickým programováním. Ve skutečnosti je to vlastně kombinace obou těchto přístupů. S genetickým programováním má společný cíl a tím také oblasti použití – je to totiž nástroj pro automatické generování počítačových programů. Oproti genetickému programování je gramatická evoluce obecnější v tom, že je navržena tak, aby byla použitelná pro hledání programů v jakémkoliv jazyce. Přesněji řečeno v jakémkoliv jazyce, který může být popsán bezkontextovou gramatikou, resp. Backusovou – Naurovou formou (BNF). S tím je úzce spojena i reprezentace jedinců. Na rozdíl od stromů používaných v genetickém programování používá jednoduché jednobodové křížení a jednoduchou bodovou mutaci, což je obojí vlastní klasickým genetickým algoritmům.

Gramatika BNF popisuje jazyk formou produkčních pravidel, ve kterých vystupují terminály, tj. atomické symboly, a neterminály, které jsou dále rozvinuty v jeden nebo více neterminálů a terminálů. Pravidla mají pevnou strukturu, kde na levé straně vystupují jednotlivé neterminály a na pravé straně je rozvoj příslušného neterminálu pomocí neterminálů a terminálů. Každý neterminál může mít více alternativních pravidel pro expandování. Výsledná gramatika se pak použije pro dekódování lineárního chromozomu. Ten má v gramatické evoluci takovou funkci, že reprezentuje posloupnost pravidel tak, jak budou postupně aplikována během generování programu. V gramatické evoluci mají chromozomy proměnnou délku. Celkový řetězec je formálně členěn na osmibitové podřetězce kódující čísla 0 – 255, které se nazývají *kodony*. Tyto kodony jsou postupně čteny od začátku chromozomu a na základě jejich hodnoty je použito odpovídající pravidlo pro rozvinutí aktuálního nejlevějšího neterminálu. Vzhledem k tomu, že hodnota kodonu je ve většině případů větší než počet pravidel použitelných pro jednotlivé neterminály, je číslo pravidla, které se pro rozvinutí neterminálu použije, stanovenou pomocí výrazu:

$$\text{Pravidlo} = \text{kodon} \bmod \text{počet\_pravidel\_pro\_daný\_neterminál}$$

Čtení chromozomu pokračuje zleva doprava, dokud nedojde k jedné z následujících situací:

- 1) Program je hotov, tzn., že všechny neterminály už byly přepsány na terminály nebo
- 2) ještě je třeba rozvinout jeden nebo více neterminálů, ale už byl přečten poslední kodon chromozomu.

V prvním případě, pokud ještě zbývají v chromozomu nějaké nepoužité kodony, se jedná o tzv. přespecifikovaný chromozom. Tuto situaci nijak speciálně ošetřovat nemusíme, prostě zbytek chromozomu ignorujeme. Ve druhém případě ale musíme nějak dokončit odvození programu, jinak nebudeme schopni ohodnotit daného jedince. V takovém případě si můžeme pomoci například tak, že chybějící kodony budeme číst znovu od začátku chromozomu. Aby nedošlo k nekonečnému nebo příliš dlouhému cyklení, tak se používá omezení na počet průchodů chromozomem.

Díky mapování genotypu na genotyp je gramatická evoluce blíže přirozenému konceptu genetického kódování než kterýkoliv jiný evoluční algoritmus. Toto mapování je navíc redundantní, jedno pravidlo je reprezentováno více hodnotami genu. To umožňuje tzv. *tiché mutace*, kdy drobné změny v chromozomu nezpůsobí změnu programu. Standardně se v gramatické evoluci používá jednobodové křížení, které je aplikováno na chromozomy. Podle efektu, který má tento operátor na křížené stromové struktury rodičovských jedinců, se také nazývá vlnové křížení (ripple crossover). Křížení funguje tak, že si rodičovské řetězce vzájemně vymění sekvence bitů na náhodně zvoleným bodem křížení. Tato jednoduchá operace má však poměrně velký dopad na nově vygenerované jedince.

## 4 APLIKACE EVOLUČNÍCH ALGORITMŮ NA ŘEŠENÍ VYBRANÝCH PROBLÉMŮ

Pro představení řešení pomocí evolučních algoritmů jsem si vybral jednoduché logické funkce tak, aby bylo jasně zřetelné jak evoluční algoritmy fungují. Definoval jsem tři základní logické problémy:

- a) Světelná signalizace – do procesu vstupují tři proměnné a jeden výstup, které určují vhodnou kombinaci, tzn. stavy kdy bude svítit zelená a vozidlo může projet křižovatkou
- b) Řízení vlakových výhybek – do procesu vstupují 4 proměnné a jeden výstup, které ovlivní nastavení výhybek tak aby nedošlo k nehodě a vlak mohl bezpečně projet
- c) Tepelná regulace – do procesu vstupují tři proměnné a jeden výstup, řeší problematiku regulace teploty v bytě 2+1

Na těchto příkladech chci prokázat, že evoluční algoritmy nejsou vhodné pouze jako výborný nástroj pro nejrůznější analýzy matematických funkcí, ale také, a to zejména, pro řešení nejrůznějších technologických problémů, především pak v optimalizaci výrobních procesů, návrh elektronických obvodů, regulace apod.

Jak již víme, existuje celá řada evolučních algoritmů, které pro tuto simulaci můžeme využít. Jak jsem již uvedl výše, téměř všechny tyto metody jsou již prověřeny aplikací mnoha složitých problémů a navzájem byli nespočetně mnohokrát porovnány. Touto prací nechci opět porovnávat výhody a nevýhody jednotlivých metod, ale chci již přímo využít jednu z nich.

Metoda SOMA - Samo – Organizují se – Migrační – Algoritmus je mi osobně nejbližší z hlediska jednoduchosti provedení a tudíž i jejího případného zavedení do všeobecné technické praxe jako mocný nástroj pro optimalizaci nejrůznějších technologických procesů. V dnešní době neustálého zlepšování procesů je pak obzvláště důležité umět použít, využít a porozumět nástrojům, které nám umožní lépe a efektivněji vyhodnocovat data. Účelem této práce je tak přiblížit na jednoduchých příkladech vědu k praxi a naopak.

Cílem jednotlivých simulací je dokázat, že je možné za velice krátkou dobu získat informace (vhodné řešení – jedince), jejichž získání by nám standardní numerickou metodou mohlo trvat až několik hodin.

Z řádku „Solution“ jsem převedl výsledek z Boolovy algebry do blokového zapojení. Poté jsem pomocí programu ElectronicsWorkbench provedl návrh a následné ověření funkčnosti zapojení pomocí integrovaných obvodů.

#### 4.1 Účelová funkce

Účelová funkce byla definována jako suma jednotlivých absolutních výstupních programových hodnot minus celkový výstup.

$$F_{COST} = \sum_{i=1}^n ABS(PRG_{out}; -OUT;)$$

Každá jednotlivá řešená úloha bude řešena 50x pomocí předem definovaných podmínek:

GFS Definition (definice použitých operátorů, proměnné, funkce a konstanty)

BasicLogicZal = {

{And, 2, {{{}}}, {Or, 2, {{{}}}, {Nand, 2, {{{}}}, {Nor, 2, {{{}}}, {And, 2, {{{}}},

{Or, 2, {{{}}}, {Nand, 2, {{{}}}, {Nor, 2, {{{}}}

} ;

- množina definovaných funkcí a argumenty

LogVarZal = LogVar = {{A, 0, {{{}}}, {B, 0, {{{}}}, {C, 0, {{{}}}} ;

- konstanty, logické proměnné – množina terminálů – suboptimální pomocí nichž byli přidáváni nejvhodnější jedinci

## 4.2 Řešení úlohy „Světelná signalizace“ pomocí SOMA algoritmu

Tabulka I. Vstupní pravdivostní tabulka „Světelná signalizace“

| Vstup-výstup 1 | Vstup-výstup 2 | Vstup-výstup 3 | Output |
|----------------|----------------|----------------|--------|
| False          | False          | False          | False  |
| False          | False          | True           | False  |
| False          | True           | False          | False  |
| False          | True           | True           | True   |
| True           | False          | False          | False  |
| True           | False          | True           | True   |
| True           | True           | False          | False  |
| True           | True           | True           | True   |

Parametry algoritmu SOMA:

pocitadlo = 1 ; - začátek ohodnocení účelové funkce

PathLength = 3. ; - délka cesty

Step = 0.11 ; - délka kroku

PRT = .1 ; perturbační parametr

PopSize = 300 ; - velikost populace

Specimen = Table[{{Integer, {1, AllDim}}}, {i, 30}] ; - výběr jedince

Dim = Dimensions[Specimen][[1]] ; - rozměr jedince

Migrations = 30 ; - počet migrací

MinDiv = -.01 ; - minimální odchylka

**Vzor průběhu evoluce:**

Start of simulation No. 1 Time: 12 : 59 : 52.611925

Appended suboptimal solution with CV = 2 Time: 12 : 59 : 52.736657

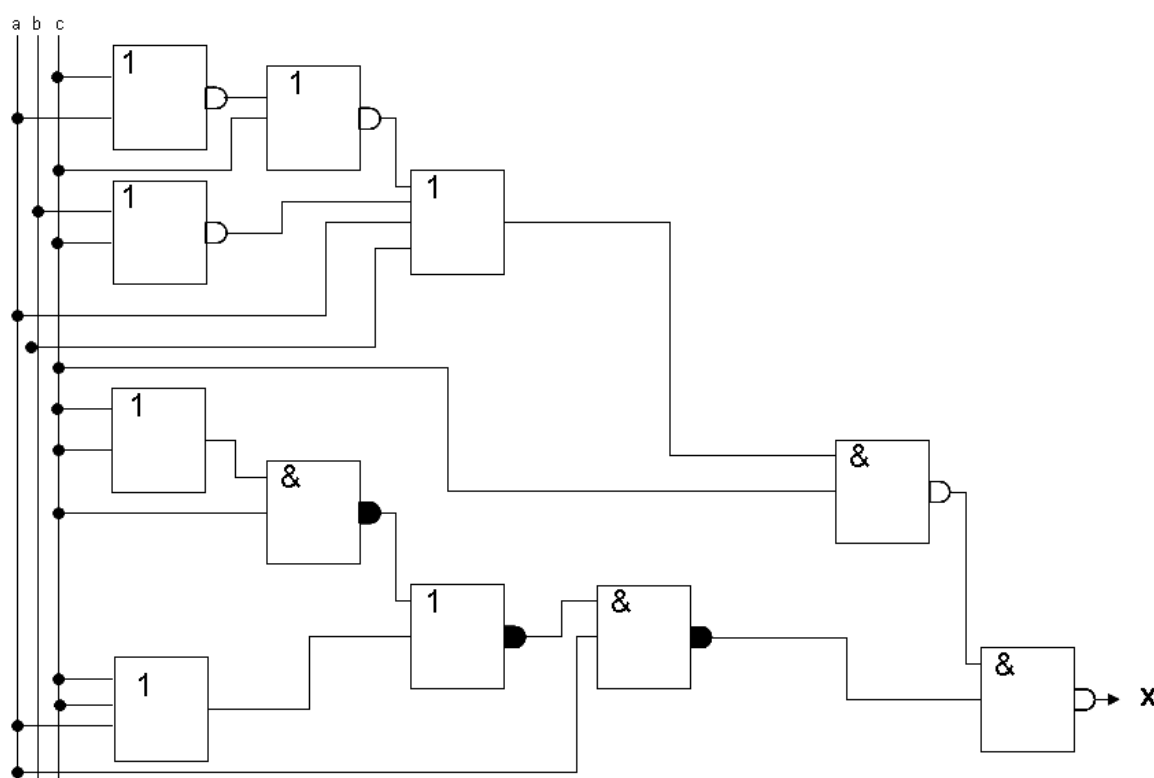
Appended suboptimal solution with CV = 1 Time: 12 : 59 : 52.762175

Number of cost function evaluations: 470

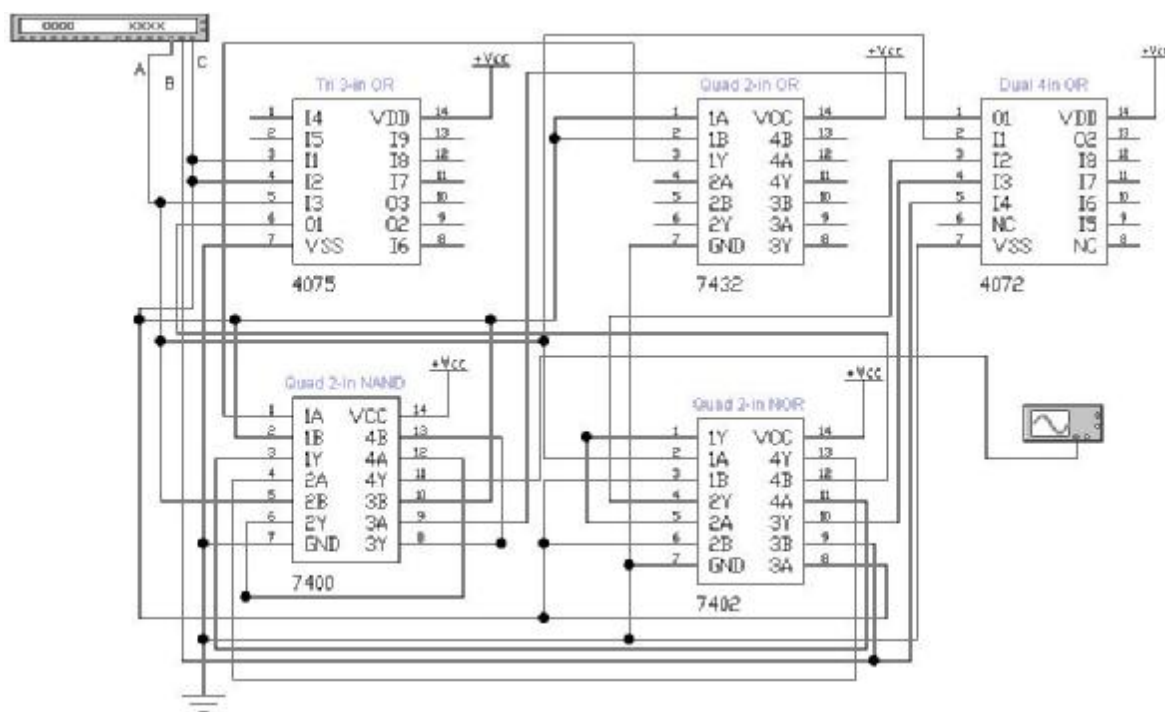
Cost value: 0

The best individual: {7, 7, 3, 11, 2, 9, 4, 2, 2, 6, 3, 4, 4, 9, 10, 11, 6, 2, 11, 4, 1, 3, 10, 6, 2, 11, 10, 5, 2, 11}

Solution:  $(C \bar{A} (((C \oplus A) \oplus C) \vee (B \oplus C) \vee A \vee B)) \bar{\wedge} (A \bar{\wedge} ((C \vee C \vee A) \oplus ((C \vee C) \bar{\wedge} C)))$



Obr. 2. Simulace 1 – Blokové zapojení „Světelná signalizace“



Obr. 3. Simulace 1 –Schématické zapojení pomocí integrovaných obvodů

Pro toto nalezené řešení jsem pro zapojení použil tyto integrované obvody:

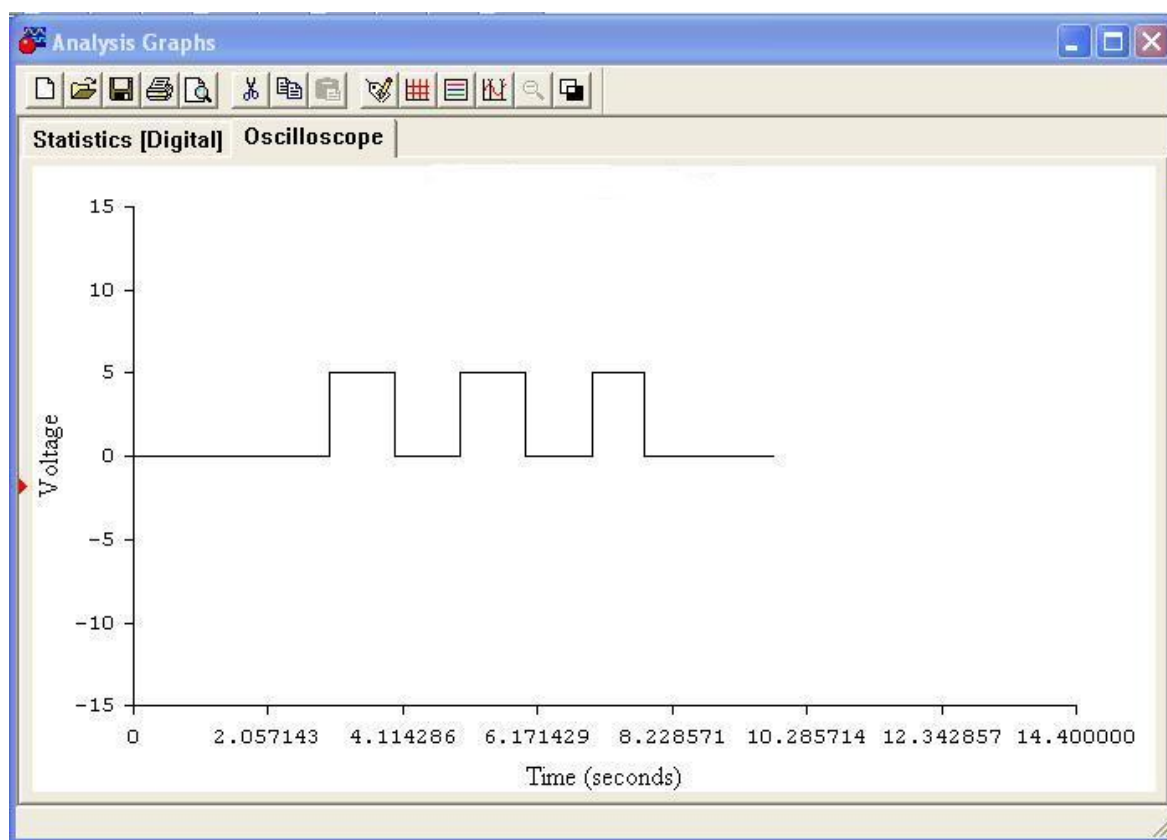
MC 4075 1x – tří vstupý OR

MC 7400 1x – čtyř vstupý NAND

MC 7432 1x – čtyř vstupý OR

MC 7402 1x – čtyř vstupý NOR

MC 4072 1x – osmi vstupý OR



Obr. 4. Simulace 1 – Ověření funkčnosti zapojení integrovaných obvodů

Ověření funkčnosti zapojení podle Obr. 3 a Tabulka I odpovídá požadovanému výstupu. Tzn. řešení nalezené pomocí algoritmu je správné.

Start of simulation No. 2 Time: 13 : 023.375901

Appended suboptimal solution with CV = 4 Time: 13 : 023.483415

Appended suboptimal solution with CV = 3 Time: 13 : 023.504295

Appended suboptimal solution with CV = 1 Time: 13 : 023.521962

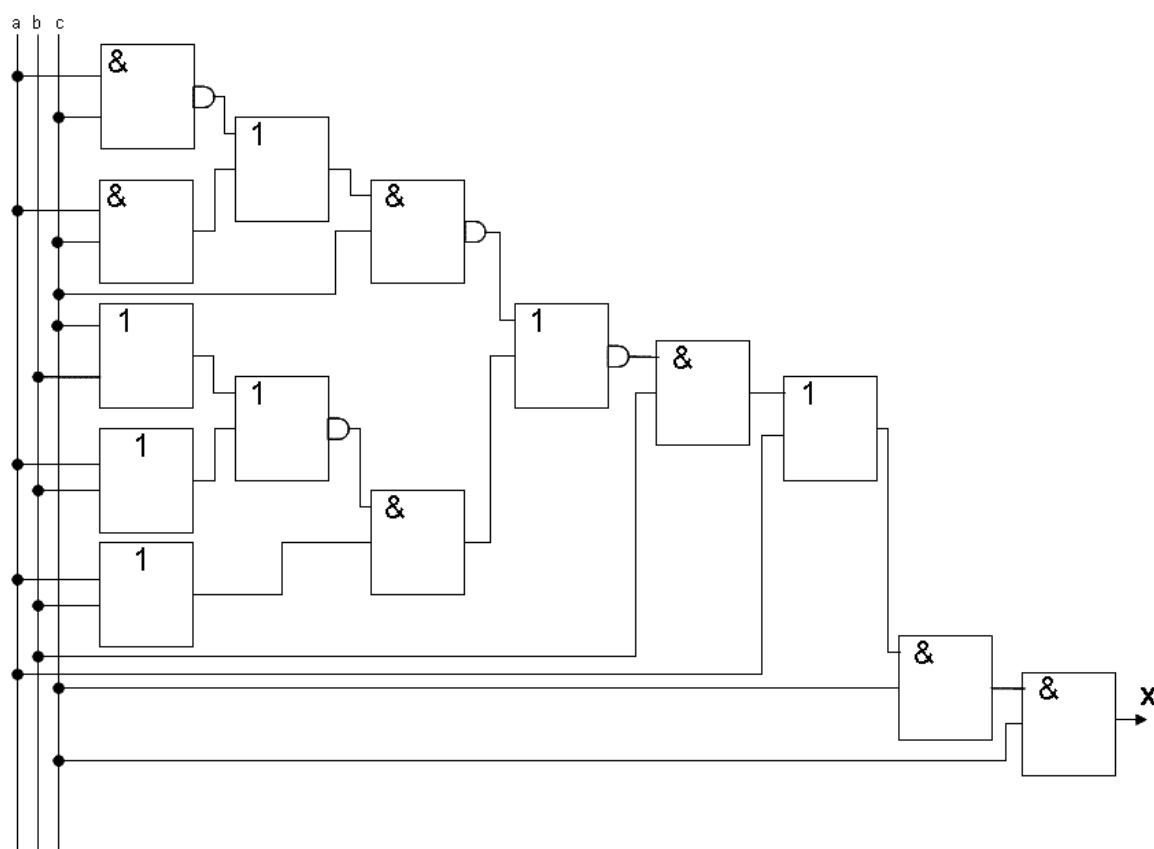
Number of cost function evaluations: 4154

Cost value: 0

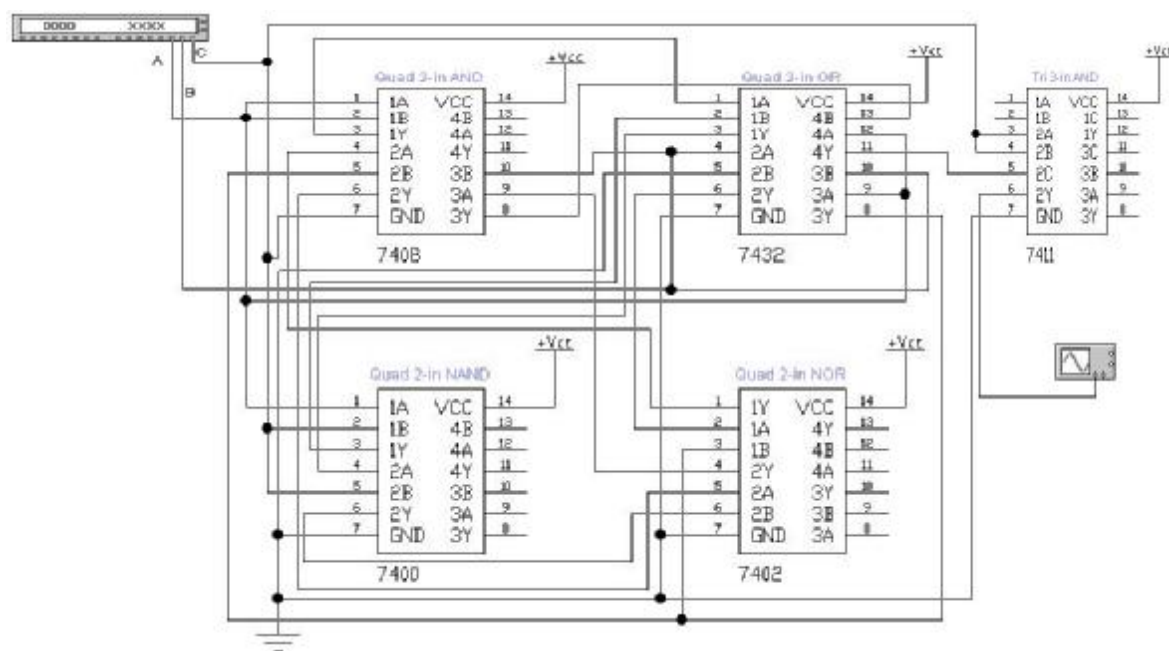
The best individual: {5, 6, 5, 9, 1, 11, 11, 10, 8, 7, 5, 2, 11, 8, 2, 3, 5, 2, 6, 2, 3, 7, 9, 2, 6, 6,

3, 2, 8, 2}

Solution:  $(A \vee (B \wedge (((A \bar{A} C) \vee (A \wedge C)) \bar{A} C) \bar{\vee} (((C \vee B) \bar{\vee} (A \vee B)) \wedge (A \vee B)))) \wedge C \wedge C$



Obr. 5. Simulace 2 – Blokové zapojení „Světelná signalizace“



Obr. 6. Simulace 2 – Schématické zapojení pomocí integrovaných

Pro toto nalezené řešení jsem pro zapojení použil tyto integrované obvody:

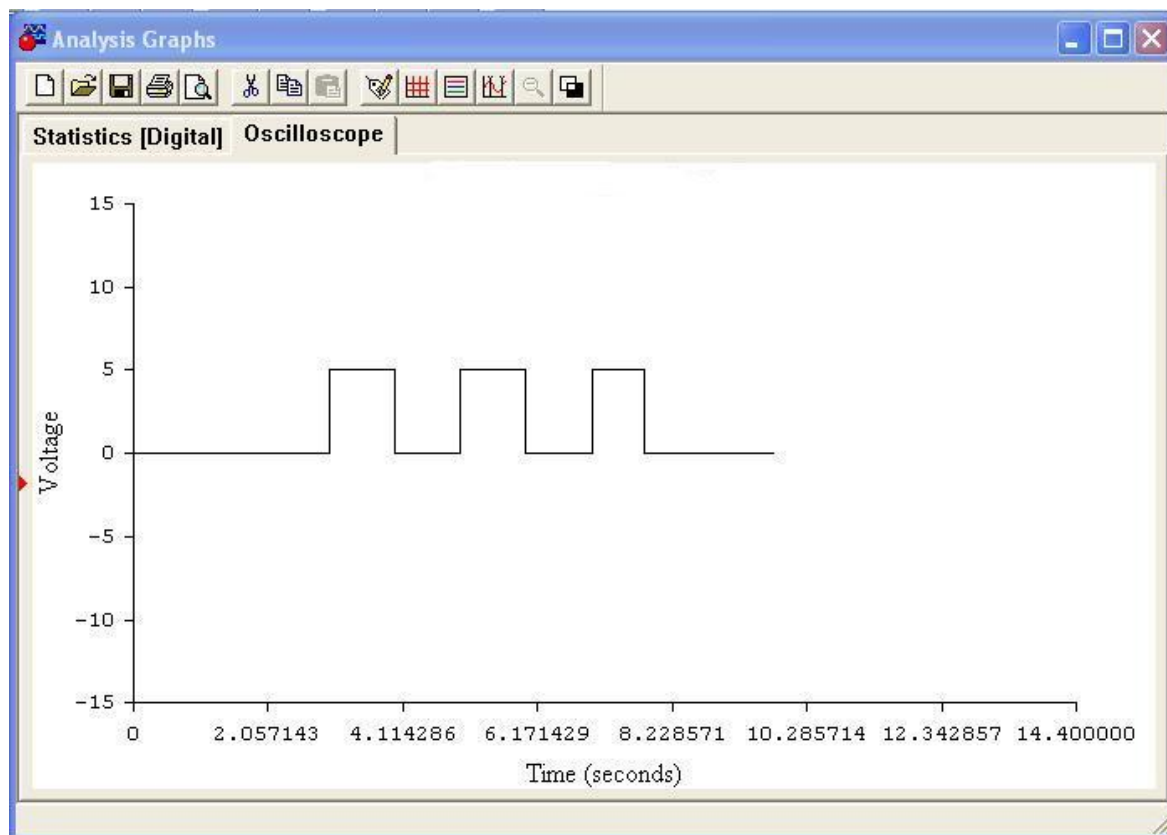
MC 7408 1x – čtyř vstupů AND

MC 7400 1x – čtyř vstupů NAND

MC 7432 1x – čtyř vstupů OR

MC 7402 1x – čtyř vstupů NOR

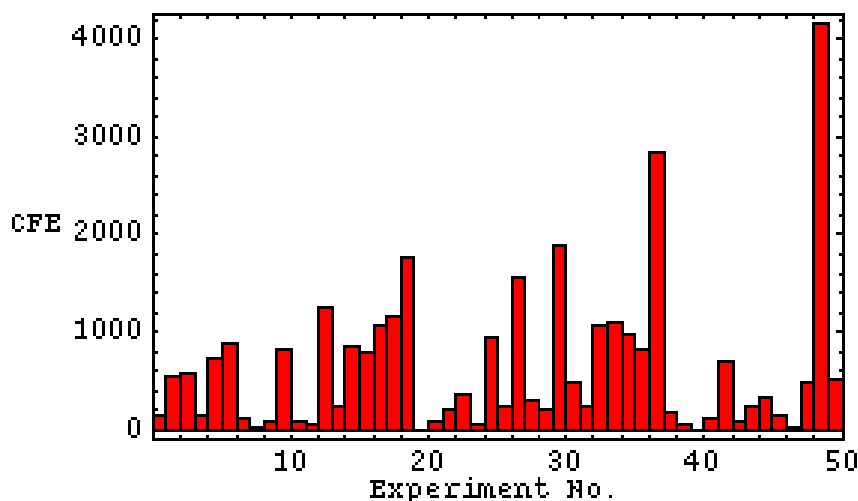
MC 7411 1x – tří vstupů AND



Obr. 7. Simulace 2 – Ověření funkčnosti zapojení pomocí IO

Ověření funkčnosti zapojení podle Obr. 6 a Tabulka I odpovídá požadovanému výstupu. Tzn. řešení nalezené pomocí algoritmu je správné.

### Grafické znázornění souhrnných výsledků 50-ti simulací

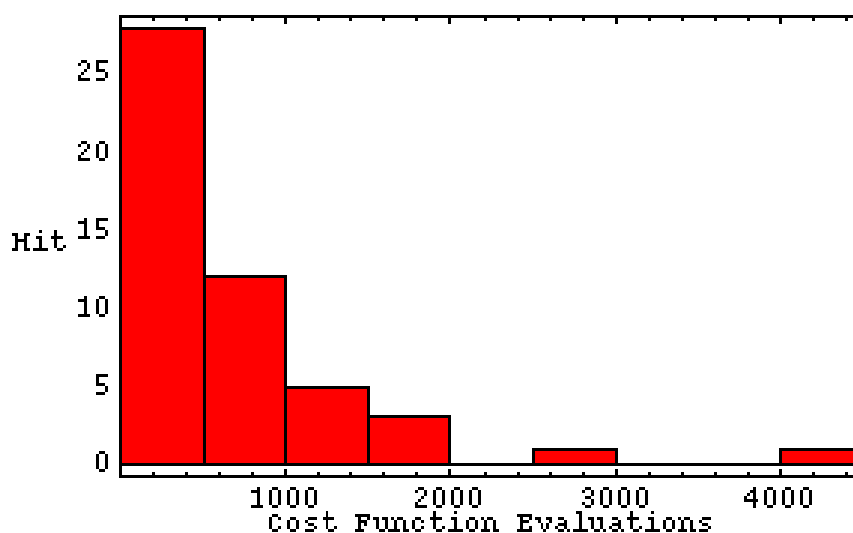


Obr. 8. Přehled hodnot CFE pro jednotlivé experimenty – „Světelná signalizace“

Dle grafického znázornění jde zřetelně rozpoznat, které experimenty dosáhly nejnižší a nejvyšší hodnoty CFE (účelové funkce).

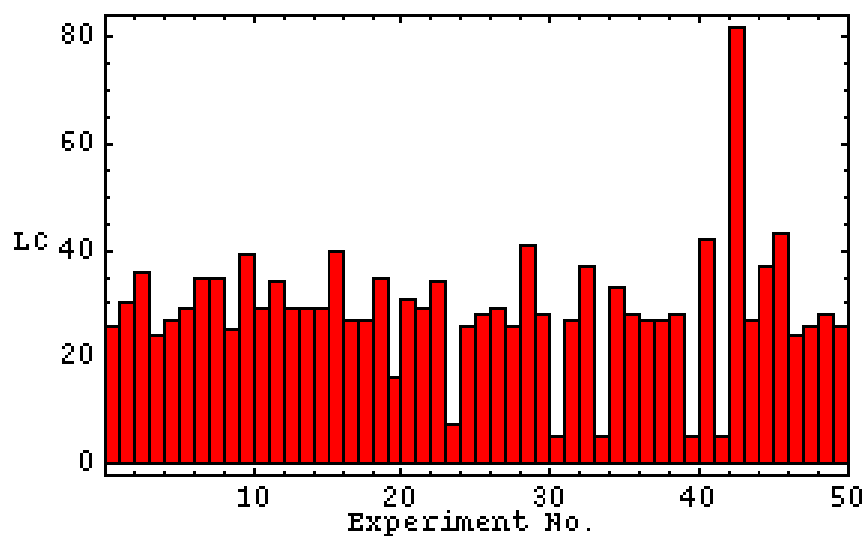
Hodnoty CFE :

{158, 555, 578, 151, 722, 888, 133, 34, 82, 829, 88, 46, 1236, 244, 849, 790, 1066, 1169, 1774, 3, 76, 207, 372, 57, 931, 243, 1557, 290, 210, 1901, 480, 236, 1071, 1105, 982, 829, 2848, 180, 70, 7, 108, 712, 82, 230, 317, 148, 23, 470, 4154, 519}



Obr. 9. Histogram – Četnost výskytu hodnot CFE v intervalech – „Světelná signalizace“

Hit – tato hodnota udává četnost výskytu v jednotlivých intervalech. Jak je podle histogramu naprosto zřejmé tak 70% hodnot CFE se vyskytuje v intervalu 0 až 600.



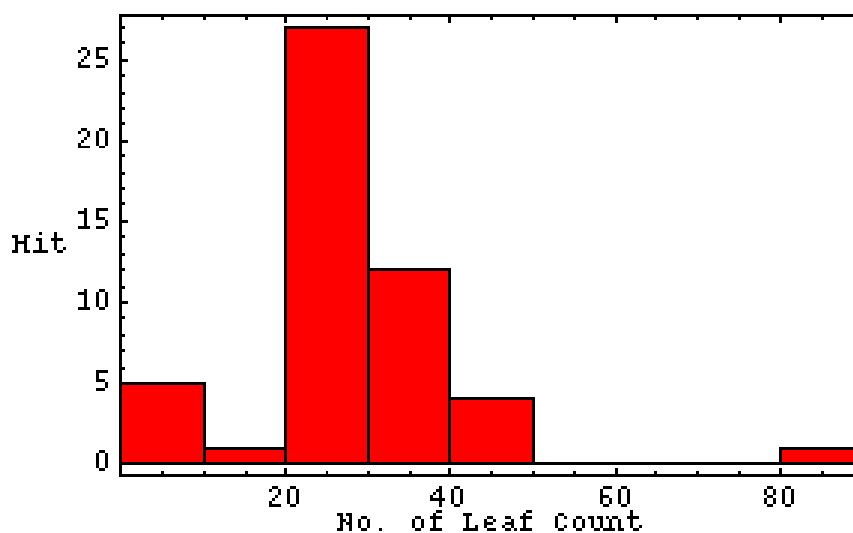
Obr. 10. Hodnoty Leaf count vůči opakovanému experimentu – „Světelná signalizace“

LC – Leaf count – počet operátorů, konstant, které byly použity pro zápis výsledného řešení

$$\text{Avg} = \text{CFE} / 50 = 636.2$$

$$\text{Mn} = \min [\text{CFE}] = 3$$

$$\text{Mx} = \max [\text{CFE}] = 4154$$



Obr. 11. Histogram – Četnost výskytu hodnot Leaf count v intervalech – „Světelná signalizace“

$$\text{Avg} = (\text{Plus}@@ \text{lfcnt3}) / 50 = 28.84$$

$$\text{mn} = \min [\text{lfcnt3}] = 5$$

$$\text{mx} = \max [\text{lfcnt3}] = 82$$

### 4.3 Řešení úlohy „Tepelná regulace“ pomocí SOMA algoritmu

Tabulka II. Vstupní pravdivostní tabulka „Tepelná regulace“

| Vstup-výstup 1 | Vstup-výstup 2 | Vstup-výstup 3 | Output |
|----------------|----------------|----------------|--------|
| False          | False          | False          | False  |
| False          | False          | True           | False  |
| False          | True           | False          | False  |
| False          | True           | True           | True   |
| True           | False          | False          | True   |
| True           | False          | True           | True   |
| True           | True           | False          | True   |
| True           | True           | True           | True   |

Parametry algoritmu SOMA:

pocitadlo = 1 ;

PathLength = 3. ;

Step = 0.11 ;

PRT = .1 ;

PopSize = 300 ;

Specimen = Table[{{Integer, {1, AllDim}}}, {i, 30}] ;

Dim = Dimensions[Specimen][[1]] ;

Migrations = 30 ;

MinDiv = -.01 ;

**Vzor průběhu evoluce:**

Start of simulation No. 2 Time: 12 : 19 : 24.611585

Appended suboptimal solution with CV = 2 Time: 12 : 19 : 24.719701

Appended suboptimal solution with CV = 1 Time: 12 : 19 : 24.748308

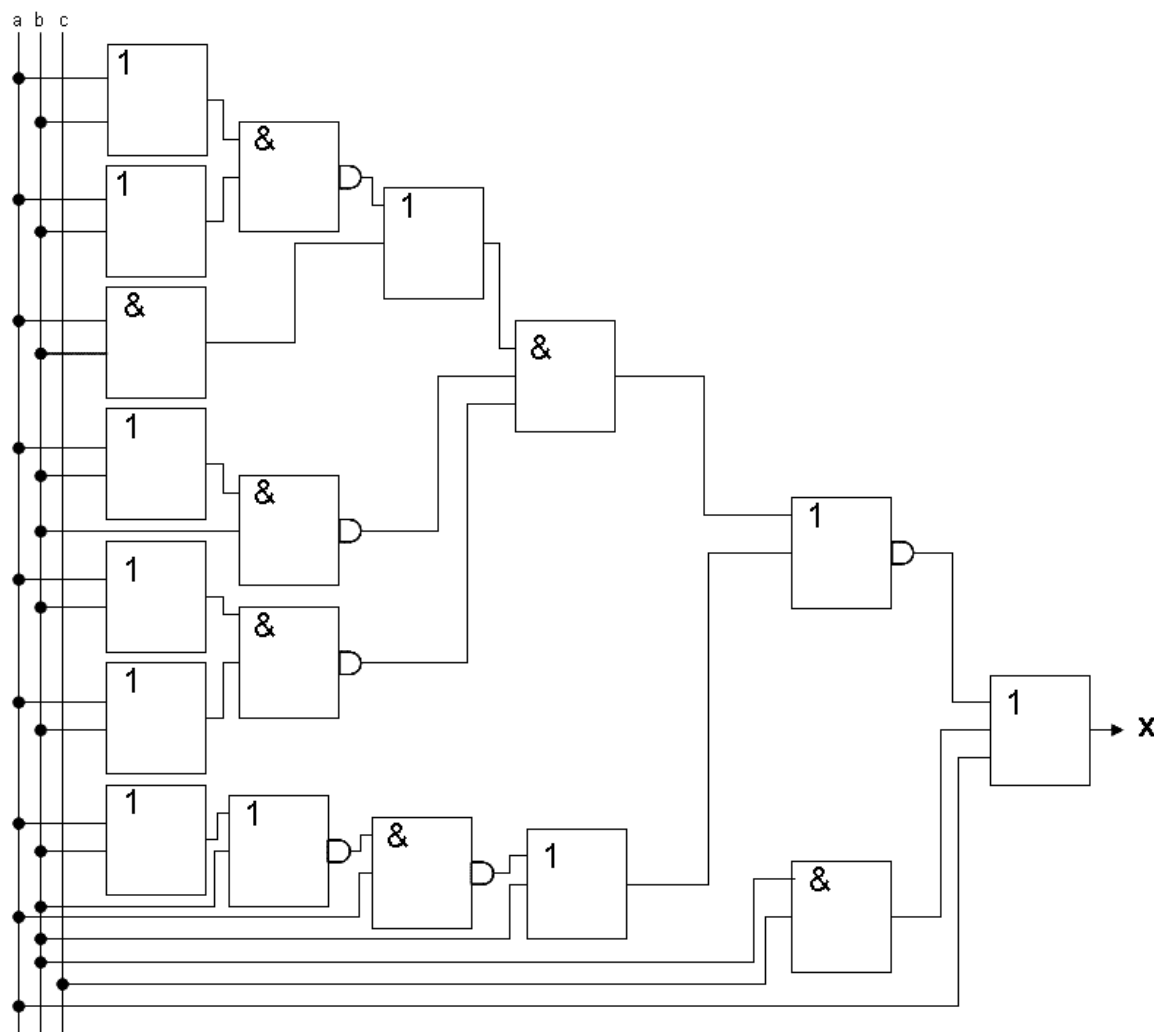
Number of cost function evaluations: 2529

Cost value: 0

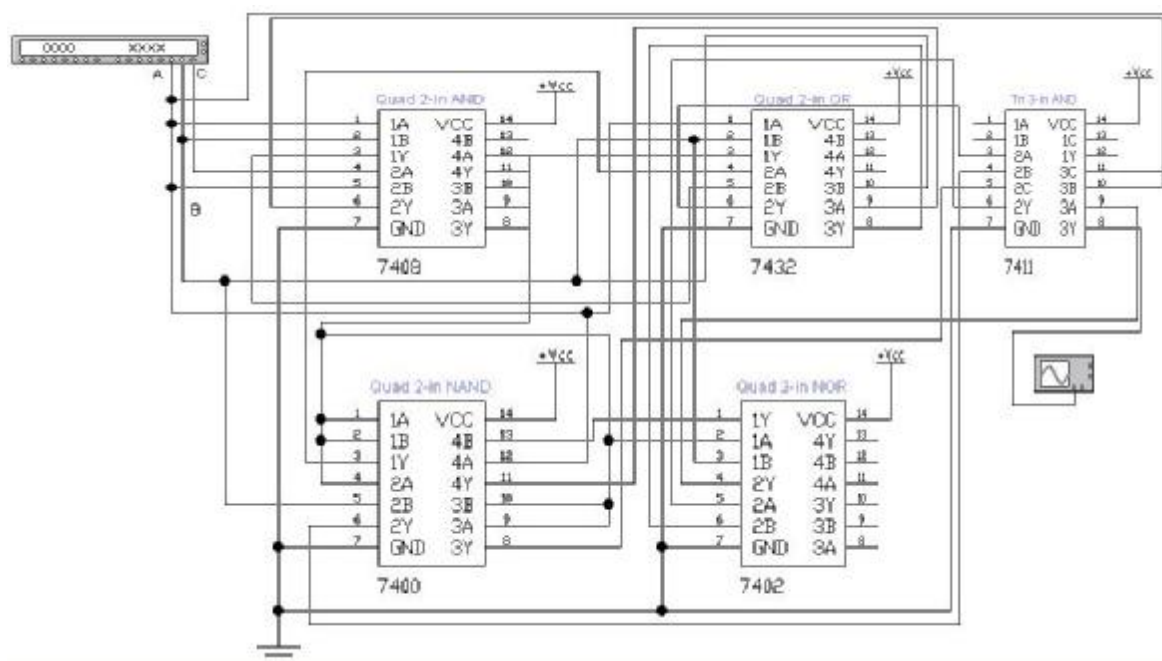
The best individual: {6, 6, 5, 9, 4, 10, 11, 1, 2, 2, 1, 7, 10, 7, 5, 7, 7, 4, 7, 11, 5, 7, 3, 3, 6,  
11, 6, 1, 3, 3}

Solution:

$A \vee (((((A \vee B) \wedge (A \vee B)) \vee (A \wedge B)) \wedge (B \wedge (A \vee B)) \wedge ((A \vee B) \wedge (A \vee B))) \vee (((A \vee B) \vee B) \wedge A) \vee B)) \vee (B \wedge C)$



Obr. 12. Simulace 2 – Blokové zapojení „Tepelná regulace“



Obr. 13. Simulace 2 – Schématické zapojení pomocí integrovaných obvodů

Pro toto nalezené řešení jsem pro zapojení použil tyto integrované obvody:

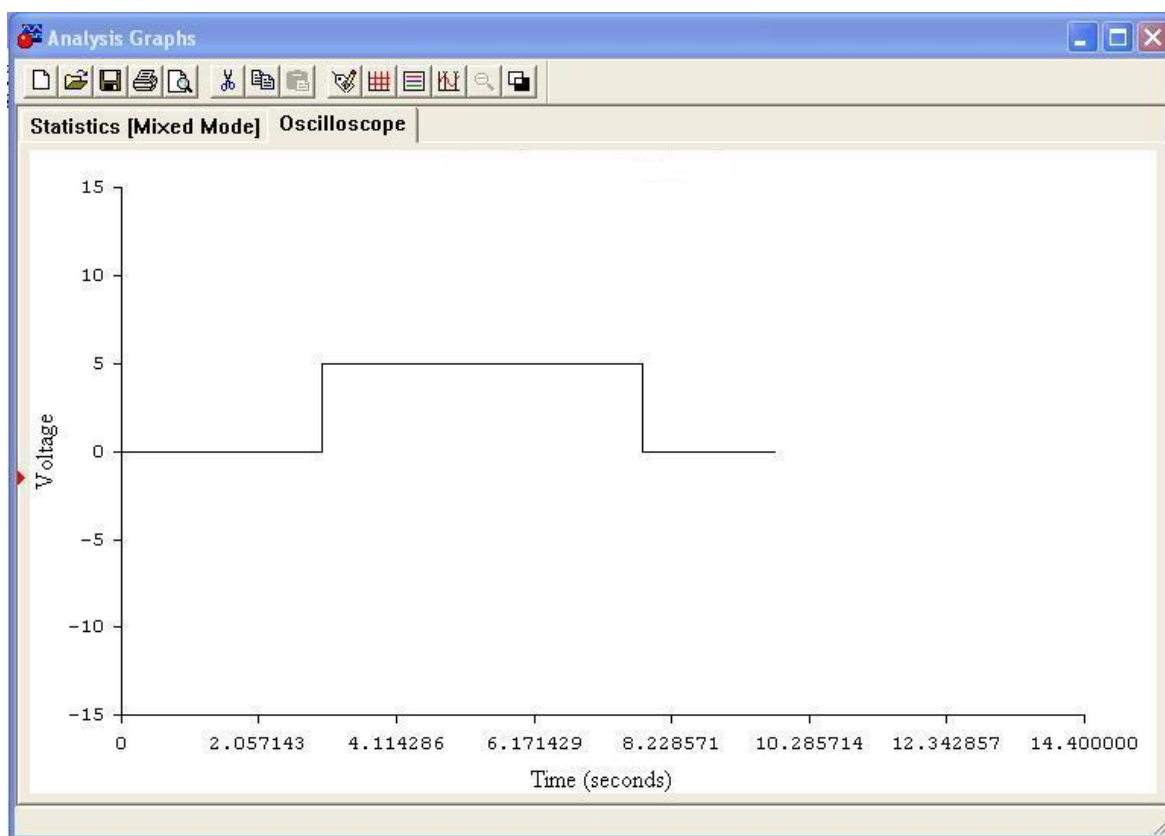
MC 7408 1x – čtyř vstupů AND

MC 7400 1x – čtyř vstupů NAND

MC 7432 1x – čtyř vstupů OR

MC 7402 1x – čtyř vstupů NOR

MC 7411 1x – tří vstupů AND



Obr. 14. Simulace 2 – Ověření funkčnosti zapojení pomocí IO

Ověření funkčnosti zapojení podle Obr. 13 a Tabulka II odpovídá požadovanému výstupu. Tzn. řešení nalezené pomocí algoritmu je správné.

Start of simulation No. 7 Time: 12 : 22 : 2.519061

Appended suboptimal solution with CV = 3 Time: 12 : 22 : 2.626946

Appended suboptimal solution with CV = 1 Time: 12 : 22 : 2.668224

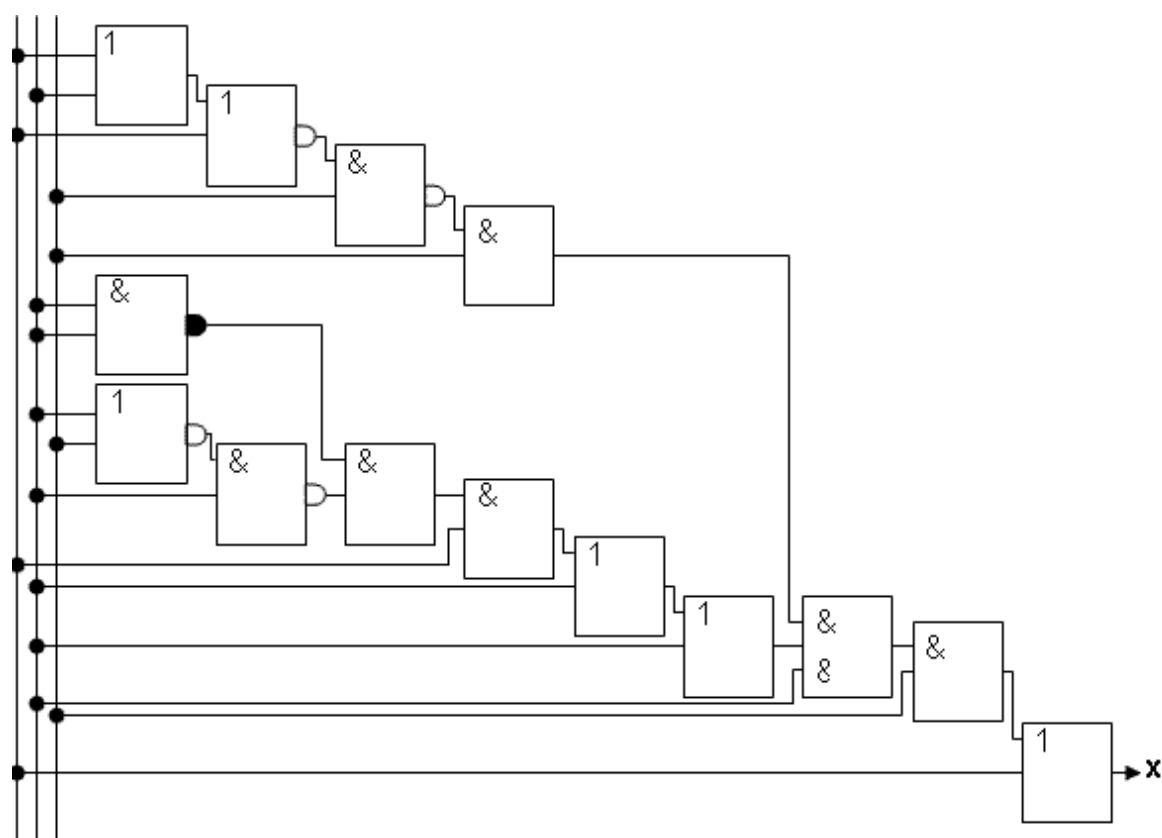
Number of cost function evaluations: 101

Cost value: 0

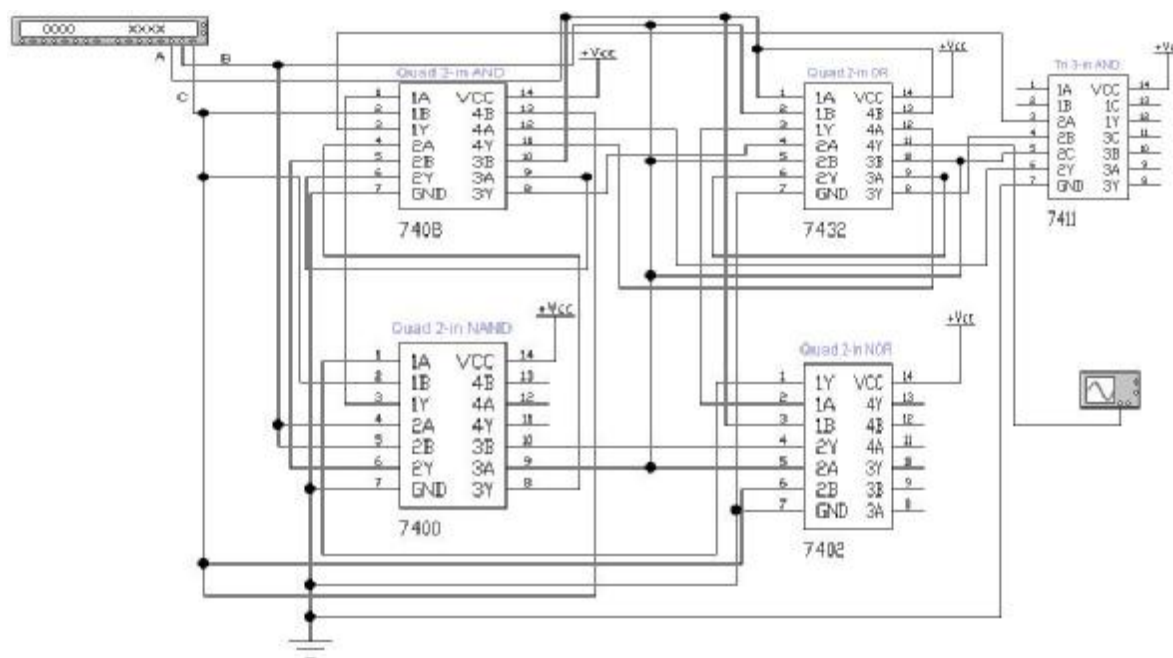
The best individual: {6, 9, 1, 5, 2, 11, 10, 2, 5, 10, 10, 9, 7, 5, 1, 7, 7, 11, 7, 10, 4, 9, 3, 4,  
11, 7, 10}

Solution:

$$A \vee (C \wedge B \wedge (B \vee B \vee (A \wedge (((B \wedge (B \vee C)) \wedge (B \wedge B)) \wedge (C \wedge (((A \vee B) \vee A) \wedge C))))))$$



Obr. 15. Simulace 7 – Blokové zapojení „Tepelná regulace“



Obr. 16. Simulace 7 - schéma zapojení pomocí integrovaných obvodů

Pro toto nalezené řešení jsem pro zapojení použil tyto integrované obvody:

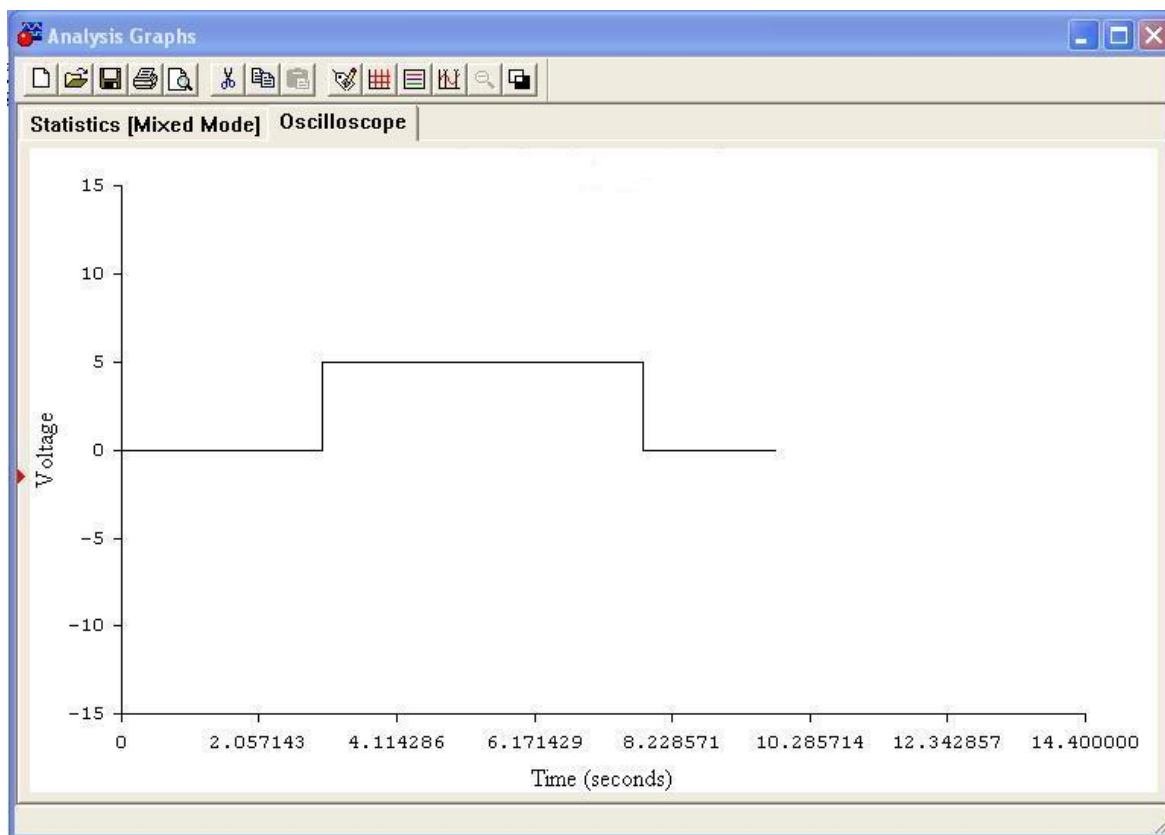
MC 7408 1x – čtyř vstupů AND

MC 7400 1x – čtyř vstupů NAND

MC 7432 1x – čtyř vstupů OR

MC 7402 1x – čtyř vstupů NOR

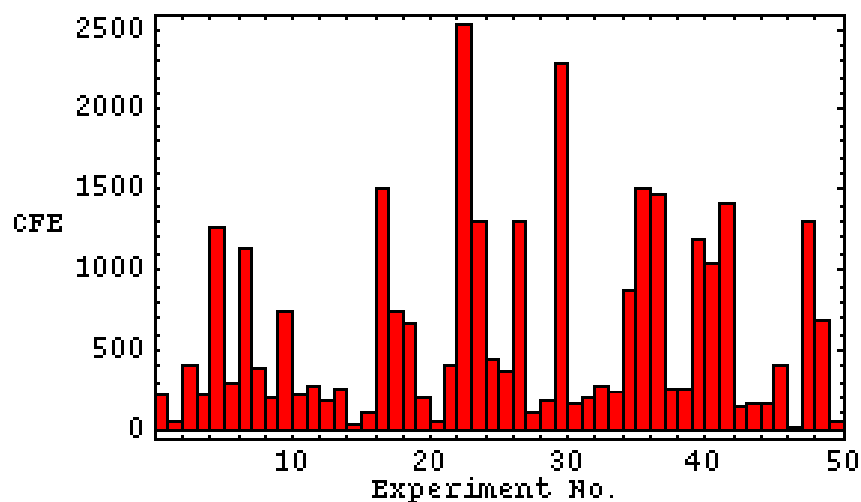
MC 7411 1x – tří vstupů AND



Obr. 17. Simulace 7 – Ověření funkčnosti zapojení pomocí IO

Ověření funkčnosti zapojení podle Obr. 16 a Tabulka II odpovídá požadovanému výstupu. Tzn. řešení nalezené pomocí algoritmu je správné.

### Grafické znázornění souhrnných výsledků 50-ti simulací

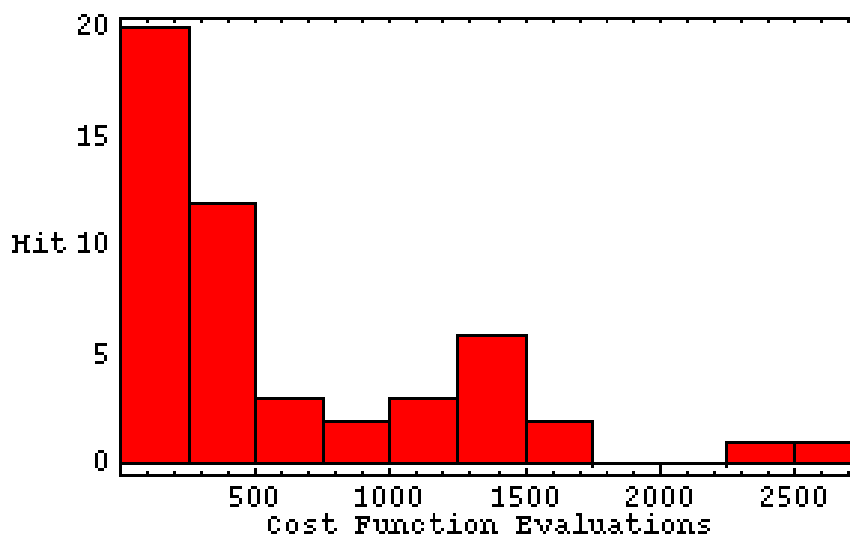


Obr. 18. Hodnoty CFE vůči opakovanému experimentu „Tepelná regulace“

Dle grafického znázornění jde zřetelně rozpoznat, které experimenty měly nejlepší jedince a dosáhli nejnižší a nejvyšší hodnoty CFE (účelové funkce).

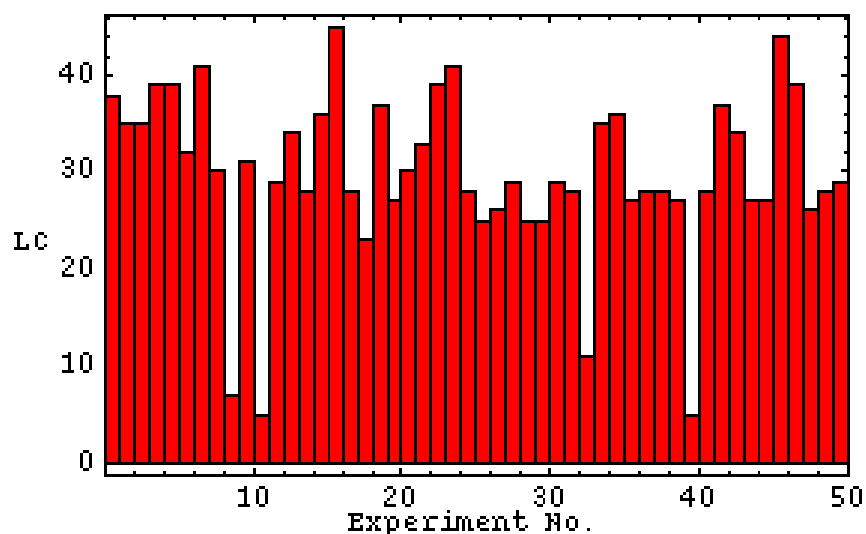
Hodnoty CFE:

{217, 50, 409, 219, 1258, 289, 1140, 397, 201, 741, 227, 278, 177, 257, 29, 115, 1509, 750, 664, 205, 45, 416, 2529, 1306, 447, 375, 1310, 101, 182, 2281, 160, 208, 273, 236, 874, 1515, 1466, 256, 256, 1191, 1048, 1415, 141, 167, 168, 415, 23, 1301, 683, 51}



Obr. 19. Histogram – Četnost výskytu CFE hodnot v intervalech – „Tepelná regulace“

Hit – tato hodnota udává četnost výskytu v jednotlivých intervalech. Jak je podle histogramu patrné pak 70% hodnot CFE se vyskytuje v intervalu 0 až 500.



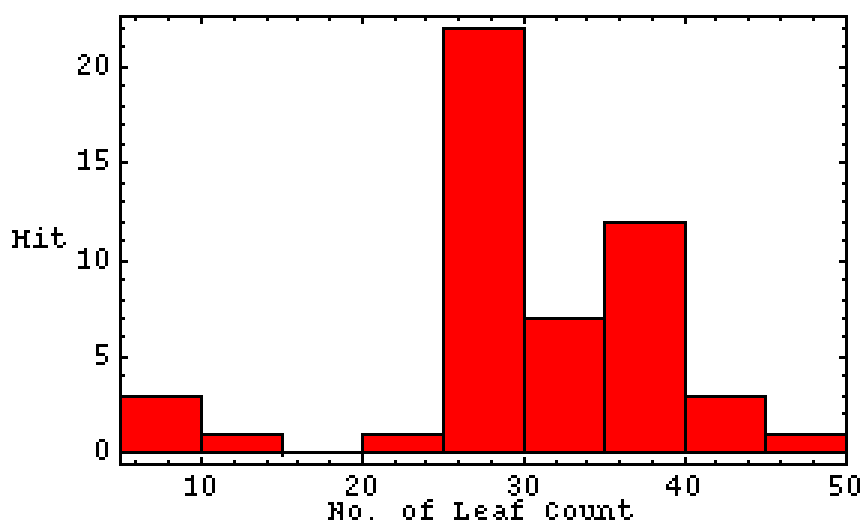
Obr. 20. Hodnoty Leaf count vůči opakovanému experimentu – „Tepelná regulace“

LC – Leaf count – počet operátorů, konstant, které byly použity pro zápis výsledného řešení

$$\text{Avg} = \text{CFE} / 50 = 599.42$$

$$\text{Mn} = \min [\text{CFE}] = 23$$

$$\text{Mx} = \max [\text{CFE}] = 2529$$



Obr. 21. Histogram – Četnost výskytu hodnot Leaf count v intervalech – „Tepelná regulace“

$$\text{Avg} = (\text{Plus}@@ \text{lfcnt3}) / 50 = 29.86$$

$$\text{mn} = \min [\text{lfcnt3}] = 5$$

$$\text{mx} = \max [\text{lfcnt3}] = 45$$

#### 4.4 Řešení úlohy „Řízení vlakových výhybek“ pomocí SOMA algoritmu

Tabulka III. Vstupní pravdivostní tabulka „Řízení vlakových výhybek“

| Vstup-výstup<br>1 | Vstup-výstup<br>2 | Vstup-výstup<br>3 | Vstup-výstup<br>4 | Output |
|-------------------|-------------------|-------------------|-------------------|--------|
| False             | False             | False             | False             | False  |
| False             | False             | False             | True              | False  |
| False             | False             | True              | False             | False  |
| False             | False             | True              | True              | False  |
| False             | True              | False             | False             | False  |
| False             | True              | False             | True              | False  |
| False             | True              | True              | False             | False  |
| False             | True              | True              | True              | False  |
| True              | False             | False             | False             | False  |
| True              | False             | False             | True              | True   |
| True              | False             | True              | False             | False  |
| True              | False             | True              | True              | True   |
| True              | True              | False             | False             | False  |
| True              | True              | False             | True              | True   |
| True              | True              | True              | False             | False  |
| True              | True              | True              | True              | False  |

Parametry algoritmu SOMA:

pocitadlo = 1 ;

PathLength = 3. ;

Step = 0.11 ;

PRT = .1 ;

PopSize = 300 ;

Specimen = Table[{{Integer, {1, AllDim}}}, {i, 30}] ;

Dim = Dimensions[Specimen][[1]] ;

Migrations = 30 ;

MinDiv = -.01 ;

**Vzor průběhu evoluce:**

Start of simulation No. 22 Time: 9 : 52 : 28.330371

Appended suboptimal solution with CV = 3 Time: 9 : 52 : 28.464847

Appended suboptimal solution with CV = 2 Time: 9 : 52 : 28.568059

Appended suboptimal solution with CV = 1 Time: 9 : 52 : 28.662280

Number of cost function evaluations: 12713

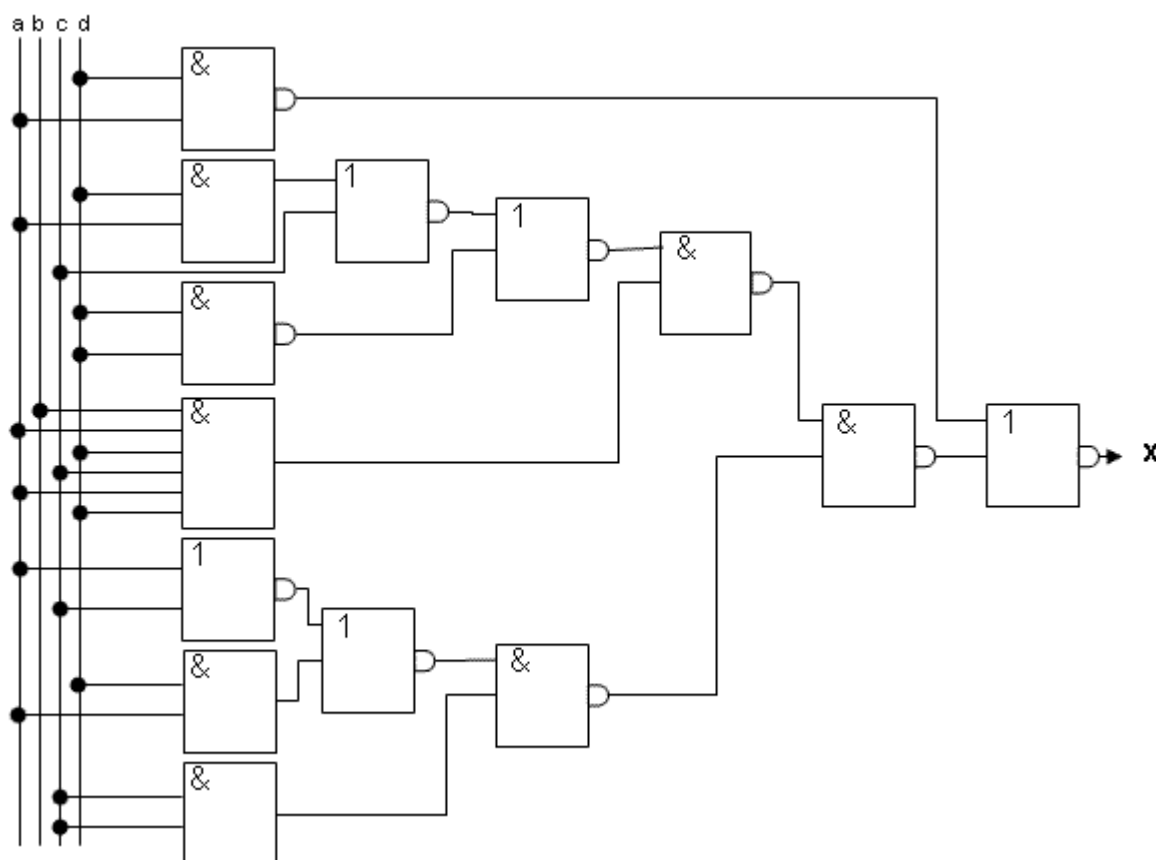
Cost value: 0

The best individual: {8, 3, 7, 12, 9, 7, 7, 8, 1, 8, 1, 8, 3, 1, 1, 10, 4, 1, 4, 5, 5, 3, 6, 4, 7, 8, 4,

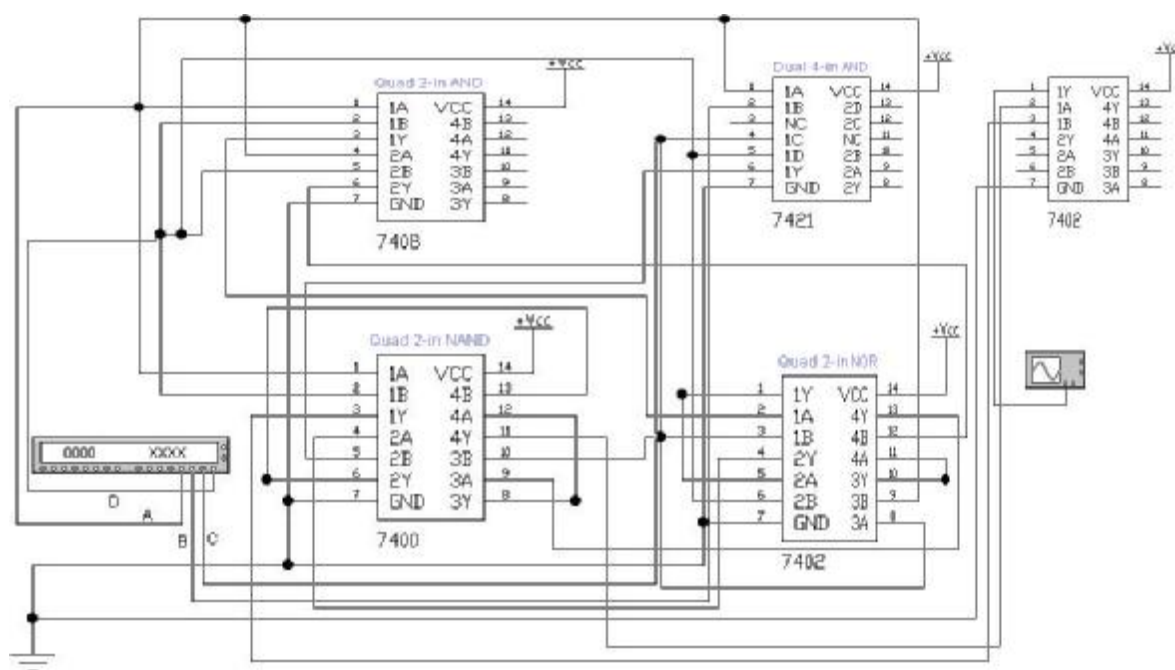
5}

Solution:

$$(D \wedge A) \vee (((((A \wedge D) \vee C) \vee (D \wedge D)) \wedge (B \wedge A \wedge D \wedge C \wedge A \wedge D)) \wedge (((A \vee C) \vee (A \wedge D)) \wedge (C \wedge C)))$$



Obr. 22. Simulace 22 – Blokové zapojení „Řízení vlakových výhybek“



Obr. 23. Simulace 22 – Schéma zapojení pomocí integrovaných obvodů

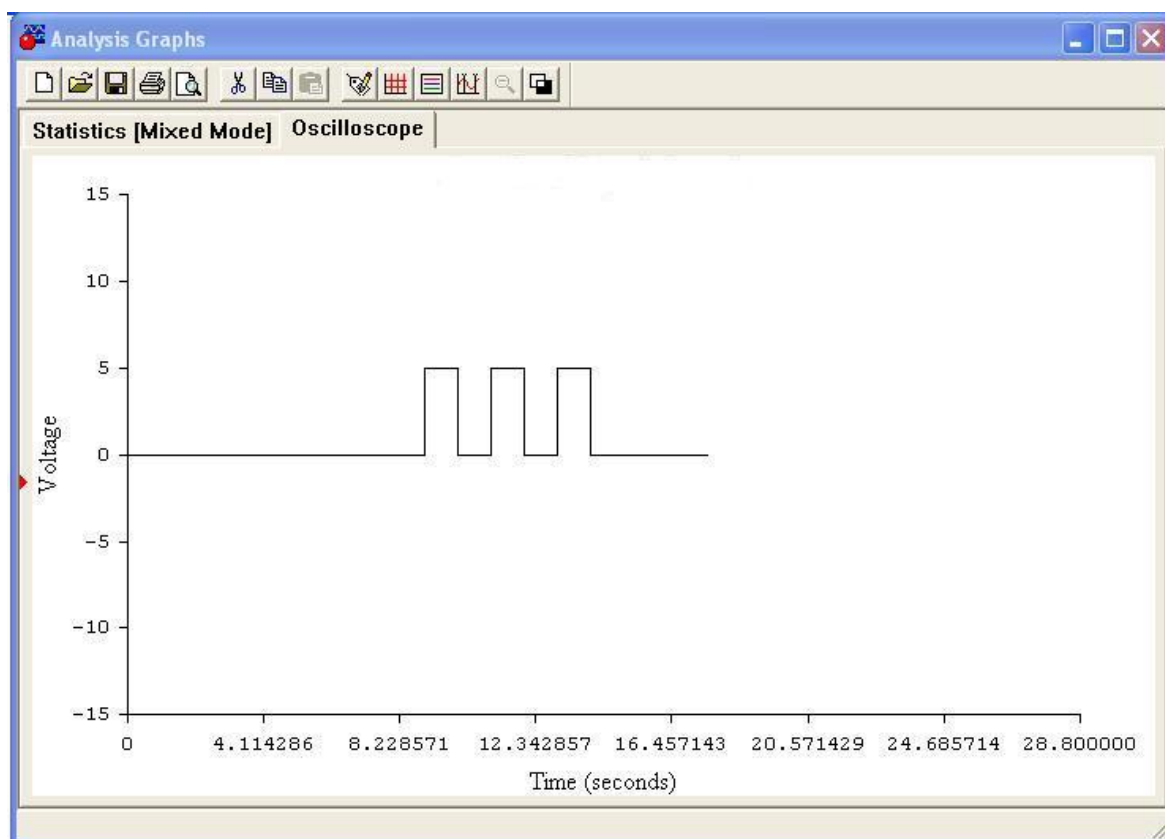
Pro toto nalezené řešení jsem pro zapojení použil tyto integrované obvody:

MC 7408 1x – čtyř vstupů AND

MC 7400 1x – čtyř vstupů NAND

MC 7421 1x – osmi vstupů AND

MC 7402 2x – čtyř vstupů NOR



Obr. 24. Simulace 22 – Ověření funkčnosti zapojení pomocí IO

Ověření funkčnosti zapojení podle Obr. 23 a Tabulka III odpovídá požadovanému výstupu. Tzn. řešení nalezené pomocí algoritmu je správné.

Start of simulation No. 37 Time: 10 : 4 : 45.633434

Appended suboptimal solution with CV = 4 Time: 10 : 4 : 45.737827

Appended suboptimal solution with CV = 3 Time: 10 : 4 : 45.767699

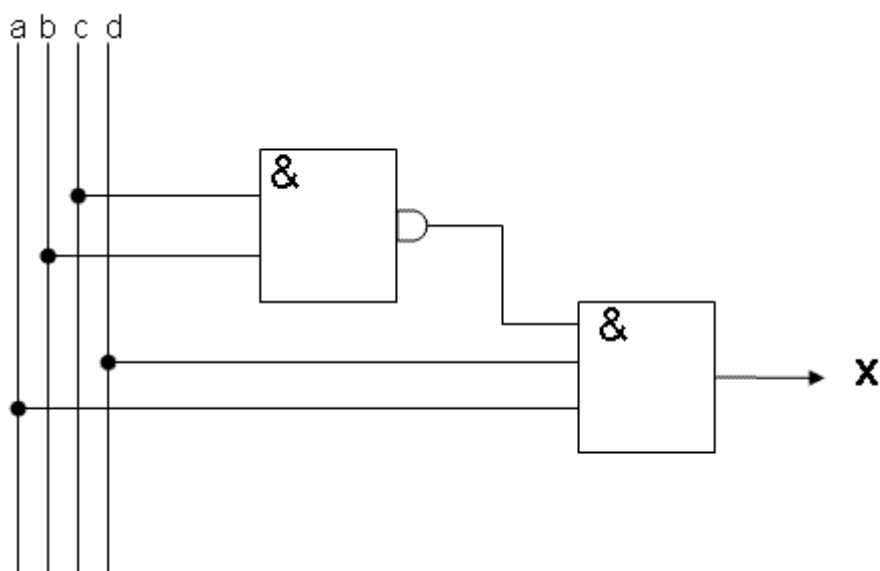
Appended suboptimal solution with CV = 2 Time: 10 : 5 : 23.054468

Number of cost function evaluations: 1999

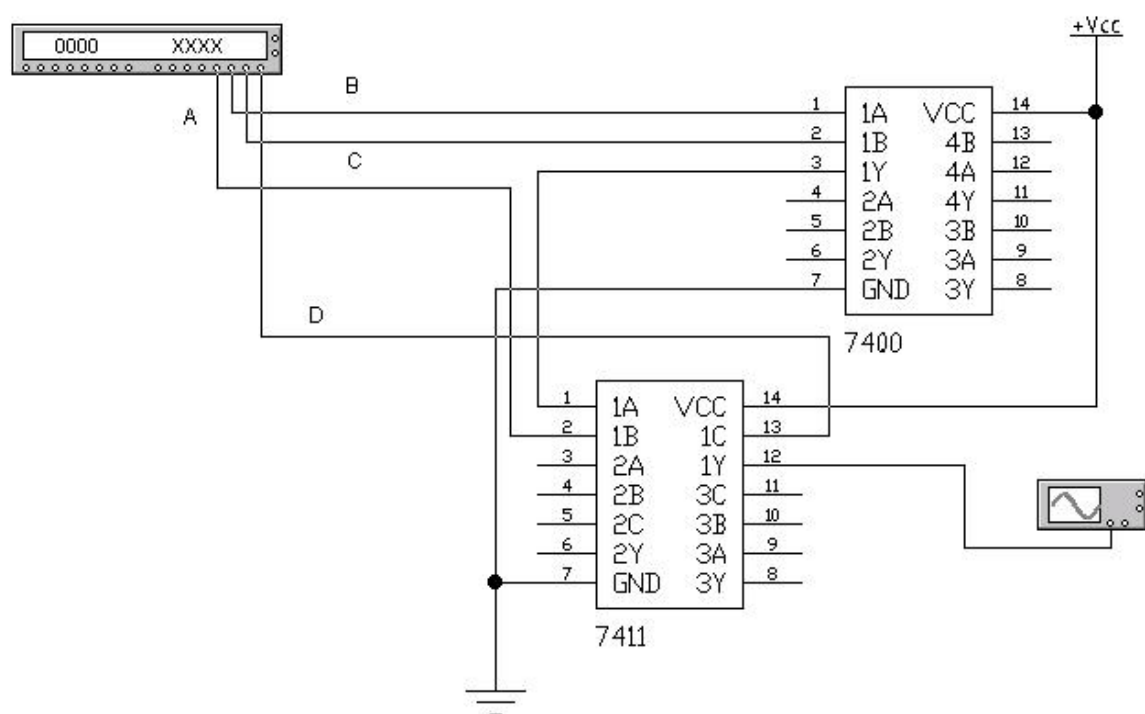
Cost value: 0

The best individual: {1, 12, 1, 3, 9, 11, 10, 1, 1, 9, 10, 10, 6, 9, 6, 10, 8, 9, 6, 7, 5, 6, 3, 10,  
6, 2, 2, 8, 7, 3}

Solution:  $D \wedge (C \wedge B) \wedge A$



Obr. 25. Simulace 37 – Blokové zapojení „Řízení vlakových výhybek“

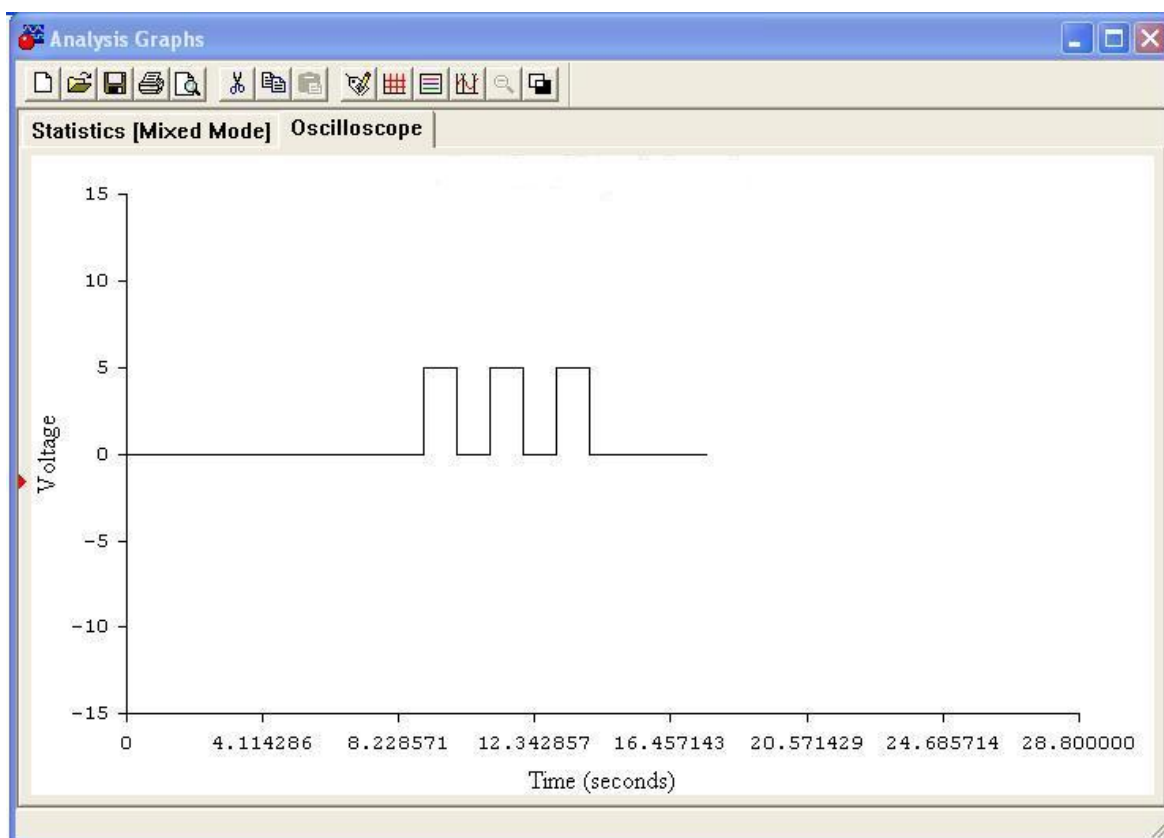


Obr. 26. Simulace 37 – Schématické zapojení pomocí integrovaných obvodů

Pro toto nalezené řešení jsem pro zapojení použil tyto integrované obvody:

MC 7400 1x – čtyř vstupý NAND

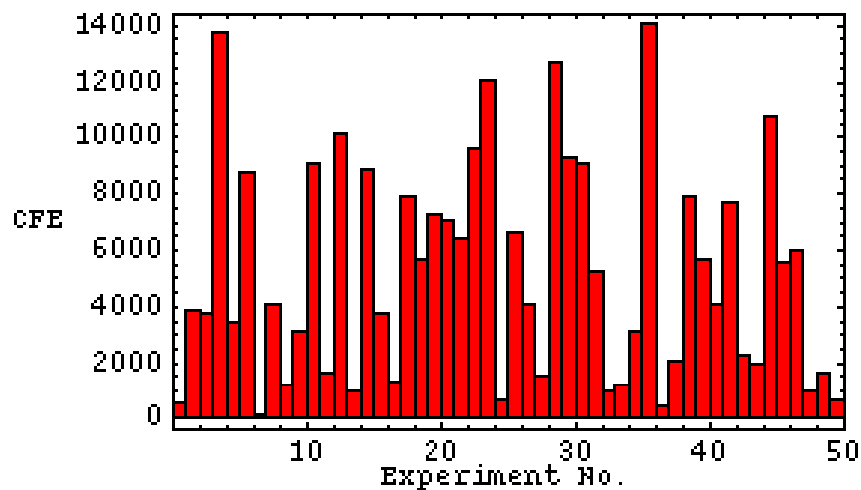
MC 7411 1x – tři vstupý AND



Obr. 27. Simulace 37 – Ověření funkčnosti zapojení pomocí IO

Ověření funkčnosti zapojení podle Obr. 26 a Tabulka III odpovídá požadovanému výstupu. Tzn. řešení nalezené pomocí algoritmu je správné.

### Grafické znázornění souhrnných výsledků 50-ti simulací

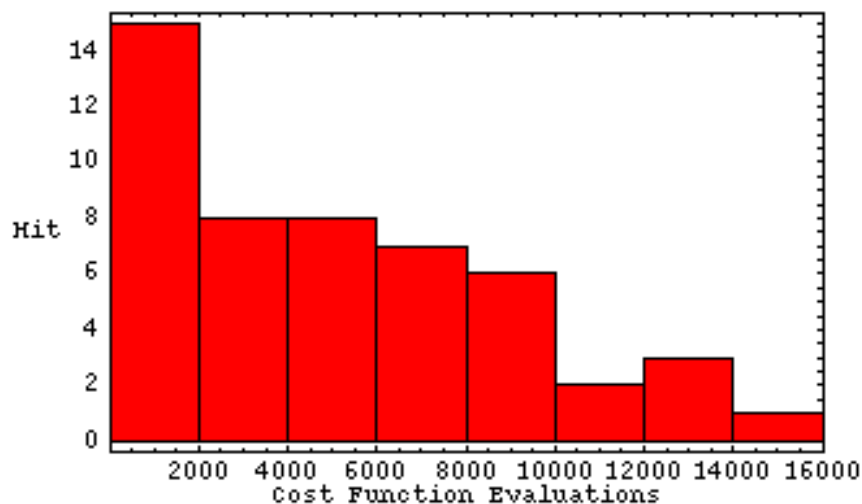


Obr. 28. Hodnoty CFE vůči opakovanému experimentu „Řízení vlakových výhybek“

Dle grafického znázornění jde zřetelně rozpoznat, které experimenty měly nejlepší jedince a dosáhli nejnižší a nejvyšší hodnoty CFE.

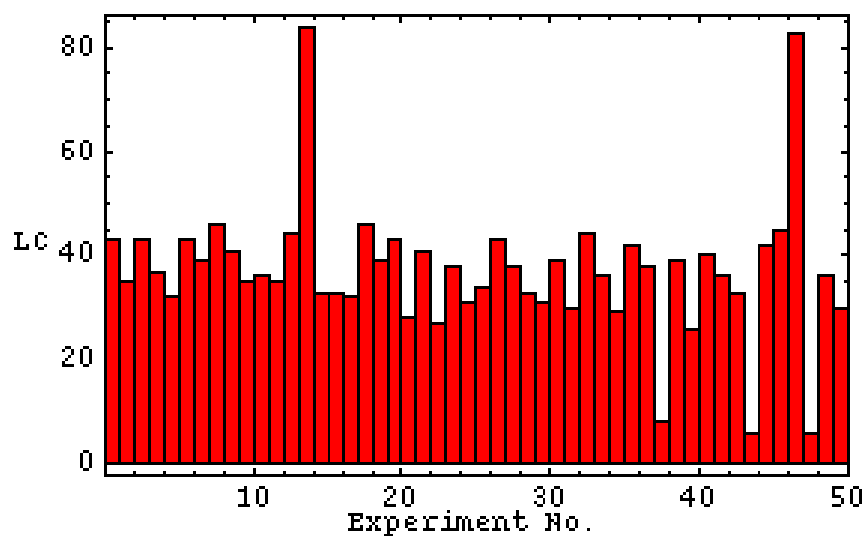
Hodnoty CFE (Cost function evaluation):

{570, 3926, 3786, 13744, 3425, 8800, 147, 4054, 1182, 3180, 9108, 1680, 10166, 988, 8890, 3742, 1270, 7883, 5683, 7287, 7092, 6456, 9628, 12088, 706, 6664, 4066, 1511, 12713, 9349, 9048, 5205, 991, 1196, 3116, 14069, 494, 2034, 7863, 5713, 4096, 7702, 2242, 1999, 10816, 5575, 5981, 1022, 1680, 729}



Obr. 29. Histogram – Četnost výskytu CFE hodnot v intervalech „Řízení vlakových výhybek“

Hit – tato hodnota udává četnost výskytu v jednotlivých intervalech. Jak je podle histogramu patrné pak 80% hodnot CFE se vyskytuje v intervalu 50 000 až 100 000.

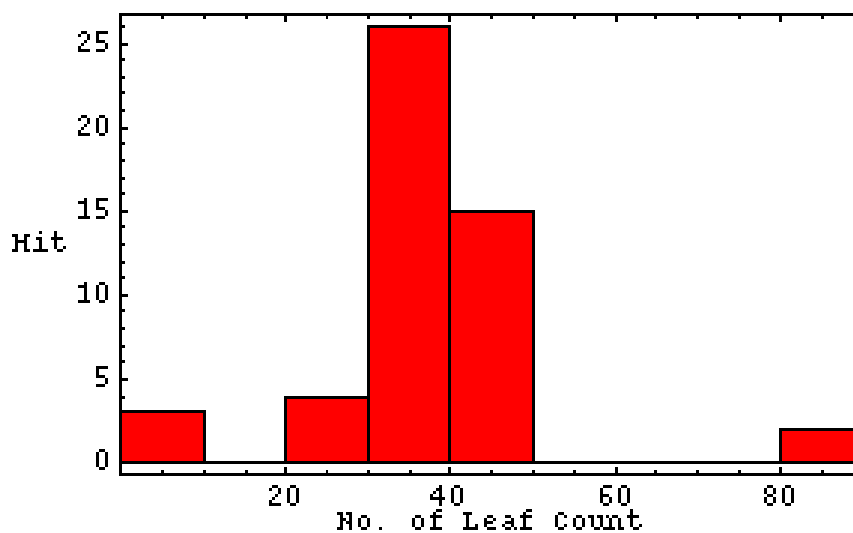


Obr. 30. Hodnoty Leaf count vůči opakovanému experimentu „Řízení vlakových výhybek“

$$\text{Avg} = (\text{Plus} @@ \text{CFE} / 50 = 5227.1$$

$$\text{Mn} = \min [\text{CFE}] = 147$$

$$\text{Mx} = \max [\text{CFE}] = 14069$$



Obr. 31. Histogram – Četnost výskytu Leaf count hodnot v „Řízení vlakových výhybek“

$$\text{Avg} = (\text{Plus} @@ \text{lfcnt3}) / 50 = 37.02$$

$$\text{mn} = \min [\text{lfcnt3}] = 6$$

$$\text{mx} = \max [\text{lfcnt3}] = 84$$

## 4.5 Aplikace penalizačního faktoru na Řízení vlakových výhybek

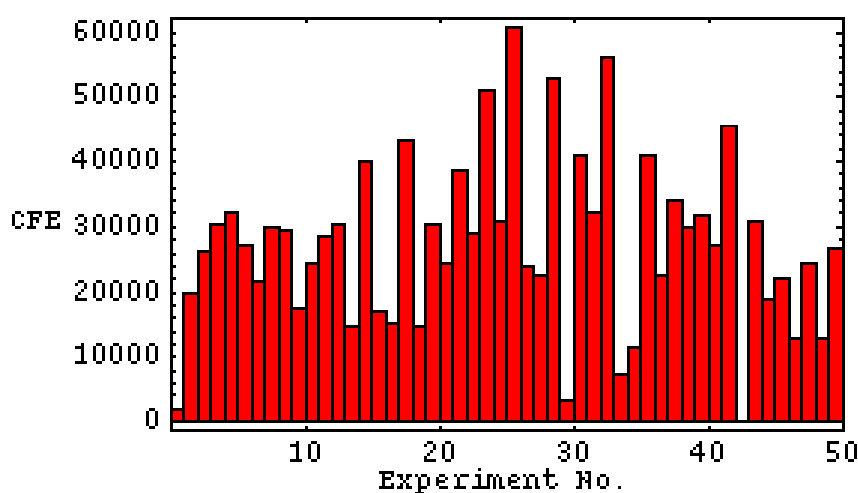
### 4.5.1 Definování účelové funkce

$$F_{COST} = \sum_{i=1}^n ABS(PRG_{out}; -OUT;) + 10 * LC(PRG)$$

Účelová funkce byla definována jako suma jednotlivých absolutních výstupních programových hodnot mínus celkový výstup násobeno celkovou délkou programu.

Vzhledem k tomu, že evoluční algoritmus vygeneroval během cyklů mnoho redundantních závislostí, rozhodl jsem se aplikovat tzv. penalizační faktor tak, abych dosáhl optimálních výsledků, tzn. aby hodnoty Leaf Count byly co nejnižší. Pro názornost jsem toto aplikoval na řešení „Řízení vlakových výhybek“, kde jsou 4 vstupní faktory a výsledná funkce se stávala často nepřehlednou. Po aplikaci penalizačního faktoru na řešení tohoto problému jsme dostali celkové výsledky tyto:

**Grafické znázornění souhrnných výsledků 50-ti simulací s aplikací penalizačního faktoru**

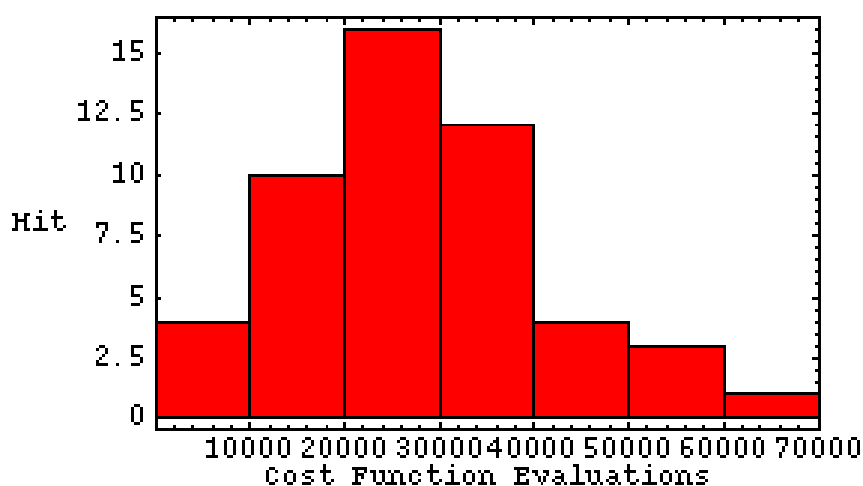


Obr. 32. Hodnoty CFE vůči opakovanému experimentu „Řízení vlakových výhybek“ – s aplikací penalizačního faktoru

Dle grafického znázornění jde zřetelně rozpoznat, které experimenty měly nejlepší jedince a dosáhli nejnižší a nejvyšší hodnoty CFE (účelové funkce).

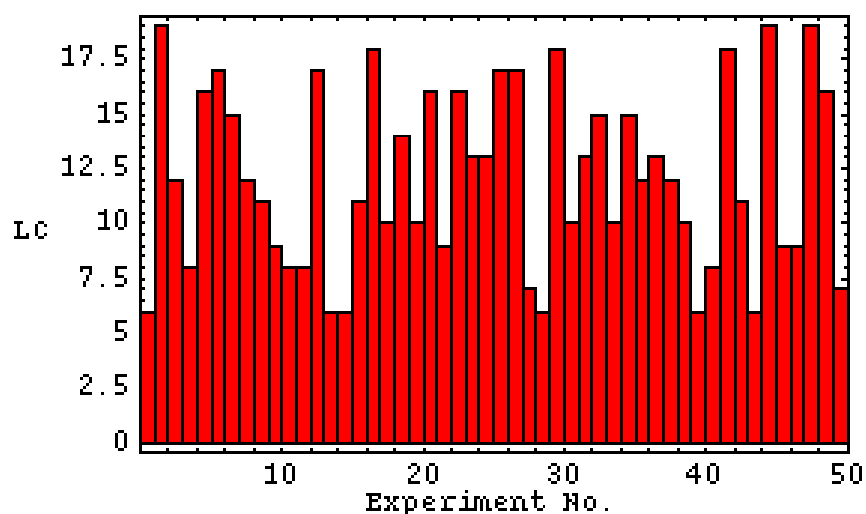
Hodnoty CFE:

```
{2005, 19984, 26440, 30389, 32269, 27102, 21729, 30010, 29653,
17479, 24296, 28642, 30268, 14888, 39992, 16926, 15177,
43166, 14646, 30306, 24327, 38840, 28851, 51173, 30772,
60762, 23907, 22605, 52991, 3332, 40741, 32038, 56129,
7362, 11684, 40895, 22705, 33869, 29979, 31869, 27341,
45635, 10, 30810, 18917, 22176, 13143, 24578, 12960, 26780}
```



Obr. 33. Histogram – Četnost výskytu CFE hodnot v intervalech „Řízení vlakových výhybek“ – s aplikací penalizačního faktoru

Hit – tato hodnota udává četnost výskytu v jednotlivých intervalech. Jak je podle histogramu patrné pak 80% hodnot CFE se vyskytuje v intervalu 10 000 až 40 000.

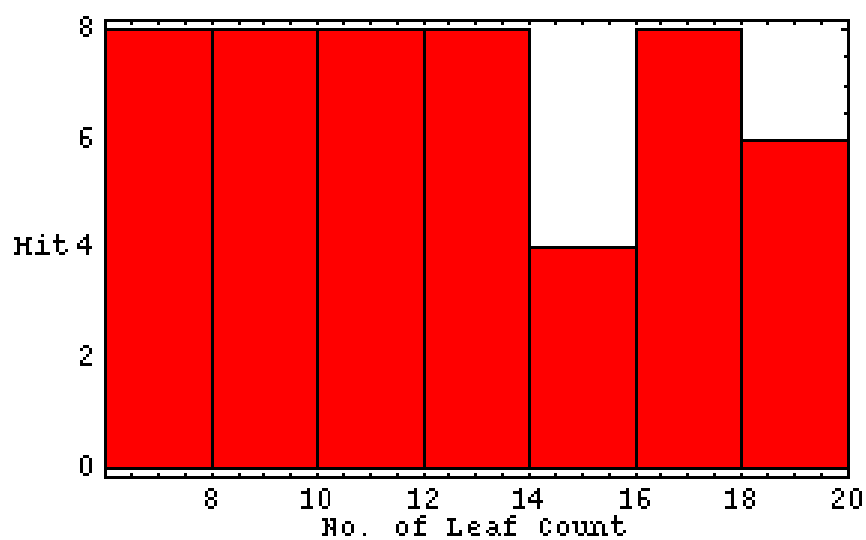


Obr. 34. Hodnoty Leaf count vůči opakovanému experimentu „Řízení vlakových výhybek“ – s aplikací penalizačního faktoru

$$\text{Avg} = (\text{Plus} @@ \text{CFE} / 50 = 27251$$

$$\text{Mn} = \min [\text{CFE}] = 10$$

$$\text{Mx} = \max [\text{CFE}] = 60\,762$$



Obr. 35. Histogram – Četnost výskytu Leaf count hodnot v „Řízení vlakových výhybek“ – s aplikací penalizačního faktoru

$$\text{Avg} = (\text{Plus} @@ \text{lfcnt3}) / 50 = 12.06$$

$$\text{mn} = \min [\text{lfcnt3}] = 6$$

$$\text{mx} = \max [\text{lfcnt3}] = 19$$

## ZÁVĚR

Tato bakalářská práce je zaměřena na ověření, zda mohou být Evoluční algoritmy vhodný nástroj pro design elektronických obvodů.

Všechny výše uvedené úlohy našli vždy během všech 50-ti opakovaných simulací řešení. Po zpracování a analýze vybraných výsledků jsem došel k názoru, že je třeba zanést do účelové funkce i tzv. penalizační faktor, který optimalizuje výsledné řešení tak, že se ve výsledných hodnotách nebudou vyskytovat redundantní hodnoty typu jako C OR C apod. Po aplikaci penalizačního faktoru na problém „Řízení vlakových výhybek“ se výsledná hodnota Leaf Count snížila více jak o 50%. Tzn. že celková průměrná délka výstupní funkce je na hodnotě 12,06 místo 37,02. Což se pro vyhodnocení a aplikaci řešení stává naprosto přehledným a dostačujícím. Bohužel naproti tomu se nám zvýšila hodnota CFE (účelové funkce) z průměrných 5227 na 27 251. Naštěstí pro nás s tímto výsledkem dále nepracujeme a proto pro nás není tolik kritický.

Z výsledků, analýz a konsultací jsem došel k názoru, že tato metoda je mnohem progresivnější, než používají některé renomované firmy, které se touto problematikou zabývají. Například Designové středisko firmy ONSEMICONDUCTOR používá pro své návrhy el. obvodů tzv. heuristickou analýzu, tato metoda je numerická a může při ní docházet k chybám.

Při aplikaci Evolučních algoritmů je to téměř nemožné.

**SEZNAM POUŽITÉ LITERATURY**

- ( 1 ) Koza J. R. Genetic Programing, MIT Press, ISBN 0-262-11189-6, 1998
- ( 2 ) Ryan C., Collins J.J., O'Neill M. Grammatical Evolution: Evolving Programs for an Arbitrary Language. Lecture Notes in Computer Science 1391. First European Workshop on Genetic Programming 1998
- ( 3 ) Koza J.R., Bennert F. H., Andre D., Keane M., Genetic Programming III, Morgan Kaufmann pub., ISBN 1-55860-543-6, 1999
- ( 4 ) Zelinka I.:Analytic programing by Means of Soma Algorithm. Mendel '02, In: Proc. 8<sup>th</sup> International Conference on Soft Computing Mendel '02 Brno, Czech Republic, 2002, 93-101., ISBN 80-214-2135-5
- ( 5 ) Zelinka I., Oplatkova Z.: Analytic programming – Comparative Study. CIRAS'03, The sekond International Konference on Computational Inteligence, Robotíce, and Autonomous Systems, Singapure, 2003, ISSN 0219-6131

## SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

AP – Analytické programování

GP – Genetické programování

CFE – hodnota účelové funkce

LC – souhrnný název pro konstanty, operátory

SR – symbolická regrese

MIM - Manipulace s izolovanou množinou

HFP – hlavní funkční prostor

## SEZNAM OBRÁZKŮ

|                                                                                                                                     |    |
|-------------------------------------------------------------------------------------------------------------------------------------|----|
| Obr. 1. Algoritmus genetického programování.....                                                                                    | 9  |
| Obr. 2. Simulace 1 – Blokové zapojení „Světelná signalizace“.....                                                                   | 21 |
| Obr. 3. Simulace 1 – Schématické zapojení pomocí integrovaných obvodů .....                                                         | 22 |
| Obr. 4. Simulace 1 – Ověření funkčnosti zapojení integrovaných obvodů .....                                                         | 23 |
| Obr. 5. Simulace 2 – Blokové zapojení „Světelná signalizace“.....                                                                   | 24 |
| Obr. 6. Simulace 2 – Schématické zapojení pomocí integrovaných.....                                                                 | 25 |
| Obr. 7. Simulace 2 – Ověření funkčnosti zapojení pomocí IO.....                                                                     | 26 |
| Obr. 8. Přehled hodnot CFE pro jednotlivé experimenty – „Světelná signalizace“ .....                                                | 27 |
| Obr. 9. Histogram – Četnost výskytu hodnot CFE v intervalech – „Světelná signalizace“ ..                                            | 27 |
| Obr. 10. Hodnoty Leaf count vůči opakovanému experimentu – „Světelná signalizace“.....                                              | 28 |
| Obr. 11. Histogram – Četnost výskytu hodnot Leaf count v intervalech – „Světelná<br>signalizace“ .....                              | 28 |
| Obr. 12. Simulace 2 – Blokové zapojení „Tepelná regulace“ .....                                                                     | 30 |
| Obr. 13. Simulace 2 – Schématické zapojení pomocí integrovaných obvodů.....                                                         | 31 |
| Obr. 14. Simulace 2 – Ověření funkčnosti zapojení pomocí IO.....                                                                    | 32 |
| Obr. 15. Simulace 7 – Blokové zapojení „Tepelná regulace“ .....                                                                     | 33 |
| Obr. 16. Simulace 7 - schéma zapojení pomocí integrovaných obvodů .....                                                             | 34 |
| Obr. 17. Simulace 7 – Ověření funkčnosti zapojení pomocí IO.....                                                                    | 35 |
| Obr. 18. Hodnoty CFE vůči opakovanému experimentu „Tepelná regulace“ .....                                                          | 36 |
| Obr. 19. Histogram – Četnost výskytu CFE hodnot v intervalech – „Tepelná regulace“.....                                             | 36 |
| Obr. 20. Hodnoty Leaf count vůči opakovanému experimentu – „Tepelná regulace“ .....                                                 | 37 |
| Obr. 21. Histogram – Četnost výskytu hodnot Leaf count v intervalech – „Tepelná<br>regulace“ .....                                  | 37 |
| Obr. 22. Simulace 22 – Blokové zapojení „Řízení vlakových výhybek“ .....                                                            | 39 |
| Obr. 23. Simulace 22 – Schéma zapojení pomocí integrovaných obvodů .....                                                            | 40 |
| Obr. 24. Simulace 22 – Ověření funkčnosti zapojení pomocí IO.....                                                                   | 41 |
| Obr. 25. Simulace 37 – Blokové zapojení „Řízení vlakových výhybek“ .....                                                            | 42 |
| Obr. 26. Simulace 37 – Schématické zapojení pomocí integrovaných obvodů.....                                                        | 42 |
| Obr. 27. Simulace 37 – Ověření funkčnosti zapojení pomocí IO.....                                                                   | 43 |
| Obr. 28. Hodnoty CFE vůči opakovanému experimentu „Řízení vlakových výhybek“.....                                                   | 44 |
| Obr. 29. Histogram – Četnost výskytu CFE hodnot v intervalech „Řízení vlakových<br>výhybek“ .....                                   | 44 |
| Obr. 30. Hodnoty Leaf count vůči opakovanému experimentu „Řízení vlakových výhybek“.....                                            | 45 |
| Obr. 31. Histogram – Četnost výskytu Leaf count hodnot v „Řízení vlakových výhybek“ ..                                              | 45 |
| Obr. 32. Hodnoty CFE vůči opakovanému experimentu „Řízení vlakových výhybek“ –<br>s aplikací penalizačního faktoru.....             | 46 |
| Obr. 33. Histogram – Četnost výskytu CFE hodnot v intervalech „Řízení vlakových<br>výhybek“ – s aplikací penalizačního faktoru..... | 47 |
| Obr. 34. Hodnoty Leaf count vůči opakovanému experimentu „Řízení vlakových výhybek“<br>– s aplikací penalizačního faktoru.....      | 48 |
| Obr. 35. Histogram – Četnost výskytu Leaf count hodnot v „Řízení vlakových výhybek“ –<br>s aplikací penalizačního faktoru.....      | 48 |

**SEZNAM TABULEK**

|                                                                            |    |
|----------------------------------------------------------------------------|----|
| Tabulka I. Vstupní pravdivostní tabulka „Světelná signalizace“ .....       | 20 |
| Tabulka II. Vstupní pravdivostní tabulka „Tepelná regulace“ .....          | 29 |
| Tabulka III. Vstupní pravdivostní tabulka „Řízení vlakových výhybek“ ..... | 38 |