VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

EVOLUČNÍ NÁVRH STRUKTUR VYUŽÍVAJÍCÍ PŘEPISOVACÍ SYSTÉMY

EVOLUTIONARY DESIGN OF STRUCTURES USING REWRITING SYSTEMS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. MICHAL DOBEŠ**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MICHAL BIDLO, Ph.D.**
SUPERVISOR

BRNO 2017

Zadání diplomové práce/19540/2016/xdobes13

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2016/2017

Zadání diplomové práce

Řešitel: **Dobeš Michal, Bc.**
Obor: Inteligentní systémy
Téma: **Evoluční návrh struktur využívající přepisovací systémy**
Evolutionary Design of Structures Using Rewriting Systems
Kategorie: Umělá inteligence

Pokyny:

1. Seznamte se s problematikou evolučních algoritmů (EA).
2. Seznamte se s principy biologií inspirovaného vývinu, aplikovaného v evolučních algoritmech, kdy je tento proces realizován pomocí přepisovacích systémů.
3. Zvolte vhodný typ přepisovacího systému a navrhněte jeho zakódování pro EA tak, aby bylo možné interpretovat jednotlivé jeho symboly jako stavební bloky určité struktury (např. struktury číslicového obvodu), případně jako instrukce pro jejich generování.
4. Navržený systém implementujte.
5. Zvolte vhodnou případovou studii (např. z oblasti algoritmů, návrhu obvodových struktur apod.) a proveďte sadu experimentů, na kterých ukážete možnosti jejího automatického (evolučního) řešení pomocí techniky navržené v bodu 3.
6. Zhodnoťte dosažené výsledky a diskutujte možnosti pokračování projektu.

Literatura:

- Podle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 3 zadání, demonstrace základního prototypu evolučního systému z bodu 4.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Bidlo Michal, Ing., Ph.D.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 24. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

prof. Ing. Lukáš Sekanina, Ph.D.
vedoucí ústavu

Abstrakt

Tato práce se zabývá aplikací přepisovacích systémů pro biologií inspirovaný vývin struktur v oblasti evolučních algoritmů. V rámci práce byla vytvořena metoda pro návrh obecných struktur založená na uvedeném principu. Práce popisuje teorii řadicích sítí a jejich návrhu, jakožto problému, který lze řešit navrženou metodou. Dále je uvedena teorie z oblasti evolučních algoritmů a přepisovacích systémů. V oblasti přepisovacích systémů bylo pro použití v navržené metodě vytvořeno rozšíření třídy IL-systémů, MDIL-systémy. Navržená metoda byla úspěšně uplatněna při návrhu rostoucích řadicích sítí. Dosažené výsledky v oblasti konstrukce řadicích sítí rostoucích po třech a čtyřech vstupech výrazně překonávají aktuálně nejlepší srovnatelnou metodu.

Abstract

This work focuses on application of rewriting systems in the context of biology-inspired development of structures in evolutionary algorithms. As a part of this work, a method has been proposed for design of general structures that uses the aforementioned principle. This document includes an introduction into the theory of sorting networks and their design, the problem of which is later shown to be solvable using the proposed method. In addition, the theoretical background of evolutionary algorithms and rewriting systems is discussed. In the field of rewriting systems, an extension to the class of IL-systems has been created for use in the proposed method. The proposed method has been successfuly applied in the design of growing sorting networks. The results produced by the proposed method significantly outperform the results of the currently best-known comparable method.

Klíčová slova

evoluční algoritmus, řadicí síť, biologií inspirovaný vývin, přepisovací systém, L-systém

Keywords

evolutionary algorithm, sorting network, biology inspired development, development, rewriting system, L-system

Citace

DOBEŠ, Michal. *Evoluční návrh struktur využívající přepisovací systémy.* Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Bidlo Michal.

Evoluční návrh struktur využívající přepisovací systémy

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Bidla, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Michal Dobeš
17. května 2017

Poděkování

Děkuji vedoucímu své práce, Ing. Michalu Bidlovi, PhD., za ochotu při konzultacích, za odborné vedení a za umožnění výpočtů a poskytnutí výpočetního času na superpočítači Anselm.

Tato práce byla podpořena Ministerstvem školství, mládeže a tělovýchovy České republiky v rámci projektu velkých infrastruktur pro výzkum, vývoj a inovace „IT4Innovations národní superpočítačové centrum – LM2015070“.

Obsah

Kapitola 1

Úvod

Již dlouhou dobu používáme pro řešení některých inženýrských problémů genetické algoritmy. Uplatnění našly i při tvorbě elektronických analogových i číslicových obvodů, kde často dosahují lepších výsledků než tradiční inženýrské přístupy. V této oblasti se pojmem staly metody založené na *genetickém programování*, především Kozova metoda [12] a metoda kartézského genetického programování (CGP) [15]. Velmi vhodné jsou pak pro konstrukci řadicích sítí, kde jejich výsledky mají potenciál překonat tradiční přístupy [10] [23]. Pro hledání metod konstrukce řadicích sítí úspěšně použil evoluci například M. Bidlo [4].

Některé z těchto metod (např. CGP) používají tzv. *přímou reprezentaci*, kde genotypy vyvíjených jedinců[1] lze přímo transformovat (mapovat) na kandidátní řešení daného problému, tzv. *fenotypy*. Genotypy tak například mohou popisovat propojení hradel v číslicovém obvodu. Se vzrůstající velikostí vyvíjených struktur však tato skupina metod naráží na tzv. *problém škálovatelnosti reprezentace*. Jinými slovy, přímé kódování, které bylo vhodné pro menší instance problému, přestává být vhodným pro kódování větších instancí. Problém škálovatelnosti reprezentace je obzvláště patrný u metody CGP, kde pro veliké obvody metoda přestává být v rozumném čase schopna nalézt řešení.

Jedno z řešení problému škálovatelnosti reprezentace představuje skupina metod, které používají tzv. *nepřímou reprezentaci*. V nepřímé reprezentaci je genotyp jedince pouhým předpisem pro vytvoření fenotypu (struktury). Při volbě vhodného kódování tak lze relativně krátkým genotypem reprezentovat složitou strukturu, což značně usnadňuje proces evoluce. Do této skupiny patří např. Kozova metoda [12], či metoda navržená M. Bidlem [4].

Obě zmíněné metody přitom sdílejí princip tzv. *výpočetního developmentu*. Jedná se o princip růstu organismu z jeho zárodku, tzv. *embrya*, na větší organismus. V Kozově metodě je genotyp tvořen stromem operací, které jsou postupně prováděny nad stanoveným zárodkem obvodu (embryem). V metodě M. Bidla je genotyp dán sekvencí instrukcí, které jsou aplikovány na stanovenou zárodečnou řadicí síť. V obou případech tak z embrya vzrůstá jedincův fenotyp, přičemž je tento růst řízen genotypem.

Při genetickém návrhu řadicích sítí, jakožto i obecných číslicových obvodů, narážíme také na tzv. *problém škálovatelnosti evaluace*. Tento problém spočívá v nárůstu počtu operací potřebných pro ověření funkčnosti kandidátního řešení. Jak v případě řadicích sítí, tak v případě číslicových obvodů obecně, roste počet potřebných operací exponenciálně v závislosti na počtu vstupů obvodu. To znesnadňuje evoluční návrh větších instancí těchto problémů.

[1] Jedincem v terminologii evolučních algoritmů rozumějme strukturu, nad kterou probíhá evoluce.

Cílem této práce je prověřit aplikovatelnost nové metody v oblasti genetických algoritmů. V této nové metodě neprobíhá evoluce nad přímou reprezentací vyvíjeného jedince, nýbrž nad množinou pravidel pro přepisovací systém, jejichž postupnou aplikací dochází k růstu řetězce ze zvoleného embrya. Vzniklé řetězce pak kódují fenotypy jedinců. Motivací pro průzkum této metody je hypotéza, že pokud by byla nalezena množina pravidel, pomocí které je možno tímto způsobem sestrojit plně funkčního jedince, mohlo by z těchto pravidel teoreticky být možné extrahovat obecný princip pro tvorbu větších jedinců plnících požadovanou funkci. Zkoumaná metoda tak má potenciál vyhnout se problému škálovatelnosti reprezentace a částečně také problému škálovatelnosti evaluace.

Práce je členěna následovně: v kapitole 2 jsou uvedeny poznatky z oblasti řadicích sítí, v kapitole 3 jsou pak popsány principy přepisovacích systémů, obzvláště L-systémů, a uvedeno navržené rozšíření třídy IL-systémů, MDIL-systémy. Kapitola 4 pojednává o oblasti evolučních algoritmů a biologií inspirovaném vývoji.

V kapitole 5 je popsána navržená metoda, která využívá biologií inspirovaného vývinu založeného na MDIL-systému pro evoluční návrh obecných struktur. Kapitola 6 popisuje experimenty, které byly provedeny s implementací navržené metody v oblasti návrhu řadicích sítí, a uvádí dosažené výsledky ve srovnání s aktuálně nejlepší srovnatelnou metodou [4]. Práci uzavírá kapitola 7, která shrnuje dosažené výsledky a uvádí cíle, kterými by se mohl ubírat další vývoj v oblasti zkoumání navržené metody.

Kapitola 2

Řadicí sítě

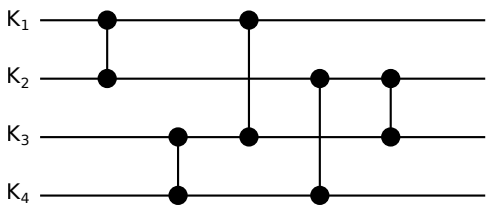
Řadicí sítě jsou jedním z možných přístupů k řešení problému řazení posloupností čísel. Počátky studia řadicích sítí se datují do roku 1954 a jejich problematiku detailně popsal Donald Knuth v [11].

Řadicí síť se skládá z jistého počtu tzv. *komparátorů* (comparator modules), což jsou jednotky se dvěma vstupy a dvěma výstupy. Komparátory provádějí tzv. *compare-swap* operace, kde výstupem komparátoru je nižší ze vstupních hodnot na prvním výstupu a vyšší ze vstupních hodnot na druhém výstupu. Komparátory tak formálně realizují funkci

$$f(x, y) = (\min(x, y), \max(x, y)).$$

Každá řadicí síť pak určuje posloupnost compare-swap operací, které musí být postupně provedeny nad vstupní posloupnosti délky n. Hodnota n tedy označuje počet prvků řazené posloupnosti, v terminologii řadicích sítí ji nazýváme *šířkou* řadicí sítě. Pokud je posloupnost compare-swap operací správná, dojde k seřazení libovolné vstupní posloupnosti. Důležitým poznatkem je fakt, že, na rozdíl od některých jiných přístupů k řazení, posloupnost compare-swap operací v rámci dané řadicí sítě dané velikosti je totožná pro všechny možné vstupní posloupnosti řazených hodnot, a nezáleží tak na hodnotách řazených prvků, ani na jejich uspořádání ve vstupní posloupnosti [11].

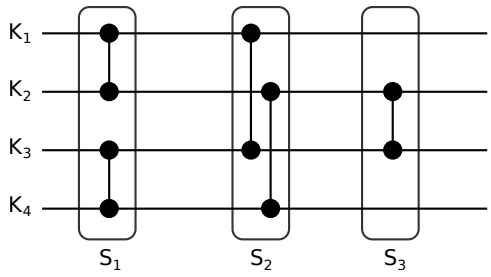
Pro vizualizaci řadicích sítí lze použít zobrazení uvedené na obrázku 2.1, které je používáno v [11]. Vodorovné linie zde reprezentují jednotlivé prvky řazené posloupnosti během průchodu řadicí sítí, svislé úsečky pak představují operace compare-swap nad dvěma z prvků řazené posloupnosti. Řadicí síť lze také reprezentovat jako posloupnost komparátorů, přičemž komparátor je definován dvojicí indexů prvků posloupnosti, které porovnává. Pro síť uvedenou na obrázku 2.1 by tato posloupnost byla $C = ((1, 2), (3, 4), (1, 3), (2, 4), (2, 3))$.



Obrázek 2.1: *Ukázka vizualizace řadicí sítě z [11].*

Výhodou řadicích sítí je dále fakt, že operace compare-swap, které vzájemně nezávisí na svých výsledcích, lze provádět paralelně. Jedná se tedy o koncept vhodný pro obvodovou

(hardwarovou) realizaci. Komparátory řadicí sítě proto rozdělíme do skupin, tzv. *paralelních hladin*, přičemž komparátory mohou náležet do stejné paralelní hladiny, jen pokud jsou jejich vstupy a výstupy vzájemně nezávislé. Komparátory řadicí sítě na obrázku 2.1 tak lze seskupit do tří skupin S_1, S_2 a S_3 způsobem, který je znázorněn na obrázku 2.2. Operace v rámci každé z těchto skupin mohou být provedeny paralelně, zpoždění znázorněné řadicí sítě tedy bude rovno trojnásobku trvání operace compare-swap.



Obrázek 2.2: *Skupiny operací compare-swap, které lze provádět paralelně.*

2.1 Platnost řadicí sítě

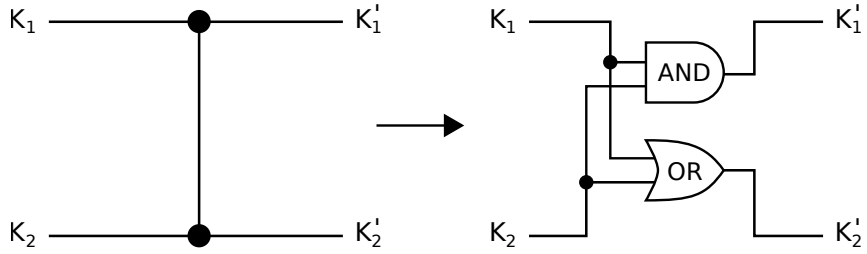
Řadicí síť s n vstupy označíme jako *platnou*, pokud správně seřadí libovolnou posloupnost délky n. Tuto vlastnost by bylo možno ověřit použitím ověřované sítě k seřazení všech $n!$ různých posloupností a ověření, že došlo k jejich seřazení. Časová složitost tohoto přístupu by však však byla $\mathcal{O}(n!)$, což je z výpočetního hlediska silně nevýhodné. Použitím tzv. *teorému Z*[11], také známého jako *zero-one principle*, lze časovou složitost snížit na $\mathcal{O}(2^n)$. Podle tohoto teorému libovolná řadicí síť, která správně seřadí všech 2^n možných vstupních posloupností složených pouze z nul a jedniček, také správně seřadí posloupnost libovolných[1] vstupních hodnot délky n.

Během ověřování řadicí sítě lze také odhalit komparátory, které jsou v řadicí síti zbytečné. Pokud při seřazování alespoň jedné z 2^n posloupností určitý komparátor změnil pořadí hodnot, pak byl tento komparátor použit a jeho přítomnost v řadicí síti je nutná pro správnou funkci této sítě. V opačném případě se jedná o nepoužitý komparátor, jehož přítomnosti není v řadicí síti třeba. Tyto komparátory jsou v dalším textu označovány jako jako *redundantní* či *zbytečné*.

2.1.1 Evaluace řadicích sítí

Díky zero-one teorému můžeme bez újmy na obecnosti uvažovat všechny vstupní hodnoty jako binární. Pro ověření, zda je řadicí síť plně funkční, je tedy třeba tuto síť použít k seřazení všech 2^n možných binárních vstupních posloupností. Jelikož uvažujeme všechny vstupy jako binární, lze pro účely evaluace definovat komparátor pomocí hradel AND a OR tak, jak je uvedeno na obrázku 2.3. Hradlo AND vybírá nižší z obou prvků a hradlo OR vyšší z obou prvků, funkce komparátoru je tedy pro binární vstupy zachována. Libovolnou řadicí síť pro n prvků tak lze reprezentovat kombinačním logickým obvodem o n vstupech a n výstupech obsahujícím pouze hradla AND a OR.

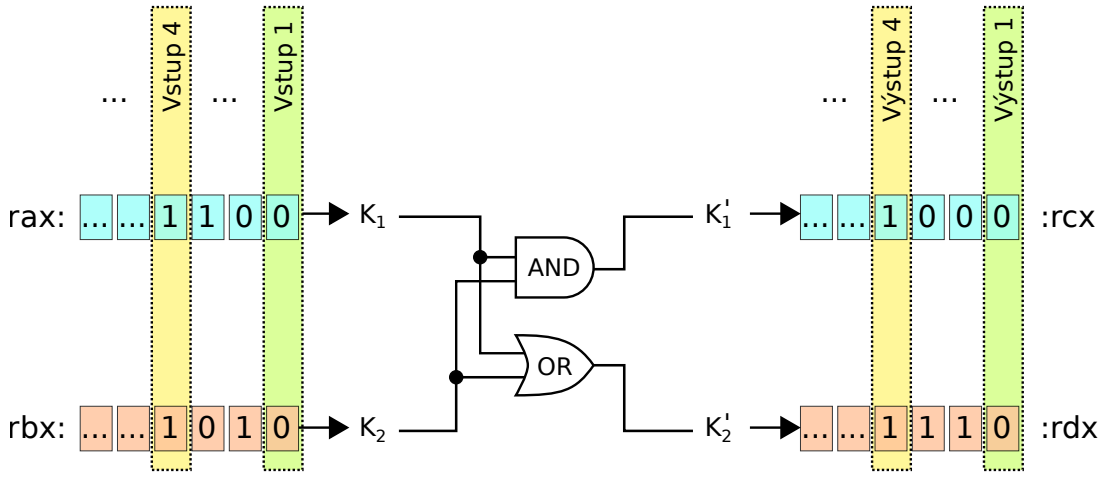
[1] Jediným požadavkem je existence úplného uspořádání nad vstupními hodnotami.



Obrázek 2.3: *Ukázka reprezentace řadicí sítě kombinačním logickým obvodem pro účely evaluace.*

Výhodou reprezentace řadicí sítě jako logického obvodu je možnost použití tzv. *bitově paralelní simulace*, často užívané při evolučním návrhu kombinačních obvodů. Naivní implementace evaluace řadicí sítě by v této fázi pro každý testovací vstup a pro každý komparátor řadicí sítě provedla jednou operaci AND a jednou operaci OR. Pro řadicí síť o velikosti $n = 12$ by tedy bylo třeba pro každý komparátor sítě provést $2 \cdot 2^{12} = 2 \cdot 4096$ operací AND a OR dohromady.

Bitově paralelní simulace využívá skutečnosti, že na hadrwarové úrovni lze v procesoru do jednoho registru uložit více bitů zároveň a instrukce AND a OR provést nad všemi bity registru najednou. Například na typickém procesoru s 64 bitovými registry lze pro jeden komparátor vyhodnotit 64 vstupních posloupností pouhými dvěma instrukcemi. Pro jeden komparátor zmíněné řadicí sítě o velikosti $n = 12$ je tedy třeba pouze $2 \cdot \frac{4096}{64} = 2 \cdot 64$ instrukcí AND a OR dohromady. Ještě větší zrychlení lze získat použitím SIMD registrů instrukčních sad SSE2 a AVX2, pokud jsou podporovány. Užití paralelní simulace pro evaluaci řadicí sítě je ilustrováno na obrázku 2.4. Řádky symbolizují obsahy příslušných registrů, sloupce protínající tyto řádky pak představují příslušné vstupní posloupnosti.



Obrázek 2.4: *Ukázka bitově paralelní simulace logického obvodu pro evaluaci řadicí sítě pomocí registrů rax, rbx, rcx a rdx.*

2.2 Optimální řadicí sítě

V oblasti řadicích sítí lze provádět optimalizace s ohledem na celkový počet compare-swap operací vyžadovaných danou sítí, tedy s ohledem na počet komparátorů obsažených v dané síti. Lze také optimalizovat s ohledem na čas potřebný k paralelnímu seřazení vstupní

posloupnosti (tzv. *zpoždění*), tedy s ohledem na počet skupin compare-swap operací, které jsou na sobě vzájemně nezávislé a které lze tedy provést paralelně. Aktuálně známé meze optimálních řadicích sítí o $n \leq 20$ vstupech jsou uvedeny níže v tabulce 2.1, převzaté z [7]. Meze počtu komparátorů jsou uvedeny v řádku označeném s_n, meze optimálních zpoždění jsou uvedeny v řádku označeném t_n.

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
s_n	0	1	3	5	9	12	16	19	25	29	35 33	39 37	45 41	51 45	56 49	60 53	71 58	78 63	86 68	92 73
t_n	0	1	3	3	5	5	6	6	7	7	8	8	9	9	9	9	10	11 10	11 10	11 10

Tabulka 2.1: *Aktuálně známé horní a spodní meze optimálních řadicích sítí pro $n \leq 20$ [7]. Řádek s_n obsahuje meze počtu komparátorů, řádek t_n meze zpoždění. V případech, kdy přesné hodnoty nejsou známy, jsou uvedeny nejlepší známé horní a spodní meze.*

2.3 Historie vývoje efektivních řadicích sítí

Knuth uvádí [11, str. 225], že řadicí sítě byly poprvé zkoumány v roce 1954, přičemž efektivity bylo dosahováno užitím dovednosti tvůrce. Později bylo odvozeno množství schémat pro konstrukci efektivních řadicích sítí. Mezi nejznámější z nich patří Bubble Sort a Batcherův Bitonic Merge Sort, kterým se blíže věnuje sekce 2.4. Optimální řadicí sítě pro $n \leq 16$ vstupů uvádí Knuth již ve vydání [11] z roku 1973, optimalita některých z nich však nebyla dokázána až do roku 2014 [6].

V roce 1983 publikovali M. Ajtai, J. Komlós a E. Szemerédi teoretickou metodu pro konstrukci řadicích sítí se zpožděním $\mathcal{O}(\log(n))$, jedná se však pouze o teoretickou konstrukci, jelikož $\mathcal{O}$-notace skrývá velmi vysokou konstantu [11] [17] [6]. Paterson [17] sice dosáhl částečného snížení této konstanty, jeho konstrukce je však, jak sám uvádí, stále vysoce neefektivní pro jakoukoliv prakticky požadovatelnou řadicí síť.

V oblasti návrhu efektivních řadicích sítí našly hojné uplatnění SAT-solvery. Roku 1989 použil Parberry na superpočítači CRAY-2 SAT-solver k důkazu optimality prvních deseti Knuthem uváděných sítí [16]. Bundala a Závodný [6] použili SAT-solver k důkazu optimality zbylých Knuthem uváděných sítí. SAT kódování problému řadicích sítí dále vylepšují Codish a spol. [7], čímž posouvají známé meze optimality pro řadicí sítě o 17, 19 a 20 vstupech. V roce 2017 použil T. Ehlers SAT-solver k nalezení nových 23 a 24 vstupých řadicích sítí, čímž posunul známé meze optimality pro řadicí sítě o těchto velikostech [8].

Pro hledání efektivních řadicích sítí jsou také hojně využívány evoluční algoritmy. Již některé z Knuthem uváděných řadicích sítí vycházejí z použití evolučních algoritmů. Například síť uvedená pro $n = 13$ vstupů byla vytvořena H. Juillém, a to počítačovou simulací evolučního procesu. Uvedená síť nemá žádnou patrnou logickou strukturu komparátorů, přesto, či možná právě proto, překonává dřívější člověkem vytvořené řadicí sítě. Valsalam a Miikkulainen použili v roce 2013 evoluční algoritmus nad booleovskou reprezentací řadicích sítí [23]. Výsledkem byly sítě, které překonaly parametry tehdy známých řadicích sítí.

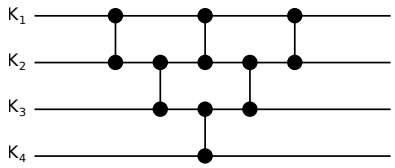
L. Sekanina [21] použil evoluční algoritmus pro hledání konstruktorů nekonečně rostoucích řadicích sítí. Pod pojmem konstruktory se v této práci myslí programy, které obdrží

na vstupu řadicí síť a na výstupu vytvoří řadicí síť pro více vstupů. Tento přístup rozšířil a aplikoval M. Bidlo [4], přičemž nalezl nové efektivní konstrukce pro nekonečně rostoucí řadicí sítě rostoucí po dvou a po třech vstupech. Tato práce navazuje na práci L. Sekaniny a M. Bidla, přičemž namísto konstruktorů používá přepisovací systémy.

2.4 Metody pro konstrukci řadicích sítí různých velikostí

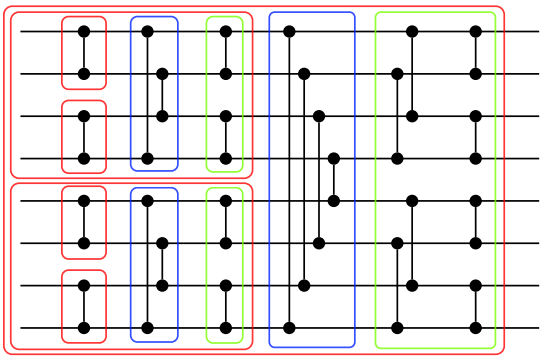
Přirozeným požadavkem je schopnost umět sestavit řadicí síť pro posloupnost požadované velikosti. Pro tento účel bylo vytvořeno několik schémat. Níže jsou uvedeny metody postavené na principech Insertion/Selection sort a Bitonic Merge sort.

Pravděpodobně nejjednodušším přístupem je použití principu vkládání či výběru prvků. Oba principy jsou popsány v [11] a aplikací obou vznikne totožná struktura řadicí sítě, odpovídající algoritmu Bubble Sort. Touto metodou lze tvořit řadicí sítě pro posloupnosti libovolné délky. Síť pro čtyřprvkové posloupnosti vytvořená touto metodou je zobrazena na obrázku 2.5. Lze odvodit [11], že pro n-prvkovou posloupnost vytvoří metoda $\frac{n(n-1)}{2}$, tedy $\mathcal{O}(n^2)$, komparátorů. Zpoždění vytvořené sítě je $2n-3$ [11].



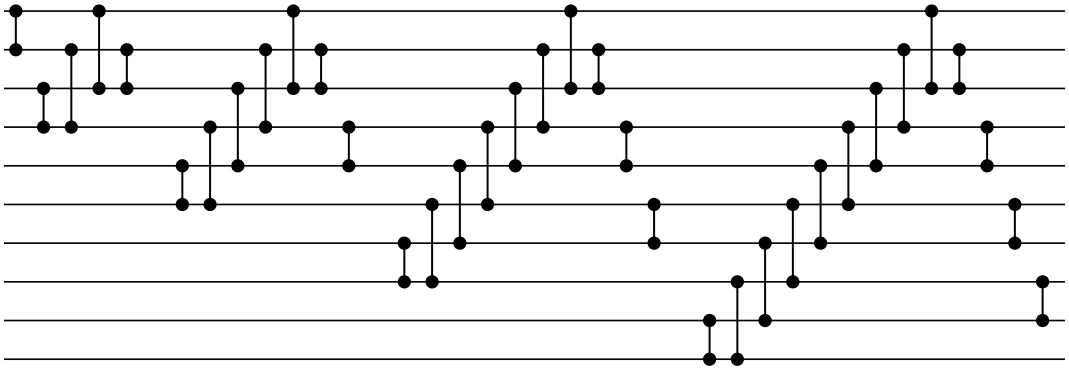
Obrázek 2.5: *Insert / Select sort.*

Jiným existujícím schématem je tzv. Bitonic Merge sort [2]. Tato metoda pracuje na principu setřiďování dvou neklesajících (monotónních) posloupností, odtud název Bitonic sort. Metoda je schopna konstruovat řadicí sekvence pro posloupnosti o délce 2^n, přičemž pro jejich konstrukci využívá řadicí sítě o $2^{(n-1)}$ vstupech. Konstrukce řadicí sítě metodou Bitonic Merge sort je zobrazena na obrázku 2.6. Červeně označené skupiny komparátorů představují plně funkční řadicí sítě, a to pro 2, 4 a 8 vstupů. Modře a zeleně označené skupiny komparátorů zajišťují setřídění dvou monotónních posloupností do jedné monotónní posloupnosti. Pro n-prvkovou posloupnost vytvoří metoda $\mathcal{O}(n{\cdot}\log(n)^2)$ komparátorů, zpoždění vytvořené sítě bude $\mathcal{O}\left(\log(n)^2\right)$ [2].



Obrázek 2.6: *Bitonic Merge sort.*

V rámci své práce představil M. Bidlo metodu pro konstrukci nekonečně rostoucích řadicích sítí, které svými parametry překonávají sítě tvořené již zmíněnou konvenční metodou vkládání prvků [4]. Tato metoda byla získána evolučním přístupem a ve své původní podobě generuje evolučně nalezené řešení při každém zvětšení sítě jeden přebytečný komparátor navíc. Ručně bylo toto řešení upraveno tak, aby představená metoda žádné přebytečné komparátory negenerovala. Metoda je uvedena na obrázku 2.7.



Obrázek 2.7: *Metoda pro konstrukci rostoucích řadicích sítí představená M. Bidlem [4].*

Kapitola 3

Přepisovací systémy

Abstraktní přepisovací systém je definován [3] jako $\mathcal{A} = (A, \{\rightarrow_\alpha | \alpha \in I\})$, tedy jako dvojice skládající se z množiny A a množiny binárních relací $\rightarrow_\alpha$ nad A, kde I je množina indexů těchto relací. Tyto binární relace nazýváme *přepisovací relace* [3]. Pro účely této práce budeme uvažovat A jako množinu řetězců Σ^* nad abecedou Σ a přepisovací relace tvaru $\Sigma^* \rightarrow \Sigma^*$. Dále, pokud $(a, b) \in \rightarrow_\alpha$ pro $a, b \in \Sigma^*$, píšeme $a \Rightarrow b$ a říkáme, že a *derivuje* b, nebo že b je *derivací* a.

Posloupnost derivací postupně rozvíjí daný počáteční řetězec, čímž získáváme věty jazyka definovaného daným přepisovacím systémem. K přepisovacímu systému se navíc váže předepsaný způsob, kterým jsou přepisovací relace aplikovány. Tento způsob označme jako *derivační krok*, či zkráceně *krok*. Derivační krok může a nemusí být deterministický.

Přepisovací relaci obvykle definujeme prostřednictvím *přepisovacích pravidel*, či zkráceně *pravidel*, která uplatňujeme pro přepis podřetězce derivovaného řetězce na jiný podřetězec. Klasickým příkladem přepisovacích systémů jsou například gramatiky, které rozlišují *terminální* a *neterminální* symboly a v rámci jednoho derivačního kroku aplikují právě jedno přepisovací pravidlo. Pro tuto práci však má mnohem větší význam kategorie přepisovacích systémů označovaná jako L-systémy. Této kategorii se blíže věnuje následující sekce.

3.1 L-systémy

Jako L-sytémy označujeme druh přepisovacích systémů, který poprvé popsal A. Lindenmayer v [14]. Na rozdíl od klasických gramatik v L-systémech nedochází k aplikaci přepisovacích pravidel v rámci derivačního kroku sekvenčně, nýbrž paralelně. Paralelní aplikace pravidel lépe odpovídá růstu organismů formou buněčného dělení, pro jehož modelování byly L-systémy navrženy. Aplikací přepisovacích pravidel na prvotní řetězec, tzv. *axiom*, dochází k vývoji tohoto řetězce, což reflektuje vývoj živého organismu.

Formálně jako 0L-systém označujeme trojici $G = (V, \omega, P)$, kde V označuje abecedu (konečnou množinu symbolů), $\omega \in V^+$ *axiom*, a $P \subset V \times V^*$ množinu přepisovacích pravidel tvaru $a \rightarrow \chi$ [18]. Pravidla P vytvářejí endomorfismus na V^* [19, str. 11]. Tento systém pak nazýváme bezkontextovým L-systémem. Systém dále nazveme deterministickým (D0L-systémem), právě když pro každé $a \in V$ existuje právě jedno $\chi \in V^*$ takové, že $a \rightarrow \chi$.

Existuje mnoho tříd L-systémů [20], za důkladnější rozbor však z hlediska této práce stojí především kontextové 2L-systémy. Kontextové L-systémy byly navrženy především s cílem modelovat pohyb informace (např. v podobě hormonů) v rostlinných organismech. 2L-systém [18] používá přepisovací pravidla tvaru $a_l\langle a\rangle a_r \rightarrow \chi$, kde $a_l, a_r \in V$ a kde symbol

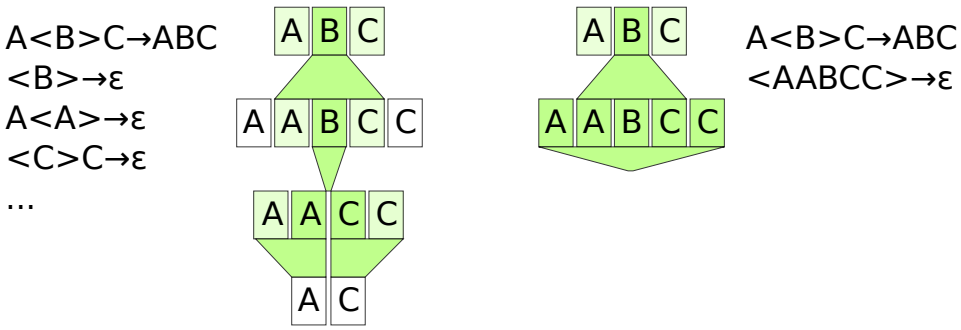
$a \in V$ může být přepsán na řetězec $\chi \in V^*$, právě když jej předchází symbol a_l a následuje symbol a_r. Symboly a_l a a_r tak tvoří levý a pravý kontext, ve kterém může být symbol a přepsán na řetězec χ.

Byl také vytvořen širší pojem pro třídu *IL-systémů* [19, str. 281], která v sobě zahrnuje třídy OL, 1L a 2L-systémů a která umožňuje jako levé a pravé kontexty použít řetězce namísto pouhých symbolů[18]. Třída IL-systémů je také označována jako *(k,l)-systémy*, kde k označuje délku levého a l pravého kontextu. Tato třída tedy umožňuje používat pravidla tvaru $V^*\langle V\rangle V^* \to V^*$.

3.2 MDIL-systém

Jak bylo uvedeno, L-systém, jak v podobě, v jaké jej představil Lindenmayer [14], tak v rozšíření představovaném třídou IL-systémů [19], povoluje na levé straně přepisovacích pravidel pouze jeden symbol. Proto byl v rámci této práce vytvořen koncept nazvaný *MDIL-systém* (Multisymbolový Deterministický IL-systém), který představuje rozšíření třídy IL-systémů o možnost použití řetězců libovolné délky na levé straně přepisovacích pravidel. Toto rozšíření tedy umožňuje používat pravidla tvaru $V^*\langle V^*\rangle V^* \to V^*$.

Tento systém tak, na rozdíl od IL-systémů, umožňuje vytvořit pravidlo, jehož levá strana bude mít větší délku než pravá strana. V oblasti developmentu v genetických algoritmech může tato vlastnost být přínosem. Například lze takovýmto pravidlem z vyvíjené struktury v jisté fázi růstu v jediném kroku odstranit podčást, která pozbyla užitečnosti. Tato skutečnost je znázorněna na obrázku 3.1. V oblasti biologie lze tento jev nalézt například při vývinu lidského plodu, kdy v určité fázi odumřou buňky vyplňující prostor mezi budoucími prsty.



Obrázek 3.1: *Srovnání IL-systémů (nalevo) s MDIL-systémy (napravo). MDIL-systém umožňuje v jediném kroku odstranit část rostoucí struktury.*

S tímto rozšířením však přicházejí jisté problémy z hlediska zajištění potřebného determinismu přepisovacího systému. Může totiž nastat situace, kdy lze na stejný symbol (v rámci obecně různých podřetězců) aplikovat více pravidel. Tradiční přístup k řešení tohoto problému pokládá na množinu pravidel příslušná omezení, aby k této situaci nemohlo dojít[19, str. 282]. Tento přístup však pro použití v evolučních algoritmech není příliš vhodný, zejména z důvodu přílišné složitosti mutačních operátorů, které by tato omezení musely respektovat. Proto byl pro MDIL-systém zvolen jiný přístup, popsaný níže.

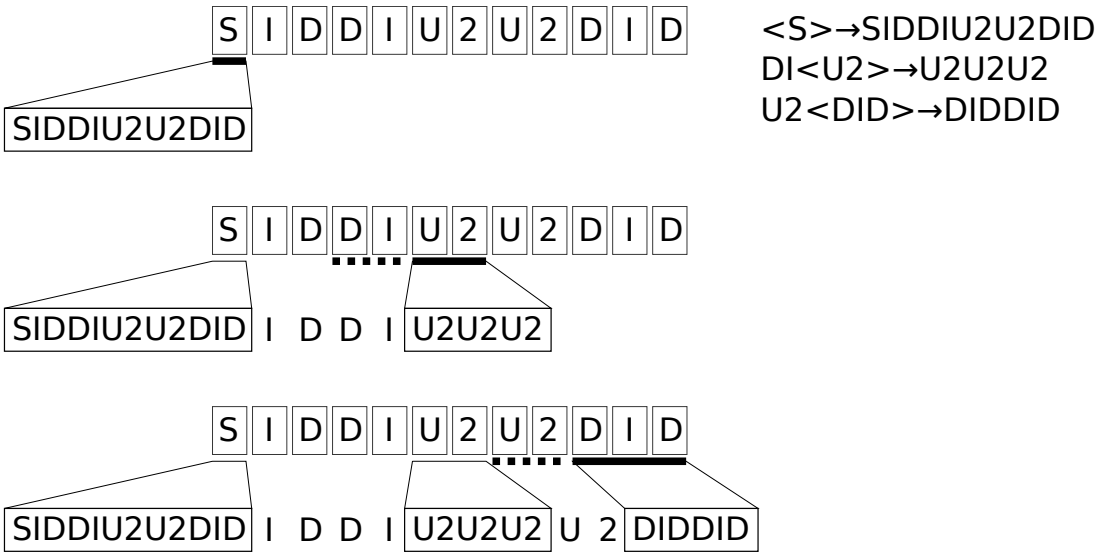
Formálně definujeme MDIL-systém jako trojici (V, ω, R), kde V je abeceda a $\omega \in V^+$ axiom. Přepisovací pravidla MDIL-systému tvoří množina $R \subseteq \{(p, w_l, w, w_r, \chi) : w_l, w, w_r, \chi \in V^*, p \in [1, |R|]\}$. Každé pravidlo je tedy tvaru $p: \ w_l\langle w\rangle w_r \to \chi$, kde p je unikátní

priorita, která nad množinou pravidel vytváří totální ostré uspořádání. Vyžadujeme tedy $\forall r_1, r_2 \in R : (p(r_1) = p(r_2)) \Rightarrow r_1 = r_2$. Právě priorita p hraje klíčovou roli v zaváděném determinismu. Při zápisu pravidel vyjadřujeme prioritu jejich pořadím, přičemž nejprioritnější pravidlo (s nejniší hodnotou p) píšeme na prvním místě.

Derivační krok $\nu_1 \Rightarrow \nu_2$, kde $\nu_1, \nu_2 \in V^*$, probíhá v MDIL-systému tak, že jsou v řetězci ν_1 ve směru zleva doprava vyhledávány levé strany pravidel (w). Pokud je nalezena levá strana w pravidla r a v řetězci ν_1 ji přímo předchází w_l a přímo následuje w_r, pak je podřetězec w v řetězci ν_1 nahrazen pravou stranou pravidla r, tj. řetězcem χ. Vyhledávání dalšího w poté pokračuje o $|w|$ symbolů dále, tedy ihned za přepsanou částí. Pokud je nalezeno více pravidel splňujících předchozí podmínku, uplatní se z nich pouze pravidlo s nejnižší hodnotou p. Symboly řetězce ν_1, které nebyly součástí některého přepsaného w, jsou ponechány beze změny.

Fungování MDIL-systému ilustruje obrázek 3.2. Prvotní řetězec (axiom) „SIDDIU2U2DID“ odpovídá dle reprezentační funkce uvedené dále v sekci 5.2.3 platné čtyřvstupé řadicí síti konstruované metodou nalezenou M. Bidlem a uvedenou v kapitole 2. Tento řetězec je v jediném kroku MDIL-systému přepsán na řetězec „SIDDIU2U2DIDIDDIU2U2U2U2DIDDID“, který odpovídá platné řadicí síti pro šest prvků.

V rámci derivačního kroku jsou ve směru zleva doprava vyhledávány podřetězce, které se vyskytují na levé straně některého z pravidel. Jako první je nalezen podřetězec „S“, který tvoří levou stranu jednoho z pravidel. Toto pravidlo je tedy aplikováno a řetězec je přepsán pravou stranou pravidla. Jako další podřetězec, který je zároveň levou stranou pravidla, je nalezen „U2“. Je ověřeno, že jeho levý kontext odpovídá symbolům před nalezeným podřetězcem, a poté je přepsán. Další vyhledávání pokračuje za přepsanou částí. Jako další je nalezen podřetězec „U2“, jeho levý kontext však neodpovídá žádnému z pravidel, které má „U2“ na své levé straně, a tak není provedeno žádné přepsání. Jako poslední je nalezen podřetězec „DID“, jeho levý kontext je ověřen a podřetězec je přepsán.



Obrázek 3.2: *Ilustrace jednoho derivačního kroku MDIL-systému.*

Kapitola 4

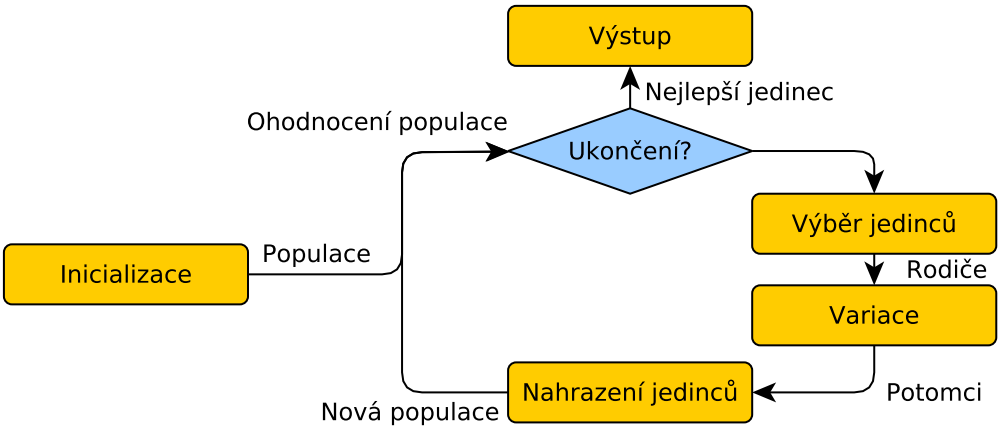
Evoluční algoritmy

Pro úlohu prohledávání stavového prostoru existuje řada algoritmů, mezi něž patří také evoluční algoritmy. Ty jsou pravděpodobně nejznámější podmnožinou biologií inspirovaných algoritmů. Algoritmy této podmnožiny simulují průběh biologií inspirované evoluce pro „vyšlechtění“ dobrých řešení daného problému [5]. Mezi evoluční algoritmy řadíme především *genetické algoritmy* (*GA*), *genetické programování* (*GP*), *evoluční strategie* (*ES*) a *evoluční programování* (*EP*) [22].

Evoluční algoritmy se vyznačují následujícími charakteristikami [5]:

- Pracují nad populací jedinců.
- Obsahují mechanismy pro výběr jedinců z populace.
- Uchovávají úspěšné jedince.
- Vytvářejí variace jedinců.

Základní meta-algoritmus [5, str. 18] znázorňuje obrázek 4.1. Krok výběru rodičů má tendenci vybírat jako rodiče ty jedince, kteří vykazují lepší vlastnosti. Krok variace vytváří nové jedince, kteří jsou odvozeni z jejich rodičů, a jsou jim tedy podobní. Jedinci v populaci představují pokusná řešení daného problému, tedy body v prohledávaném stavovém prostoru. Kvalita těchto pokusných řešení se v průběhu algoritmu zvyšuje [5].



Obrázek 4.1: *Evoluční cyklus.*

Jedinci jsou v evolučním algoritmu reprezentováni *genotypem*, který obsahuje kódovanou podobu kandidátního řešení (fenotypu). Výběr rodičů pro vytvoření potomků je řízen tzv. *fitness funkcí*, jejíž hodnota odpovídá kvalitě řešení, které daný jedinec představuje. Fitness funkce ovšem pracuje nad *fenotypem*, který je projevem genotypu, a pro ohodnocení jedince je tedy třeba převést genotyp na jeho projev, fenotyp. Mapování genotypů na fenotypy může být *přímé* či *nepřímé* [4]. Některé evoluční algoritmy mohou pro mapování použít funkci identity a nerozlišovat tak fenotypy a genotypy. Z genotypů zvolených rodičů lze vytvářet potomky (variace) za použití tzv. *genetických operátorů*.

4.1 Genetické algoritmy

Genetické algoritmy [9] přímo rozlišují genotypy a fenotypy jedinců. Pracují nad populaci jedinců, jejichž genotypy mají obvykle konstantní délku a musí být pro ohodnocení fitness funkcí převedeny na fenotypy. Kanonický genetický algoritmus [5, str. 23] je uveden níže jako algoritmus 1.

Algoritmus 1: Kanonický GA

1 Urči kódování genotypu pro kandidátní řešení a příslušnou fitness funkci.
2 Vytvoř počáteční populaci genotypů.
3 Dekóduj všechny genotypy na fenotypy a vyhodnoť fitness pro každého z n jedinců v populaci.
4 **repeat**
5 Vyber n jedinců z aktuální populace (rodiče) a umísti je do množiny rodičů.
6 **repeat**
7 Vyber náhodně dva rodiče z množiny rodičů.
8 S pravděpodobností p_{cross} proveď křížení genotypů rodičů a vytvoř dva nové potomky. Jinak vytvoř dva potomky, kteří jsou kopiemi svých rodičů.
9 S pravděpodobností p_{mut} proveď mutaci každého prvku genotypů obou potomků.
10 **until** *je vytvořeno n potomků*
11 Nahraď původní populaci nově vytvořenou populací.
12 **until** *ukončovací podmínka byla splněna*

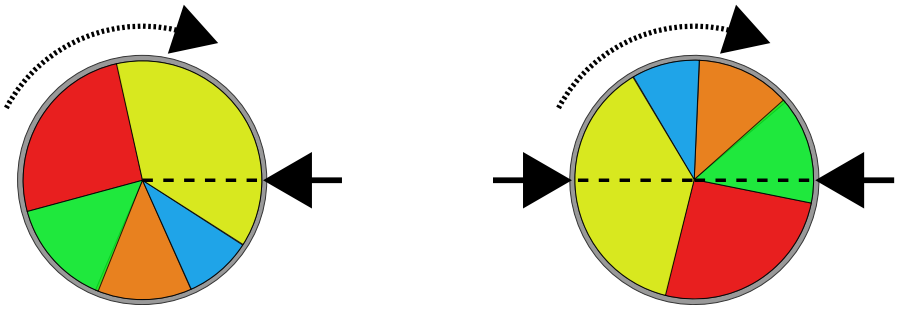
Kanonický genetický algoritmus lze tedy popsat jako algoritmus, který vytváří z aktuální populace jedinců novou populaci, a to za použití stochastických operátorů. V obecné rovině se jedná o operátory *selekce*, *křížení* a *mutace*. V následujících odstavcích jsou popsány funkce těchto operátorů, jakožto i některé jejich známé implementace.

4.1.1 Operátor selekce

Účelem operátoru selekce je zvolit z aktuální populace určitý počet jedinců, kteří přežijí a budou mít potomky. Při výběru jsou upřednostňováni úspěšnější jedinci, tedy ti s vyšším ohodnocením fitness. Důraz na výběr nejlepších jedinců nazýváme jako *selekční tlak* [1], ten se může lišit v závislosti na implementaci operátoru selekce. Funkce selekce tedy v genetickém algoritmu odpovídá tomu, co je v Darwinově teorii evoluce označeno jako *princip přirozeného výběru* [5, str. 22]. Jelikož operátor selekce pracuje s hodnotami fitness, nezáleží jeho implementace na zvolené reprezentaci genotypu. Mezi nejznámější implementace operátoru selekce patří *ruletový výběr* (roulette wheel selection) a *turnajový výběr* (tournament selection).

Ruletový výběr

Při ruletovém výběru je každému z jedinců přiřazen výsek simulovaného ruletového kola, přičemž velikost výseku odpovídá hodnotě fitness daného jedince [22, str. 100]. Roztočení a zastavení ruletového kola je simulováno generováním náhodné hodnoty v intervalu $\langle 0; 2\pi)$ a ten jedinec, jehož výsek obsahuje vygenerovaný úhel, je vybrán jako rodič. Pro výběr více rodičů je možno roztočit kolo ještě jednou (generovat novou náhodnou hodnotu), nebo k ruletovému kolu umístit více výherních pozic (k vygenerovanému úhlu přičíst pevně stanovené úhly a zvolit jedince na výsledných úhlech). Ruletový vývěr je ilustrován na obrázku 4.2.



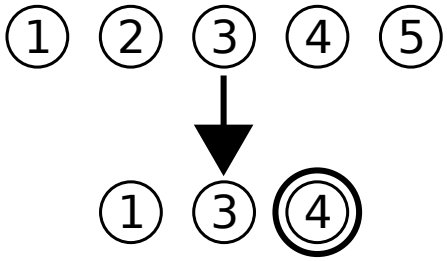
Obrázek 4.2: *Ruletový výběr. Vlevo: výběr dvou rodičů dvounásobným otočením rulety. Vpravo: výběr dvou rodičů jedním otočením rulety.*

Nevýhodou této implementace operátoru selekce je, že v prvních generacích genetického algoritmu se populace bude obvykle skládat zejména z jedinců s nízkou hodnotou fitness. Pokud se mezi nimi vyskytne jedinec s vyšší hodnotou fitness, zabere tento jedinec většinu ruletového kola, což způsobí konvergenci genetického algoritmu do bodu daného tímto jedincem [22, str. 100]. Evoluce tak přijde o možnost zkoumat potenciálně mnohem lepší řešení, neboť zmíněný jedinec sice mohl mít daleko vyšší hodnotu fitness než jeho kolegové, přesto však v porovnání s ideálním řešením mohla tato hodnota být velmi špatná a jedinec tak mohl představovat pouze lokální optimum. Tento jev označujeme jako problém *předčasné konvergence* (premature convergence) [22].

Turnajový výběr

Jedna z možností, jak se vyhnout problému předčasné konvergence, je použití turnajového výběru jako operátoru selekce [22]. Díky tomu, že je v tomto případě prováděn výběr nejlepšího jedince z relativně malé podskupiny populace, dochází k mnohem jemnější aplikaci selekčního tlaku.

Při turnajovém výběru je nejdříve z aktuální populace náhodně vybráno k jedinců, přičemž $k \in \langle 2; N\rangle$, kde N označuje počet jedinců v populaci. Z této skupiny k jedinců je poté vybrán ten s nejvyšší hodnotou fitness. Nižší hodnoty k působí nižší selekční tlak, vyšší hodnoty k působí vyšší tlak [5]. Pro $k = N$ je vítězem turnaje vždy jedinec s nejvyšší hodnotou fitness. Turnajový výběr je ilustrován na obrázku 4.3.



Obrázek 4.3: *Příklad turnajového výběru pro $k = 3$. Z pěti ohodnocených prvků byly náhodně vybrány tři, největší z nich je vítězem.*

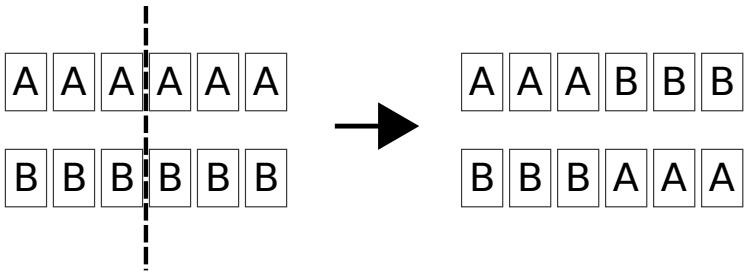
Elitismus

Elitismus není sám o sobě operátorem výběru, nýbrž jeho rozšířením. Toto rozšíření je snahou řešit situaci, kdy nově vzniklý jedinec, jehož fitness převyšuje ostatní jedince, není zahrnut do následující generace. Tato situace může nastat vlivem mutací, křížení, či stochastické povahy operátoru selekce. Elitismus spočívá v zajištění, že jedinec s nejvyšší hodnotou fitness se vždy objeví v následující generaci [22, str. 101]. Elitismus lze také rozšířit na k nejlepších jedinců namísto jediného. Některé zdroje, například [22], doporučují použít určitou formu elitismu v každém genetickém algoritmu.

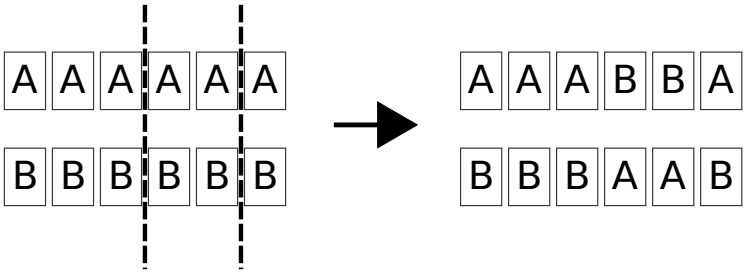
4.1.2 Operátor křížení

Operátor křížení vytváří ze zvolených rodičů nového potomka. V populaci tak vzniknou nové genotypy, které jsou vytvořeny z fragmentů jiných genotypů. Dochází tak k prohledávání stavového prostoru. Jelikož operátor křížení pracuje nad genetickou informací jedinců, přesná implementace křížení vychází ze struktury genotypu a musí jí být uzpůsobena. Mezi nejznámější způsoby křížení binárních chromosomů patří *jednobodové*, *vícebodové* a *uniformní* [5, str. 35].

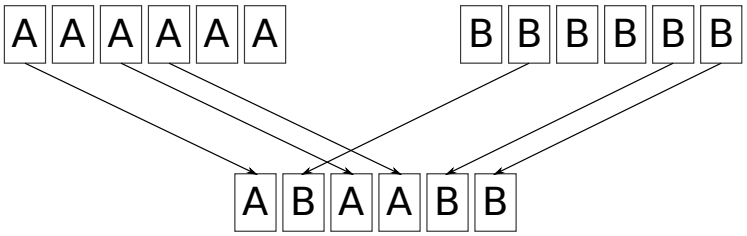
Při jednobodovém křížení je vybráno jedno místo v genotypu, ve kterém dojde ke křížení chromosomů, viz obrázek 4.4. U dvou či vícebodového křížení jsou zvolena dvě či více míst, viz obrázek 4.5. Při uniformním křížení je pro každý gen nově vznikajícího chromosomu náhodně rozhodnuto, od kterého rodiče bude pocházet, viz obrázek 4.6.



Obrázek 4.4: *Vizualizace jednobodového křížení.*



Obrázek 4.5: *Vizualizace dvoubodového křížení.*

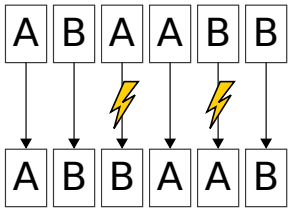


Obrázek 4.6: *Vizualizace uniformního křížení.*

4.1.3 Operátor mutace

Operátor mutace zanáší do genotypů nově vznikajících jedinců náhodné změny. Účelem těchto změn je vnést do genotypů nové hodnoty a tím zkoumat nová místa ve stavovém prostoru. Mutace tak umožňují objevovat nová, potenciálně lepší, řešení. Dříve byl tento operátor považován spíše za mechanismus pro zabránění příliš rychlé konvergenci řešení, v poslední době je mu však přikládána větší důležitost a v některých algoritmech je dokonce používán operátor mutace samotný, bez operátoru křížení [22, str. 104].

Podobně jako křížení je implementace operátoru mutace také silně závislá na zvolené reprezentaci genotypu. Pro binární chromosomy je operátor mutace obvykle implementován jako průchod všemi bity a náhodné rozhodnutí, zda aktuální bit změnit [5, str. 37].

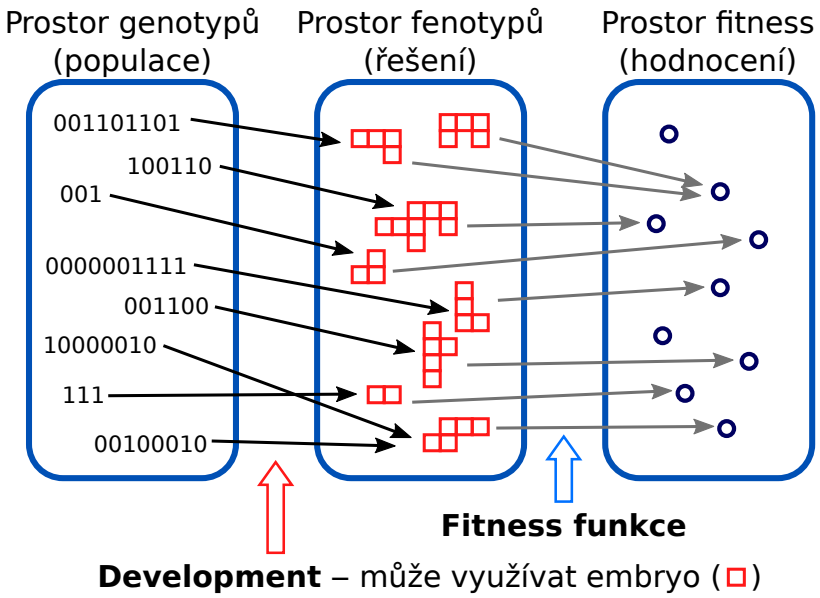


Obrázek 4.7: *Vizualizace mutace.*

4.2 Development

Při použití nepřímé reprezentace může být pro zobrazení genotypu na fenotyp použit výpočetní *development* [13]. Tento přístup vychází z biologického developmentu, tedy z procesu, kterým z prvotního zárodku (*embrya*) vzniká organismus. Mezi existující výpočetní modely pro development patří například celulární automaty, či L-systémy [13, str. 19]. Z hlediska této práce jsou jakožto přepisovací systémy významné právě L-systémy, popsané blíže v sekci 3.1.

Využití nepřímé reprezentace umožňuje zkrátit genotyp jedinců, a vyhnout se tak problému škálovatelnosti reprezentace. Náročnost ušetřená použitím nepřímého mapování je však přenesena do procesu mapování a developmentu [22, str. 350]. Tím vzniká potřeba vytvořit oba procesy co nejefektivnější. Obecné schéma developmentu v evolučních algoritmech ilustruje obrázek 4.8.



Obrázek 4.8: *Obecné schéma developmentu v evolučních algoritmech.*

Výpočetní development se obvykle odehrává po jednotlivých krocích, kdy v každém kroku dojde k určitému vývoji, například ke zvětšení struktury po vzoru buněčného dělení. V klasickém pojetí developmentu je třeba stanovit počet kroků, či jinou podmínku, při jejímž splnění bude development ukončen a vzniklá struktura prohlášena za výsledného jedince. Počet kroků developmentu tak určuje velikost vzniklé struktury. Tento přístup je označován jako *parametrický development* [4].

Existuje však také alternativní přístup, kdy od struktur vzniklých mezi jednotlivými kroky developmentu požadujeme plnou funkcionalitu pro příslušně velikou instanci řešeného problému. Tento přístup nazýváme *kontinuální development* [4]. Při tomto pojetí tedy aplikací kroku developmentu na některou ze struktur získáváme strukturu pro větší instanci řešeného problému.

Kapitola 5

Evoluční návrh řadicích sítí využívající přepisovací systémy

Tato kapitola představuje autorem navrženou metodu pro evoluční návrh struktur, která je hlavním přínosem této práce. Metoda bude nejprve představena obecně, poté budou konkretizovány její komponenty pro použití při návrhu řadicích sítí.

5.1 Metoda pro evoluční návrh obecných struktur využívající přepisovací systémy

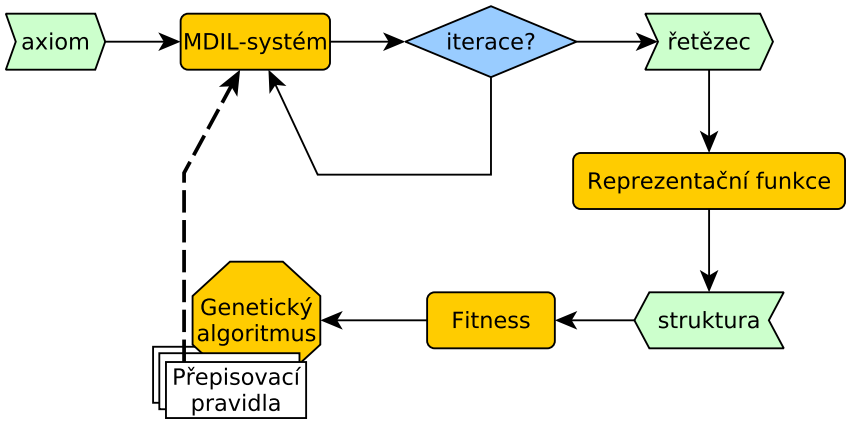
Navržená metoda pracuje na principu genetické evoluce, kde genotypy jedinců představují množiny pravidel přepisovacího systému, konkrétně MDIL-systému. Genetický algoritmus v tomto případě tedy prohledává stavový prostor pravidel ve snaze nalézt taková pravidla, jejichž aplikací při developmentu vznikne struktura s požadovanými vlastnostmi.

L-systémy jsou vhodné zejména proto, že byly vyvinuty pro modelování růstu žijících organismů. Tato vlastnost může významně přispět ke schopnosti metody tvořit struktury, které rostou. Při návrhu metody přicházely v úvahu bezkontextové L-systémy (0L-systémy) a kontextové L-systémy (IL-systémy), zvolený systém vychází právě z kontextových IL-systémů (viz sekce 3.2). Parametrické L-systémy byly po konzultaci s vedoucím práce vyhodnoceny jako nevhodné (např. kvůli složitosti jejich genotypů, nutnosti kontrolovat platnost pravidel po provedení mutace, apod.).

Úspěšnost dané množiny pravidel je ověřena tak, že je tato množina v přepisovacím systému iterativně aplikována na prvotní řetězec, tzv. *axiom*. Počet provedených kroků přepisovacího systému je přitom určen uživatelem. Vstupem prvního kroku je axiom, vstupem každého následujícího kroku je výsledek předchozího kroku. Derivační kroky MDIL-systému tak v genetickém algoritmu realizují výpočetní development. Vzniklý řetězec je poté převeden na strukturu, kterou reprezentuje, a to pomocí uživatelem zvolené reprezentační funkce.

Pro takto vzniklou strukturu je poté vyhodnocena fitness funkce, která ohodnotí, nakolik vzniklá struktura svou funkcí splňuje požadavky. Toto ohodnocení se přímo promítne do ohodnocení příslušné množiny pravidel. Obecný diagram znázorňující jednotlivé kroky základní varianty navržené metody je uveden na obrázku 5.1.

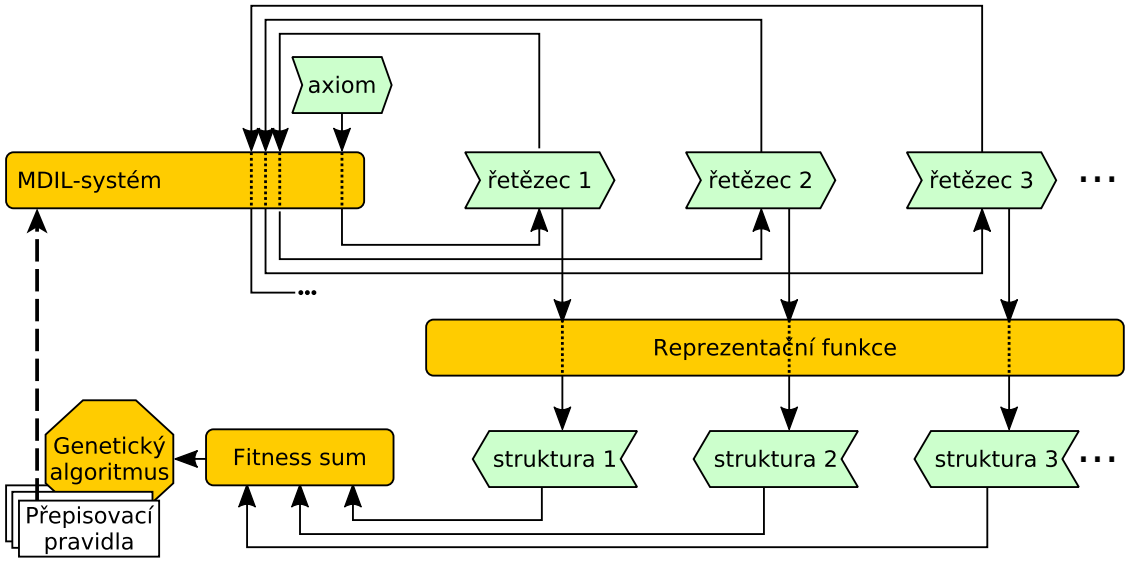
Přesný počet iterací, jakožto i další parametry genetického algoritmu, hrají podstatnou roli, jelikož významně ovlivňují schopnost algoritmu hledat vhodná řešení. Při použití navržené metody pro řešení konkrétního problému je tedy třeba navrhnout vhodnou a efektivní reprezentační funkci a prozkoumat vliv zmíněných parametrů na konkrétní řešený problém.



Obrázek 5.1: *Schéma základní verze navržené metody, implementující parametrický development.*

V případě uvedené základní varianty tedy pomocí fitness funkce hodnotíme pouze jedinou strukturu, a to tu, která vznikla z řetězce, který byl získán daným počtem iterací. Tento přístup odpovídá principu parametrického developmentu (viz sekce 4.2).

Z pohledu této práce se však ukázala být mnohem zajímavější modifikace navržené metody, která realizuje princip kontinuálního developmentu (viz sekce 4.2). V této modifikaci je ve fázi developmentu provedeno několik *iterací*. Během jedné iterace lze provést více derivací (tj. kroků přepisovacího systému), přičemž po každé iteraci je uchován řetězec, který je výsledkem této iterace, a následující iterace vychází z tohoto řetězce. Po i iteracích tak vznikne i textových řetězců reprezentujících struktury z prohledávané domény. Všechny takto získané řetězce jsou prostřednictvím reprezentační funkce převedeny na stejný počet struktur z prohledávané domény. Výsledná hodnota fitness je poté dána součtem hodnot fitness všech takto vzniklých struktur. Uživatelem stanovitelné parametry zahrnují jak počet iterací, tak počet kroků přepisovacího systému na iteraci. V praxi tento přístup klade nemalé nároky na výpočetní čas i prostor, je tedy třeba počet iterací omezit na vhodnou hodnotu. Modifikaci metody pro kontinuální development ilustruje obrázek 5.2.



Obrázek 5.2: *Schéma navržené metody ve verzi pro kontinuální development.*

Verze metody implementující kontinuální development může umožnit objevení takového přepisovacího systému, který v každé iteraci vygeneruje řešení, které je plně funkční na větší instanci problému, než řešení z předchozí iterace. Pokud by tohoto cíle bylo dosaženo, byl by genetický algoritmus schopen najít obecný předpis (množinu pravidel přepisovacího systému), pomocí kterého by bylo možno sestavit strukturu s požadovanými vlastnostmi o neomezené velikosti.

Podstatným přínosem navržené metody ve verzi s kontinuálním developmentem se ukázal být fakt, že schopnost nalezených množin pravidel generovat plně funkční řešení nekončí u velikostí instancí problémů, na kterých byly tyto množiny hodnocené v průběhu evoluce. Často byly tyto množiny schopny generovat plně funkční jedince i pro instance větší velikosti, potenciálně libovolné.

5.2 Realizace metody

Autorem navržená metoda umožňuje evoluční design obecných struktur, jádrem této práce je však její použití pro návrh efektivních řadicích sítí. Pro tento účel musí dojít ke konkrétní realizaci obecných komponent metody. Níže jsou popsány detaily specifické pro užití navržené metody v oblasti návrhu řadicích sítí.

5.2.1 Uživatelské parametry

Jisté parametry metody jsou nastavitelné uživatelem. Tyto parametry jsou shrnuty níže, přičemž jsou rozděleny na parametry genetického algoritmu a parametry přepisovacího systému. Přesný význam parametrů je objasněn dále v této kapitole.

- Parametry genetického algoritmu:
  - Ukončující podmínka (počet generací, čas, nalezení plně funkčního řešení).
  - (Maximální) počet generací (G).
  - Pravděpodobnost mutace ($p_{\mathbf{mut}}$).
  - Penalizace fitness na základě parametrů kandidátního řešení.
  - Počet jedinců v populaci.
  - Počet jedinců v turnaji.
  - Zda mutovat aktivitu pravidel.
  - Zda do genotypu zahrnout axiom.
  - Velikosti řadicích sítí zahrnutých do fitness ($n_1, n_2, \ldots$).
  - Další velikosti řadicích sítí pro finální ověření růstu.
- Parametry přepisovacího systému:
  - Axiom (a).
  - Použité symboly (Σ).
  - (Maximální) počet pravidel.
  - Počet iterací přepisovacího systému (i).
  - Počet kroků přepisovacího systému na iteraci (s).

- Maximální délky:
  - Axiomu, pokud je mutován.
  - Levých kontextů pravidel.
  - Levých stran pravidel.
  - Pravých kontextů pravidel.
  - Pravých stran pravidel.

5.2.2 Genetický algoritmus

Použitá varianta genetického algoritmu je derivátem kanonického algoritmu uvedeného v kapitole 4. Genotyp jedinců je tvořen uspořádaným seznamem pravidel MDIL-systému, kde se každé pravidlo skládá ze čtyř řetězců, a to z levé a pravé strany pravidla a levého a pravého kontextu levé strany pravidla. Priorita pravidel je dána uspořádáním seznamu. Volitelně může být do genotypu zahrnut také axiom, je-li potřeba jej vyvíjet. Zmíněné řetězce se skládají z uživatelem definovaných symbolů. Musí se přitom alespoň z části jednat o symboly používané reprezentační funkcí, navíc mohou být přidány také symboly bez významu. Ty tak mohou při developmentu plnit funkci neterminálních symbolů, přestože jsou ve skutečnosti samy terminály.

Operátor selekce byl implementován turnajovým výběrem, přičemž počet jedinců v turnaji je jedním z uživatelských parametrů. Po provedení zkušebních experimentů a konzultaci s vedoucím bylo v algoritmu zcela upuštěno od operace křížení, důraz je tedy kladen výhradně na operaci mutace.

Do implementovaného genetického algoritmu byl také zahrnut elitismus. Ten spočívá v bezpodmínečném přenesení nejlepšího jedince z aktuální populace do následující populace. Navíc je tento jedinec automaticky zvolen jako rodič jednoho z jedinců v nové populaci.

Byly implementovány dvě varianty operátoru mutace. První varianta spočívá ve standardním průchodu všemi symboly všech řetězců genotypu a náhodném rozhodnutí, zda má být daný symbol mutován jednou ze tří možných mutačních funkcí (viz dále). Druhá varianta náhodně zvolí počet provedených mutací, poté náhodně vybere tento počet symbolů v genotypu a aplikuje na každý jednu ze tří mutačních funkcí. Tento přístup byl implementován jako optimalizace pro nízkou pravděpodobnost mutace, první uvedený přístup s vyšší pravděpodobností mutace však obecně dosahoval lepších výsledků. Navíc může v obou případech během mutace dojít také ke vzájemné výměně priorit dvou náhodně zvolených pravidel. Při aktivaci příslušného uživatelského parametru je dále povolena mutace aktivity pravidel – neaktivní pravidla nebudou při přepisování uvažována. Tak lze optimalizovat počet použitých pravidel MDIL-systému.

Implementované mutační funkce, které jsou aplikovatelné na symboly v řetězcích genotypu, jsou celkem tři. První z nich změní aktuální symbol na náhodně zvolený symbol z uživatelem definované množiny symbolů. Druhá z těchto funkcí odebere aktuální symbol z řetězce, čímž se řetězec stane o symbol kratším. Třetí funkce před aktuální symbol přidá nový náhodně vygenerovaný symbol z uživatelem definované množiny symbolů, tím se řetězec stane o symbol delším.

Mezi uživatelské parametry patří omezení maximálních délek řetězců genotypu, a to separátně pro řetězce reprezentující levou stranu pravidla, pravou stranu pravidla, a také levý kontext a pravý kontext levé strany pravidla. Z toho vyplývá, že může nastat situace, kdy některé z výše uvedených mutačních funkcí nebudou kvůli těmto omezením aplikovatelné. Při provádění mutací je vždy vybíráno pouze z funkcí, které jsou na daný řetězec aplikovatelné.

Ukončovací podmínka algoritmu je ovlivnitelná uživatelským nastavením. Může se jednat o nalezení plně funkčního jedince, dosažení stanoveného počtu generací, či dosažení předem stanoveného limitu na maximální čas výpočtu. Výsledný tvar implementovaného algoritmu je uveden jako algoritmus 2.

Algoritmus 2: GA použitý v navržené metodě

```
1  Nastav uživatelské parametry:
     n = počet jedinců v populaci,
     t = počet jedinců v turnaji.
2  Vytvoř počáteční populaci genotypů.
3  repeat
4      Dekóduj všechny genotypy aktuální populace na fenotypy s využitím
       developmentu a reprezentační funkce. Poté vyhodnoť fitness pro každého z n
       jedinců v aktuální populaci.
5      Vytvoř novou populaci, inicializovanou na prázdnou množinu.
6      Do nové populace přidej nejlepšího jedince z aktuální populace.
7      Do nové populace přidej mutaci nejlepšího jedince z aktuální populace.
8      repeat
9          Vyber náhodně jednoho rodiče z aktuální populace, a to turnajovým výběrem
           z t jedinců.
10         Aplikuj operátor mutace na vybraného rodiče, vzniklého jedince umísti do
           nové populace.
11     until nová populace neobsahuje n potomků
12     Nahraď aktuální populaci nově vytvořenou populací.
13 until ukončovací podmínka byla splněna
```

Jak bylo předesláno, development je v navržené metodě realizován přepisováním řetězců pomocí derivací v MDIL-systému. Jako embryo je použit prvotní řetězec, tzv. *axiom*. Axiom přitom také může být předmětem evoluce. Development probíhá následovně:

Nejdříve je proveden stanovený počet kroků MDIL-systému pro růst axoimu do jeho nové podoby před zahájením evaluace. Tato fáze přitom může být vynechána a axiom tak může zůstat nezměněn. Poté je přistoupeno k fázi evaluace.

Ve variantě metody s parametrickým developmentem je v této fázi provedena jediná iterace, a to se stanoveným počtem s kroků přepisovacího systému. Výsledný řetězec je předán k reprezentaci a ohodnocení.

Ve variantě metody s kontinuálním developmentem je proveden stanovený počet i iterací se stanoveným počtem s kroků přepisovacího systému na iteraci. Po dokončení každé iterace je aktuální řetězec předán k reprezentaci a ohodnocení.

5.2.3 Reprezentace

Volba reprezentace pro převod textového řetězce z výstupu L-systému na strukturu z prostoru fenotypu může výrazně ovlivnit jak úspěch evolučního algoritmu, tak vlastnosti nalezených struktur. Na reprezentaci obecně klademe následující dva požadavky:

1. Reprezentace musí být schopna vyjádřit libovolnou strukturu z prostoru fenotypu
2. Reprezentace musí každému možnému řetězci na výstupu přepisovacího systému přiřadit validní (myšleno ohodnotitelnou) strukturu z prostoru fenotypu.

První z požadavků zajišťuje, že metoda bude potenciálně schopna nalézt strukturu s požadovanými vlastnostmi, pokud taková struktura v prostoru fenotypu existuje. Druhý z požadavků zajišťuje, že libovolná množina pravidel přepisovacího systému bude ohodnotitelná, skrz strukturu, která vznikne reprezentací řetězce z přepisovacího systému. Reprezentace by tak měla realizovat surjektivní zobrazení řetězce na strukturu.

V rámci této práce byl použit poměrně modulární přístup k reprezentaci řadicích sítí. Jedná se o jedinou reprezentaci, ze které však lze odebráním některých skupin symbolů získat odlišnou reprezentaci. Jelikož se jednotlivé reprezentace liší používanými symboly, jsou pojmenovány výčtem použitých symbolů. Nejdříve budou objasněny principy společné pro všechny reprezentace.

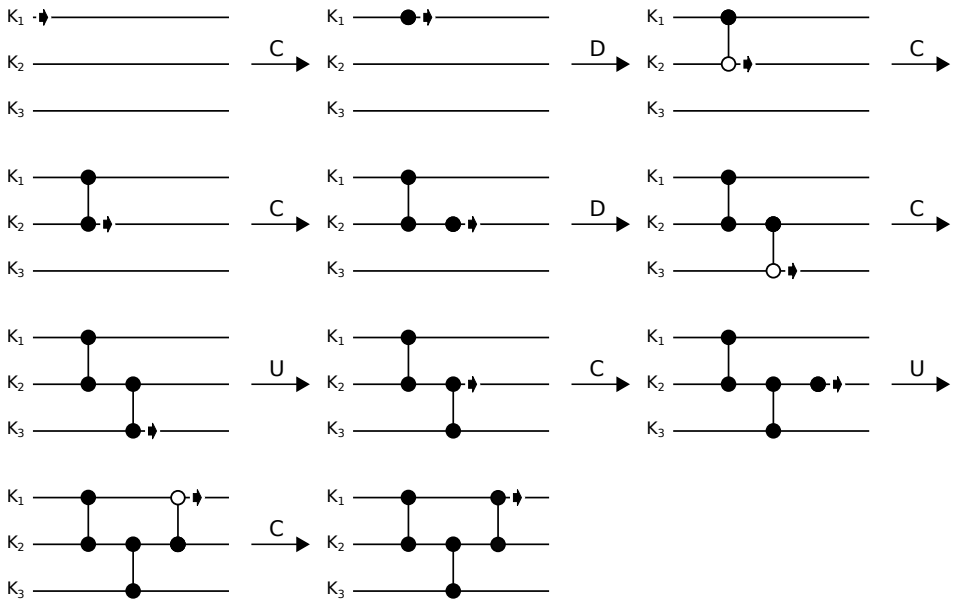
Síť pro řazení n-prvkové posloupnosti je uvažována jako n paralelních úseček, z nichž každá představuje jednu z pozic řazené posloupnosti. Úkolem reprezentace je převést textový řetězec na posloupnost propojení mezi těmito úsečkami, kde každé z propojení představuje jeden komparátor (viz obrázek 2.1 a [11]). Ve všech uvedených reprezentacích definujeme index $1 \leq i \leq n$, který označuje jednu z uvedených n úseček. Jednotlivé symboly interpretovaného řetězce jsou pak ve všech reprezentacích zpracovávány po řadě sekvenčně, přičemž tyto symboly ovlivňují hodnotu indexu i a přidávají komparátory v závislosti na hodnotě i. Výstupem reprezentace je vždy seznam dvojic úseček, kde každá dvojice představuje jeden komparátor. Celý seznam pak reprezentuje řadicí síť.

Reprezentace UDCAVI2345678

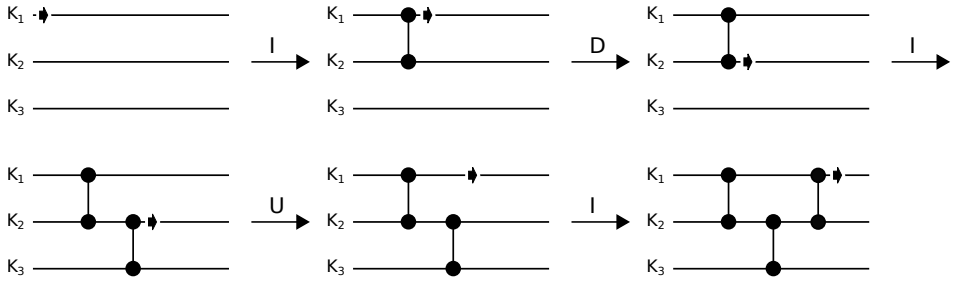
Základní koncept vytvořené reprezentace představuje výchozí reprezentace *UDCAV*. V této výchozí reprezentaci slouží symboly U a D pro pohyb indexu i vždy o jednu úsečku nahoru (Up) či dolů (Down). Symbol A posune index i na první úsečku ($i = 1$), symbol V pak na poslední ($i = n$). Symbol C slouží pro tvorbu komparátorů. Jeho první výskyt označí úsečku aktuálně na indexu i za první vstup tvořeného komparátoru. Jeho druhý výskyt označí úsečku na indexu i za druhý ze vstupů nově vzniknuvšího komparátoru, čímž je tvorba komparátoru dokončena. V případě, kdy mají být oba vstupy komparátoru nastaveny na stejnou úsečku, není při druhém symbolu C vytvořen žádný komparátor. Názorná ukázka fungování této reprezentace je uvedena na obrázku 5.3.

Tento základní koncept je dále rozšířen prostřednictvím symbolů *I2345678*. Tyto symboly vytvoří komparátor, jehož vstupy jsou připojeny na úsečky na indexech i a $(i+t) \bmod n$, kde pro symbol I je $t = 1$, pro symbol *2* je $t = 2$, atd. Obrázek 5.4 uvádí srovnání této vylepšené reprezentace s výše uvedenou reprezentací UDCAV. Je patrné, že s tímto kódováním lze jistou třídu řadicích sítí reprezentovat podstatně kratším řetězcem. To může umožnit použití jednodušších pravidel pro generování takového řetězce a snazší nalezení takových pravidel evolucí.

Ve výsledné aplikaci je implementována kompletní reprezentace UDCAVI2345678, přičemž uživatel volí množinu symbolů Σ, které mohou být použity. Uživatel tak může změnit použitou reprezentaci vynecháním určitých symbolů z Σ a ponecháním jiných. V dalším textu bude použitá reprezentace označována právě pomocí symbolů reprezentace UDCAVI2345678, které byly zahrnuty do Σ.



Obrázek 5.3: *Ukázka převodu řetězce „CDCCDCUCUC“ na řadicí síť o třech vstupech při použití reprezentace UDCAV. Umístění šipky na příslušné úsečce představuje hodnotu indexu i.*



Obrázek 5.4: *Ukázka převodu řetězce „IDIUI“ na řadicí síť o třech vstupech při použití reprezentace UDAVI. Umístění šipky na příslušné úsečce představuje hodnotu indexu i.*

5.2.4 Fitness funkce

Zásadním předpokladem pro správné fungování metody je vhodná implementace fitness funkce. Účelem této funkce je ohodnotit, do jaké míry daný jedinec splňuje očekávání na něj kladená. Pro návrh řadicích sítí by tedy měla fitness funkce zohledňovat, jak dobře je vzniklá síť schopna řadit vstupní posloupnosti. Evaluace jedince je tedy v tomto případě evaluací řadicí sítě, popsanou detailně v sekci 2.1.1.

V rámci evaluace je řadicí síť reprezentována pomocí logického obvodu. Tato reprezentace není výhodná pouze z hlediska optimalizace výpočtu, ale také z hlediska možnosti alternativní definice fitness funkce. Můžeme díky ní definovat fitness funkci nikoliv jako počet správně seřazených vstupních posloupností, ale jako počet správně spočtených bitů na výstupu obvodu. Takto definovaná fitness funkce obsahuje menší skoky a přesněji tak vystihuje míru funkčnosti hodnoceného jedince.

Penalizace fitness

Kromě funkčnosti řadicí sítě lze do fitness funkce zahrnout také další parametry řadicích sítí, které se tak stanou předmětem evoluční optimalizace. Výsledná hodnota fitness je pak dána původní hodnotou fitness sníženou o hodnotu parametrů, které je snaha minimalizovat (maximalizace parametrů nemá v případě řadicích sítí opodstatnění). Navíc, pokud je jedinec plně funkční, je jeho hodnota fitness navýšena o fixní bonus, a to z toho důvodu, aby i po odečtení všech penalizací byla hodnota fitness vyšší nebo rovna hodnotě fitness plně funkčního jedince. Produkční implementace navržené metody pro evoluční návrh řadicích sítí nabízí uživatelské parametry pro zahrnutí různých parametrů řadicí sítě do fitness funkce, a to s následujícími označeními:

- Celkové zpoždění sítě (**DT** – *Delay Total*).
- Zpoždění sítě s odstraněnými přebytečnými komparátory (**DU** – *Delay Used*).
- Celkový počet komparátorů v síti (**CT** – *Comparators Total*).
- Počet užitečných komparátorů v síti (**CU** – *Comparators Used*).
- Počet přebytečných komparátorů v síti (**CR** – *Comparators Redundant*).
- Délka řetězce reprezentujícího síť (**SL** – *String Length*).
- Počet pravidel přepisovacího systému (**RN** – *Rules Number*).
- Zakázané podřetězce v řetězcové reprezentaci řadicí sítě (**FS** – *Forbidden Substrings*).

Formálně je hodnota fitness jedince x popsána rovnicemi 5.1 až 5.3. Rovnice 5.1 definuje $\vec{p}$ jako vektor obsahující všechny výše uvedené parametry použitelné pro penalizaci fitness. Vektor $\vec{o} \in \{0,1\}^8$ vybírá složky $\vec{p}$, které budou součástí penalizace fitness. Celková hodnota penalizace fitness P je dána jako součet těchto vybraných složek vektoru $\vec{p}$. Výpočet samotné hodnoty fitness je definován rovnicí 5.3. Tento výpočet závisí na hodnotě $e(x)$, která reprezentuje sumu počtu špatně určených bitů během ověření platnosti všech řadicích sítí vzniklých z daného jedince. Výpočet se tak v případě, že je jedinec vyhodnocen jako plně funkční, liší přičtením uživatelem stanovené konstanty b (*bias*), kde $b \gg P$. Toto přičtení zajišťuje, že plně funkční jedinci budou evolucí preferováni před nefunkčními jedinci s nižší penalizací. Hodnota $w_{max}(x)$ je stanovena jako hodnota plně funkčního jedince bez penalizace, tedy jako počet testovacích vektorů násobený šířkou řadicí sítě a sumovaný pro každou ověřovanou řadicí síť vzniklou z jedince x.

$$\vec{p} = (\mathbf{DT}(x), \mathbf{DU}(x), \mathbf{CT}(x), \mathbf{CU}(x), \mathbf{CR}(x), \mathbf{SL}(x), \mathbf{RN}(x), \mathbf{FS}(x)) \tag{5.1}$$

$$P = \sum_{i=1}^{8} (\vec{o}_i \cdot \vec{p}_i) \tag{5.2}$$

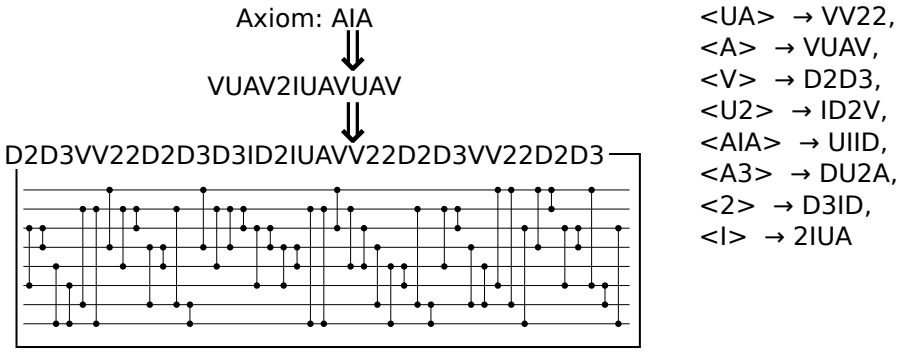
$$\text{fitness}\,(x) = \begin{cases} w_{max}(x) - e(x) - P & \mid e\,(x) \neq 0 \\ w_{max}(x) + b - P & \mid e\,(x) = 0 \end{cases} \tag{5.3}$$

Metoda může být použita jak ve verzi pro parametrický development, tak pro kontinuální development. Parametrický development lze v implementované aplikaci použít pro

návrh nerostoucích (*statických*) řadicích sítí. Kontinuální development lze použít pro návrh rostoucích řadicích sítí a také pro návrh *inkrementů* řadicích sítí. Tyto přístupy jsou objasněny v následujících sekcích.

5.3 Návrh statických řadicích sítí

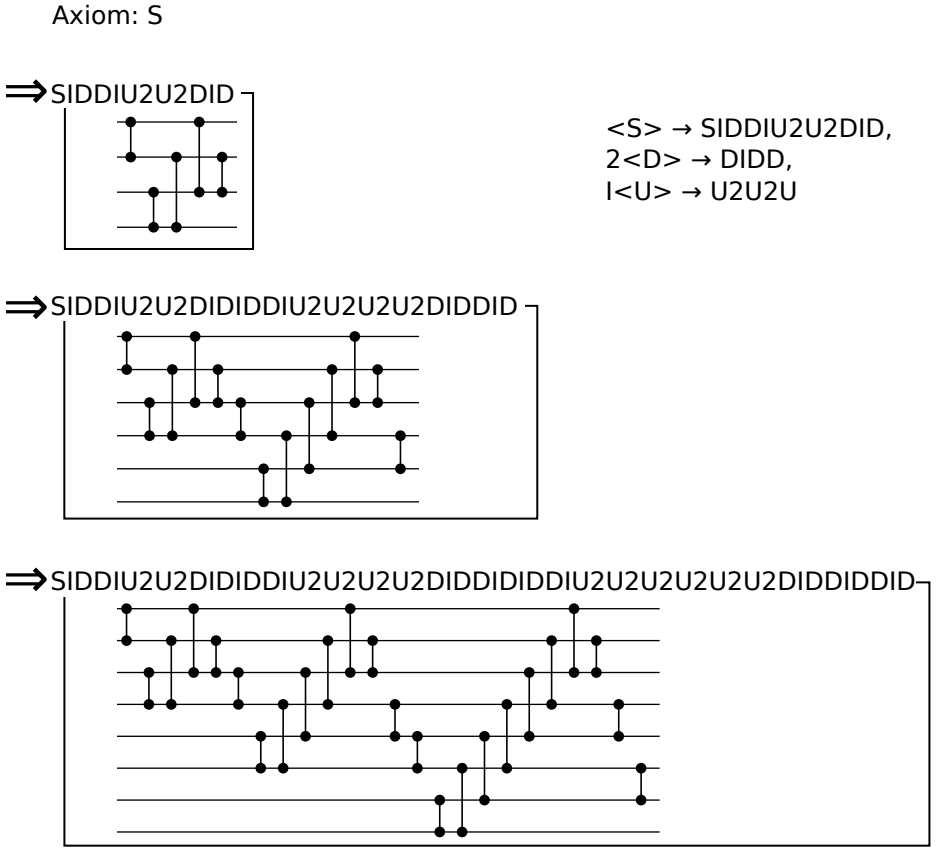
Návrh statických řadicích sítí je prvním z představovaných použití metody, a to ve verzi pro parametrický development. V tomto případě je cílem získat takovou sadu pravidel, pomocí které lze vytvořit po stanoveném počtu s kroků přepisovacího systému z pevně daného axiomu řetězec, který v použité reprezentaci odpovídá platné řadicí síti o pevně stanoveném počtu vstupů n_1. Výpočetní složitost je exponenciální vzhledem k počtu vstupů navrhované sítě, viz sekce 2.1.1. Při tomto použití metody tedy není cíleno na růst sítě, pouze na její funkčnost. Tímto použitím lze ověřit schopnost metody generovat platné řadicí sítě. Přístup znázorňuje obrázek 5.5.



Obrázek 5.5: *Ukázka užití metody pro návrh statických řadicích sítí. Axiom „AIA“ je stanoveným počtem $s = 2$ kroků přepsán MDIL-systémem s uvedenými pravidly na řetězec „D2D3VV22D2D3D3ID2IUAVV22D2D3VV22D2D3“, ten je reprezentační funkcí převeden na řadicí síť, která je následně použita pro výpočet fitness uvedené sady pravidel.*

5.4 Návrh rostoucích řadicích sítí

Dalším přínosem této práce je druhý způsob užití navržené metody. Jedná se o návrh takových pravidel přepisovacího systému, že iterativní aplikací stanoveného počtu s přepisovacích kroků v i iteracích získáme i platných sítí pro postupně rostoucí počet vstupů $n_1, n_2, \ldots, n_i$. K tomuto účelu je použita verze metody pro kontinuální development. Při výpočtu hodnoty fitness je evaluováno všech i vzniklých řadicích sítí. Jelikož však výpočetní nároky rostou exponenciálně s počtem vstupů evaluované sítě (viz sekce 2.1.1), nelze do fitness funkce zařadit příliš vysoké množství sítí. V praxi se ukázalo, že je třeba se v rámci evoluce omezit na první tři či čtyři z nich. Nižší počet nenutí evoluci nalézt dostatečně generické řešení, zatímco příliš vysoký počet kvůli vysokým výpočetním nárokům neumožní zpracovat dostatečný počet generací. Ověření, zda je vzniklý jedinec schopen dalšího růstu, je provedeno na sítích vzrostlých pro větší počty vstupů těsně před skončením algoritmu. Uvedené použití metody je ilustrováno na obrázku 5.6.

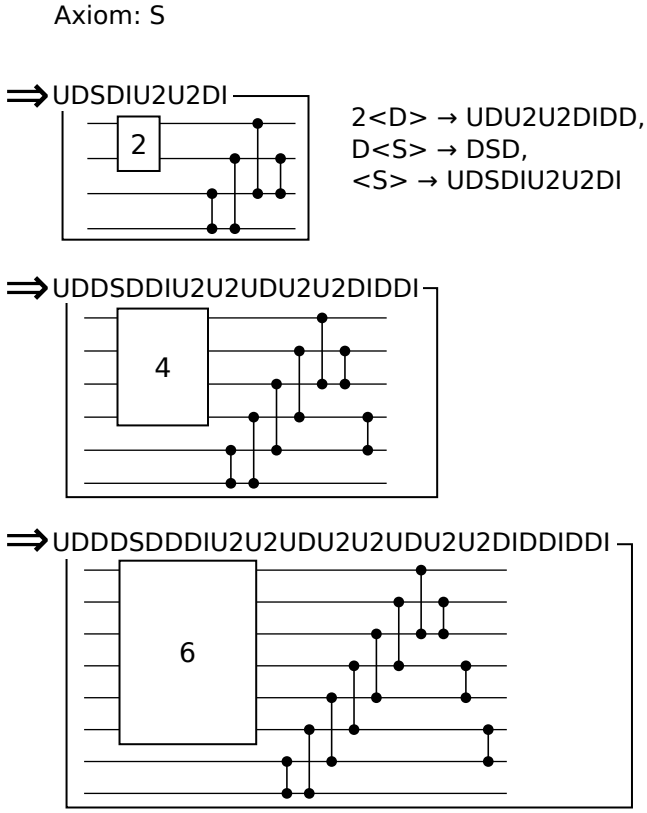


Obrázek 5.6: *Ukázka užití metody pro návrh rostoucích řadicích sítí. Obrázek ilustruje tři derivace axiomu „S“ v MDIL-systému s uvedenými pravidly. Řetězce vzniklé po každé derivaci jsou prostřednictvím reprezentační funkce převedeny na řadicí sítě. Vzniklé řadicí sítě jsou použity k výpočtu fitness daného jedince.*

5.5 Návrh inkrementů řadicích sítí

Třetím způsobem užití navržené metody je návrh *inkrementů* řadicích sítí. Inkrementem je v kontextu této práce myšlena posloupnost komparátorů, pomocí které lze již existující platnou síť o $n - k$ vstupech rozšířit na platnou řadicí síť o n vstupech. Jedná se tedy o přírůstek, či, chce-li čtenář, „přílepek“, k již existující síti, který tuto síť rozšíří o k vstupů. V rámci této práce o takovémto přírůstku budeme hovořit jako o *k-inkrementu velikosti n*, či také jako o *k-přírůstku velikosti n*. Jak znázorňuje obrázek 5.7, v tomto případě jsou navrhována pravidla přepisovacího systému, jejichž aplikací je možno konstruovat právě tyto k-inkrementy, a to postupně rostoucí po k vstupech.

Předchozí část řadicí sítě je přitom uvažována jako plně funkční black-box. Jinými slovy, prvních $n-k$ vstupů k-inkrementu velikosti n považujeme pro účely ověření platnosti řadicí sítě za seřazené. Tento přístup s sebou přináší obrovskou redukci výpočetní složitosti při konstrukci rostoucích řadicích sítí. Jestliže bylo u předešlého přístupu pro každou velikost sítě n třeba vyhodnotit 2^n různých vstupů, v tomto případě se jedná pouze o $2^k \cdot (n - k + 1)$ různých vstupů, a to právě díky tomu, že považujeme prvních $n-k$ vstupů sítě za seřazené. Pro konstantní k tedy zahrnutí dalšího k-přírůstku sítě do výpočtu hodnoty fitness způsobí pouze lineární nárůst potřebného výpočetního času pro ohodnocení tohoto přírůstku. Tato



Obrázek 5.7: *Ukázka užití metody k návrhu 2-inkrementů řadicích sítí. Obrázek ilustruje tři derivace axiomu „S“ v MDIL-systému s uvedenými pravidly. Řetězce vzniklé po každé derivaci jsou prostřednictvím reprezentační funkce převedeny na 2-inkrementy velikostí 4, 6 a 8. Vzniklé inkrementy jsou použity k výpočtu fitness daného jedince (sady pravidel). Připojení inkrementů na black-box řadicí sítě je znázorněno obdélníky umístěnými před inkrementy na prvních $n-2$ vstupech.*

úspora však přichází na úkor variability již existující části sítě. Jelikož vyvíjíme pouze přírůstek, nemůžeme již zasahovat do komparátorů uvnitř rozšiřované řadicí sítě (black-boxu). Na obrázku 5.7 jsou black-boxy znázorněny jako bílé obdélníky s číslicí odpovídající počtu jejich vstupů.

Z platné řadicí sítě o velikosti n_0 a k-inkrementů rostoucích po k vstupech od velikosti n_0+k lze sestavit platnou řadicí síť velikosti $n_0+k\cdot i$ pro $i \in \mathbb{N}$. Lze tak učinit použitím platné řadicí sítě velikosti n_0 jako základu a jejím postupným rozšiřováním pomocí rostoucích inkrementů. Jako příklad lze uvést, že z jednoho komparátoru a čtyř 2-inkrementů velikostí 4, 6, 8 a 10 tvořených metodou uvedenou na obrázku 5.7 lze sestavit řadicí síť uvedenou na obrázku 2.7.

5.6 Implementace metody

Prvotní prototyp metody byl implementován v jazyce Python 2.7. Tento prototyp byl použit pro ověření konceptu vývoje přepisovacích systémů pro řadicí sítě a pro ověření vlivu některých parametrů algoritmu. Algoritmus byl poté reimplementován do produkční im-

plementace, a to v jazyce C++11. Po implementaci bylo provedeno důkladné profilování aplikace z pohledu rychlosti a byly provedeny příslušné optimalizace. Aplikace mimo jiné používá SIMD instrukce instrukčních sad SSE2 či AVX2, pokud jsou podporovány cílovou architekturou. Toho bylo docíleno použitím direktiv OpenMP 4 a kompilátoru icpc. Pro efektivní zpracování výsledků je výstup aplikace ve formátu JSON, viz sekce A.3.

Kapitola 6

Experimentální výsledky

Bylo provedeno množství experimentů, při kterých byla navržená metoda použita pro konstrukci statických řadicích sítí, rostoucích řadicích sítí a rostoucích k-inkrementů řadicích sítí. V rámci těchto experimentů byly nejdříve evolucí znovuobjeveny některé principy uvedené v kapitole 2, po vyladění parametrů algoritmu, spuštění na superpočítači Anselm a evoluční optimalizaci výsledků pak byly nalezeny rostoucí řadicí sítě, které svými parametry výrazně předčí srovnatelné aktuálně známé principy tvorby rostoucích řadicích sítí. Vybrané experimenty a jejich výsledky jsou uvedeny níže.

6.1 Návrh statických řadicích sítí

Pro ověření konceptu navržené metody a pro ověření vlivu některých modifikací algoritmu byla metoda použita pro evoluční návrh statických řadicích sítí způsobem popsaným v sekci 5.3. Jako první sada experimentů byl zvolen návrh osmivstupých statických řadicích sítí. Pro tyto sítě je známo optimální řešení, které má 19 komparátorů v šesti paralelních skupinách, a dosahuje tedy zpoždění šest. Je tak možno porovnat získané řadicí sítě s tímto optimálním řešením. V rámci této sady experimentů bylo ověřováno, které z parametrů metody mají největší vliv na vlastnosti nalezených řadicích sítí. V prováděných experimentech byly zkoumány následující veličiny:

1. Procentuální úspěšnost evolučního algoritmu, tj. schopnost hledat platné řadicí sítě v daném limitu generací.
2. Počet použitých komparátorů v úspěšně nalezených řadicích sítích.
3. Celkový počet komparátorů v úspěšně nalezených řadicích sítích.
4. Délky řetězců reprezentujících úspěšně nalezené řadicí sítě.

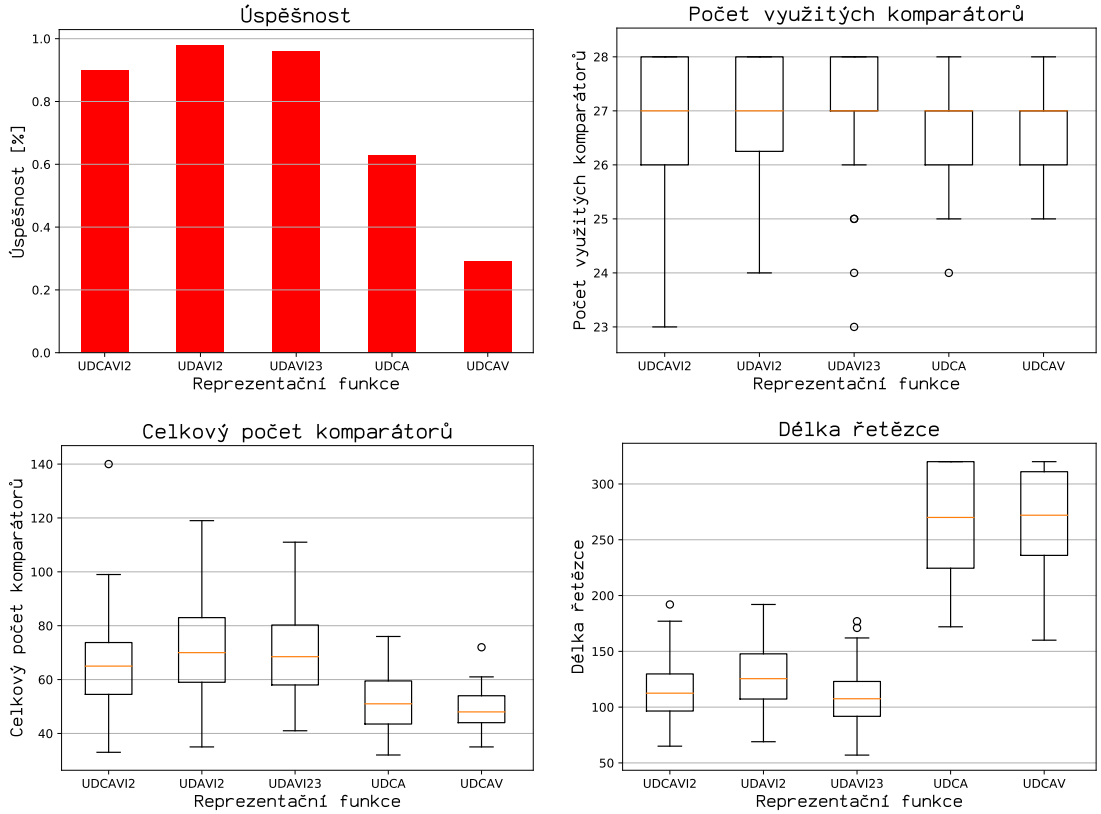
Všechny experimenty vycházely ze stejného základního nastavení, od kterého se odlišovaly vždy změnou jednoho parametru, jehož vliv byl v daném experimentu zkoumán. Pro každou zkoumanou hodnotu parametru bylo provedeno 100 běhů metody, aby měly výsledky dostatečnou statistickou významnost. Hodnoty parametrů v základním nastavení jsou uvedeny v tabulce 6.1. Tyto hodnoty byly určeny experimentálně na základě výsledků testovacích běhů aplikace. Níže je uveden rozbor vlivu tří ze zkoumaných parametrů. Jedná se o použitou reprezentační funkci, selektivní tlak ve fitness funkci a o použití symbolů, které nemají přiřazen význam v reprezentační funkci.

Parametr	Hodnota
Axiom	AIA (ACDCA, pokud reprezentace nepovoluje I)
Reprezentace	UDCAVI234BF
Typ pravidel	$0\langle 4\rangle 0 \rightarrow 4$
Počet pravidel	8
Počet jedinců v populaci	8
Počet jedinců v turnaji	4
Pravděpodobnost mutace	6,25 %
Počet generací	1000
Počet kroků MDIL-systému	4

Tabulka 6.1: *Výchozí hodnoty parametrů pro prvotní experimenty.*

6.1.1 Vliv reprezentační funkce

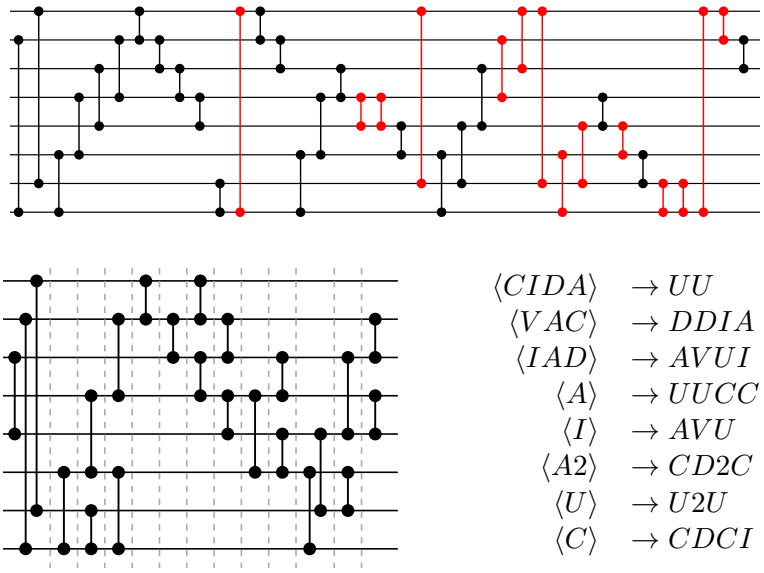
Cílem prvních experimentů bylo zjistit, která z navržených reprezentací pro převod textového řetězce z výstupu L-systému na řadicí síť je nejvhodnější. Byly provedeny experimenty se všemi navrženými reprezentacemi, vycházejícími z různých kombinací symbolů reprezentace *UDCAVI2345678*. Výsledky některých významných reprezentací jsou shrnuty v grafech na obrázku 6.1.



Obrázek 6.1: *Porovnání vybraných reprezentací. Uvedeny jsou grafy závislosti úspěšnosti algoritmu, počtu generovaných a využitých komparátorů a délky generovaného řetězce na použité reprezentační funkci.*

Z experimentu vyplývá, že nejvhodnějším kódováním je *UDCAVI2*, neboť při stejných parametrech dosahuje právě s tímto kódováním evoluční algoritmus nejvyšší úspěšnosti a zároveň ve statisticky významném počtu případů metoda tvoří řadicí sítě o výrazně nižším počtu použitých komparátorů.

Ukázka nejlepší řadicí sítě nalezené při tomto experimentu je uvedena na obrázku 6.2. Síť obsahuje celkem 37 komparátorů, z toho 23 je použitých. Zpoždění sítě je 26 před odstraněním přebytečných komparátorů a 13 po jejich odstranění.



Obrázek 6.2: *Ukázka nejlepší sítě pro $n = 8$ generované reprezentací UDCAVI2.*
Nahoře: Síť se všemi vygenerovanými komparátory, redundantní komparátory jsou označeny červeně.
Vespod vlevo: Síť s odstraněnými redundantními komparátory a rozmístěním komparátorů do příslušných paralelních hladin.
Vespod vpravo: Nalezená pravidla MDIL-systému generující řetězec reprezentující uvedenou síť.

Jeden z experimentů se dále zabýval přístupem k řešení přetečení ukazatele či komparátoru za hranice sítě. Pokud při přidávání komparátorů komparátor „přeteče“ za okraj sítě, je vhodnější jej nepřidávat. Naopak, index ukazující na příslušný řádek sítě je vhodné při přetečení umístit na opačný okraj sítě než jej nechávat na původní pozici.

6.1.2 Penalizace ve fitness funkci

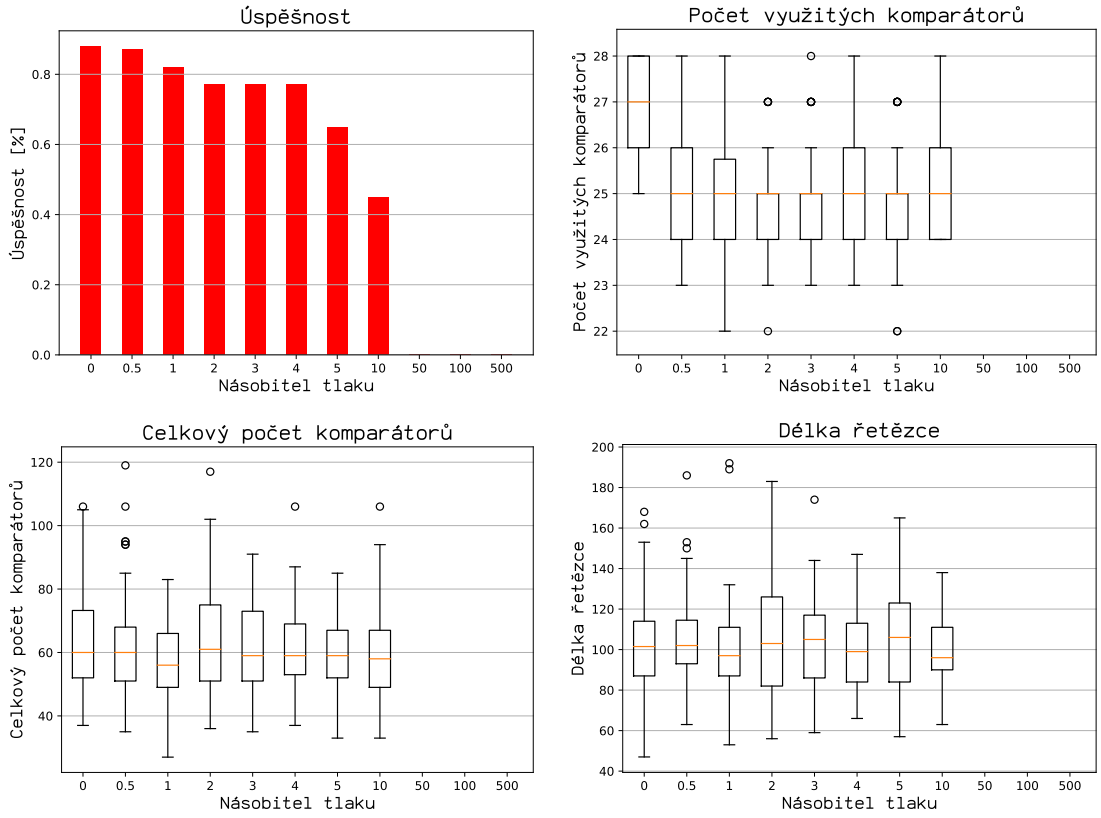
Jak je uvedeno v sekci 5.2.4, je možné upravit fitness funkci tak, aby během evoluce lépe ohodnocovala řešení s nižším počtem použitých komparátorů. Vzniklý selekční tlak způsobí významný pokles hodnoty použitých komparátorů. Pro tento experiment byl však vektor $\vec{o}$ volen tak, aby se ve fitness funkci mohly vyskytovat i vícenásobky penalizací.

Pro ilustraci je jako rovnice 6.1 uveden vzorec pro výpočet fitness v experimentu s penalizací počtu využitých komparátorů. V tomto vzorci odpovídá c zkoumanému násobiteli penalizace a $\mathbf{CU}(x)$ počtu použitých komparátorů v řadicí síti generované jedincem x. Zbylé symboly mají svůj původní význam uvedený v sekci 5.2.4.

$$\text{fitness}\,(x) = \begin{cases} w_{max}(x) - e(x) - c \cdot \mathbf{CU}(x) & \mid e\,(x) \neq 0 \\ w_{max}(x) + b - c \cdot \mathbf{CU}(x) & \mid e\,(x) = 0 \end{cases} \tag{6.1}$$

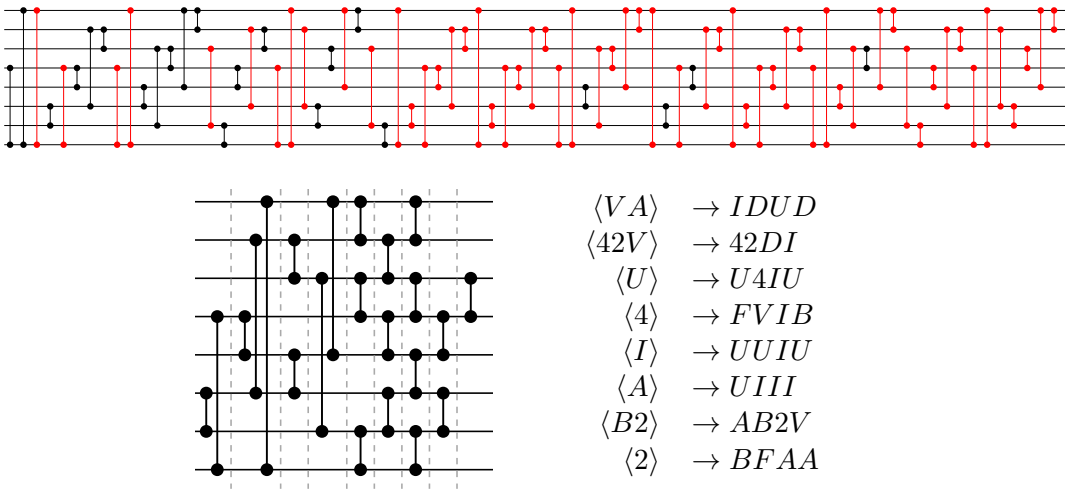
Z experimentů vyplývá, že tlak je ve fitness funkci vhodné upravit odečtením jednonásobku hodnoty dané penalizace od počtu správně spočtených bitů na výstupu evaluovaných řadicích sítí. Odečítání dvou a více násobku počtu použitých komparátorů má negativní dopad na úspěšnost genetického algoritmu a překvapivě také na počet použitých komparátorů v nalezených řešeních, což lze pozorovat na obrázku 6.3.

Z dalších výsledků vyplynulo, že penalizace celkového počtu komparátorů ovlivňuje úspěšnost metody ještě více negativně. Tlak na co nejvyšší poměr využitých komparátorů ke všem komparátorům má téměř totožný vliv jako tlak na celkový počet komparátorů. Penalizace délky výsledného řetězce způsobuje prudký pokles úspěšnosti algoritmu, což ji činí zcela nevýhodnou.



Obrázek 6.3: *Ukázka vlivu různých velikostí násobitele penalizace počtu použitých komparátorů. Uvedeny jsou grafy závislosti úspěšnosti algoritmu, počtu generovaných a využitých komparátorů a délky generovaného řetězce na použitém násobiteli selekčního tlaku na počet použitých komparátorů.*

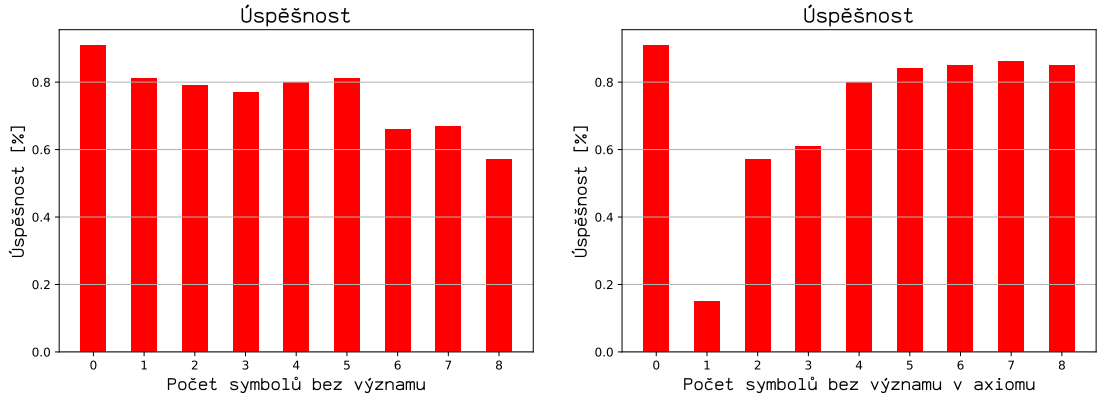
Při použití penalizace s násobitelem 1 vznikaly nejefektivnější řadicí sítě ze všech experimentů se statickými sítěmi. Jedna z těchto sítí je uvedena na obrázku 6.4. Uvedená síť původně obsahuje 79 komparátorů v 37 paralelních skupinách, po odstranění redundantních komparátorů však vznikne síť o 22 komparátorech se zpožděním 9. To je o 16 % více komparátorů než obsahuje optimální řešení, dosažené zpoždění je o tři jednotky horší než optimální řešení.



Obrázek 6.4: *Ukázka nejlepší sítě pro $n = 8$, nalezené s násobkem 1 penalizace počtu použitých komparátorů.*
Nahoře: Síť se všemi vygenerovanými komparátory, redundantní komparátory jsou označeny červeně.
Vespod vlevo: Síť s odstraněnými redundantními komparátory a rozmístěním komparátorů do příslušných paralelních hladin.
Vespod vpravo: Nalezená pravidla MDIL-systému generující řetězec reprezentující uvedenou síť.

6.1.3 Vliv symbolů bez významu

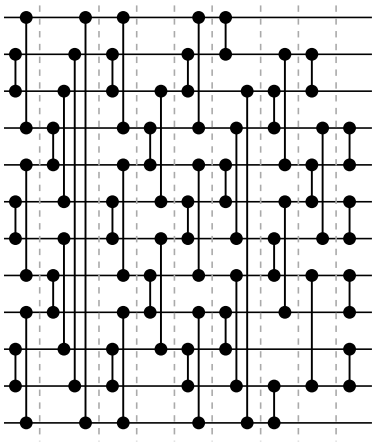
Zajímavé výsledky přinesl experiment, ve kterém bylo přepisovacímu systému umožněno přidávat do generovaného řetězce symboly, které v použité reprezentaci nemají žádný význam. S přibývajícím počtem různých těchto symbolů celková úspěšnost metody postupně klesá. Pokud je však posloupnost různých a neopakujících se symbolů bez významu zvolena jako axiom (např. „XYZW“), pak při počtu 5 a více symbolů bez významu úspěšnost metody vzroste zpět, a to téměř na původní hodnotu. Srovnání úspěšnosti metody v obou případech je uvedeno na obrázku 6.5. Použitím symbolů bez významu v axiomu namísto prvotní struktury je genetickému algoritmu dána větší volnost. Zároveň je však po něm vyžadováno, aby byl sám schopen nalézt vhodnou prvotní strukturu. Z pohledu nejnižšího počtu použitých komparátorů se jako nejvýhodnější zdá být varianta bez přidaných bezvýznamných symbolů a také varianty se dvěma a pěti bezvýznamnými symboly, kde tyto symboly tvoří axiom tak, že se v něm neopakují.



Obrázek 6.5: *Úspěšnost metody v závislosti na počtu symbolů bez významu. Pro levý graf nejsou tyto symboly použity v axiomu, pro pravý graf se axiom skládá právě ze všech symbolů bez významu.*

6.1.4 Návrh větších řadicích sítí

Po prozkoumání vlivu parametrů na osmivstupých sítích byla metoda použita pro vývoj 12 vstupých sítí. Nejlepším výsledkem byla v tomto případě síť uvedená na obrázku 6.6.



$$\begin{aligned} \langle U \rangle &\to UUUC \\ \langle C \rangle &\to DDUC \\ \langle 222 \rangle D &\to V2AV \\ \langle AVUA \rangle DI &\to AVVU \end{aligned}$$

Obrázek 6.6: *Ukázka nejlepší nalezené statické sítě pro $n = 12$. Redundantní komparátory byly odstraněny. Na pravé straně jsou uvedena pravidla MDIL-systému generující uvedenou síť, ovšem s redundantními komparátory.*

Nejlepší známá síť pro 12 vstupů obsahuje 39 komparátorů a dosahuje zpoždění 8, viz tabulka 2.1. Nalezená síť obsahuje celkem 512 komparátorů[1], použitých je však pouze 47 z nich, tj. cca 9 %. Zpoždění nalezené sítě zbavené přebytečných komparátorů je 9. Nastavení použité pro získání této sítě je uvedeno v tabulce 6.2. Lze si povšimnout, že síť vykazuje jisté znaky symetrie, zejména v prvních pěti paralelních hladinách.

[1] Z tohoto důvodu není plná síť uvedena.

Parametr	**Hodnota**
Axiom	CDUDC
Reprezentace	UDCAVI2
Typ pravidel	$0\langle 4\rangle 2 \rightarrow 4$
Počet pravidel	4
Počet jedinců v populaci	4
Počet jedinců v turnaji	4
Pravděpodobnost mutace	12,5 %
Počet generací	10000
Počet kroků MDIL-systému	7

Tabulka 6.2: *Hodnoty parametrů užité pro nalezení řadicí sítě uvedené na obrázku 6.6.*

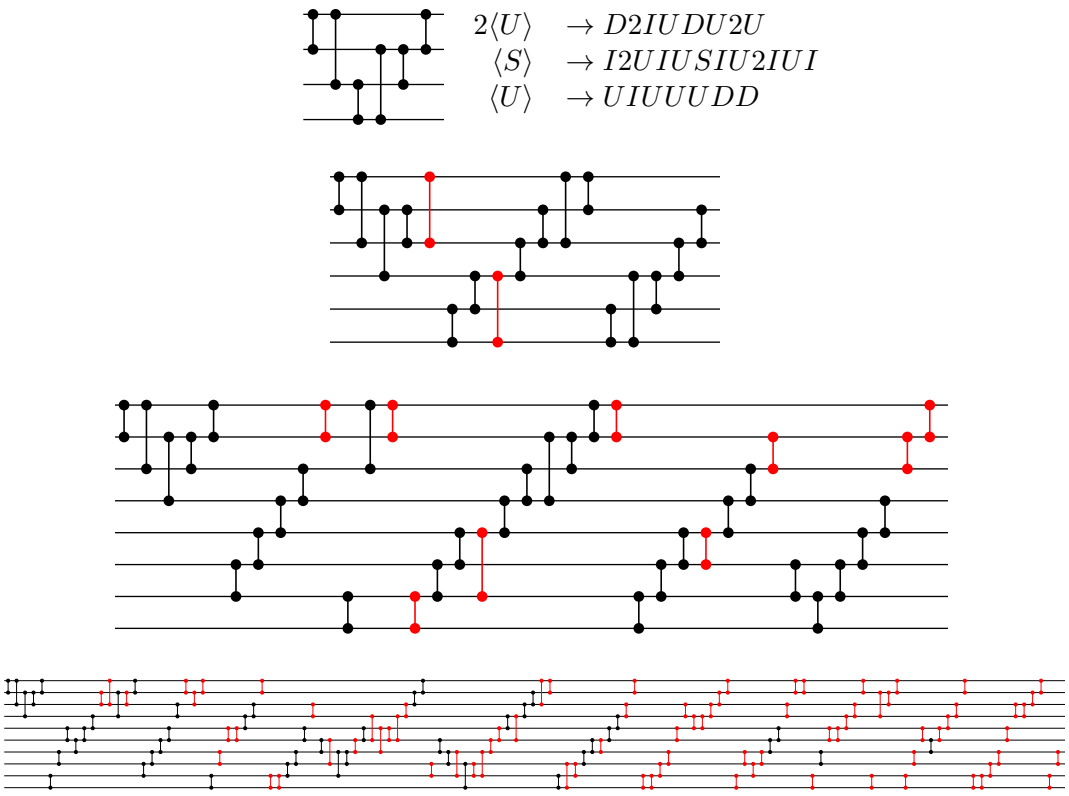
Jak je patrné z uvedených výsledků, ukázalo se, že použití metody pro návrh statických řadicích sítí není schopno přinést inovativní výsledky. Nalezené sítě obsahují extrémně vysoký podíl redundantních komparátorů a i po odstranění těchto přebytečných komparátorů nedosahují očekávaných parametrů.

Přesto však uvedené experimenty přinesly cenné poznatky o vlivu jednotlivých parametrů metody na vlastnosti generovaných sítí. S těmito poznatky bylo přikročeno k experimentům, ve kterých byla metoda použita k návrhu rostoucích řadicích sítí. O těchto experimentech pojednává následující sekce.

6.2 Návrh rostoucích řadicích sítí

V této sadě experimentů byla metoda použita k evolučnímu návrhu pravidel MDIL-systému, pomocí kterých lze konstruovat neomezeně rostoucí řadicí sítě. Experimenty byly prováděny na superpočítači Anselm, celkový počet generací byl volen tak, aby doba běhu nepřesáhla časový limit na použité výpočetní frontě. Nastavení parametrů vycházelo jednak z výše uvedených experimentů, jednak z dalšího experimentování s vývojem rostoucích sítí. Pokud není uvedeno jinak, bylo použito osm jedinců v populaci, turnajový výběr ze čtyř jedinců, a pravděpodobnost mutace 7 %. Zbylé parametry jsou individuální a jsou uvedeny pro každý experiment zvlášť. Jelikož je výsledná aplikace jednovláknová, byl pro efektivní využití výpočetní síly použit nástroj GNU Parallel. Při každém experimentu tak bylo spuštěno nejméně 96 běhů aplikace. U slibných experimentů bylo spuštěno 192 běhů.

V prvních experimentech z této sady výsledky připomínaly tvorbu řadicích sítí principem vkládání či výběru prvků. Jedna z takových sítí je zobrazena na obrázku 6.7. Pro získání této sítě bylo použito nastavení uvedené v tabulce 6.3.



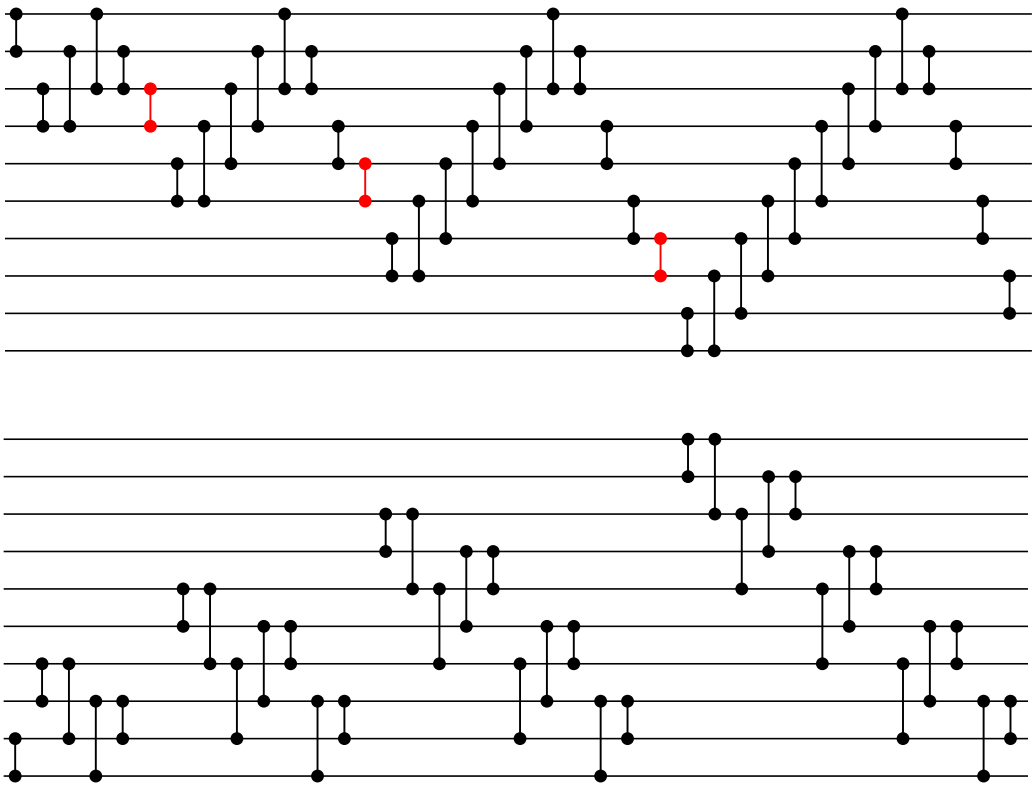
Obrázek 6.7: *Ukázka jedné z nalezených rostoucích sítí obsahujících prvky sítí tvořených metodou výběru prvků. Redundantní komparátory jsou vyznačeny červeně. Nalezená pravidla generující tuto síť jsou uvedena ve vrchní části obrázku.*

Parametr	**Hodnota**
Axiom	S
Reprezentace	SUDI2
Přetočení pozice	povoleno
Velikosti sítí pro fitness	4, 6, 8
Typ pravidel	$1\langle 1\rangle 0 \rightarrow 12$
Počet pravidel	3
Počet generací	20000000
Selektivní tlak	DT, DU, CT, CU, SL
Počet kroků MDIL-systému na iteraci	1

Tabulka 6.3: *Hodnoty parametrů užité pro nalezení rostoucí řadicí sítě uvedené na obrázku 6.7.*

V jednom z dalších experimentů evoluce znovuobjevila metodu, kterou objevil a publikoval M. Bidlo [4]. Objevená sada pravidel je bez úprav schopna generovat strukturně ekvivalentní sadu komparátorů, avšak zrcadlově převrácenou a s odlišnostmi v pořadí komparátorů v rámci paralelních skupin, což ovšem nemá vliv na celkovou strukturu sítě. V rámci dalších experimentů byly poté znovuobjeveny další sítě s identickou strukturou, zejména pokud byla pevně stanovena první vrstva sítě jako komparátory $(2 \cdot i, 2 \cdot i + 1)$ pro $i \in \langle 0, \frac{n}{2} \rangle$. Jako ukázka je zde uvedena pouze první ze znovuobjevených sítí.

Podstatný rozdíl mezi původní metodou a znovuobjevenou metodou je, že znovuobjevená metoda generuje řadicí sítě zcela bez redundantních komparátorů, na rozdíl od výsledku v [4], který obsahoval redundantní komparátory a byl dodatečně ručně vylepšen. Srovnání obou výsledků je uvedeno na obrázku 6.8. Údaje o počtu komparátorů a zpoždění generovaných sítí jsou uvedeny v tabulce 6.4. Hodnoty parametrů metody pro tento experiment jsou uvedeny v tabulce 6.5.



$$
\begin{aligned}
U\langle I\rangle &\rightarrow UUI2DDUD2U2I \\
I\langle 2\rangle &\rightarrow D \\
\langle D\rangle I &\rightarrow DDI22UUI2D2I
\end{aligned}
$$

Obrázek 6.8: *Nahoře: Metoda objevená M. Bidlem ve své původní podobě s redundantními komparátory (vyznačeny červeně).*
Níže: Znovuobjevená strukturně ekvivalentní metoda.
Vespod: Nalezená pravidla generující druhou z uvedených sítí.

n	6	8	10	12	14	16	18	20	22	24	26	28
s_n	12	22	35	51	70	92	117	145	176	210	247	287
t_n	6	9	12	15	18	21	24	27	30	33	36	39

Tabulka 6.4: *Počty komparátorů a zpoždění řadicích sítí generovaných znovuobjevenou metodou pro* $6 \leq n \leq 28$.

Parametr	**Hodnota**
Axiom	DDIUUI2D2I
Reprezentace	UDI2
Velikosti sítí pro fitness	6, 8
Typ pravidel	$1\langle 1\rangle 0 \rightarrow 12$
Počet pravidel	3
Počet jedinců v populaci	8
Počet jedinců v turnaji	4
Počet mutací	8
Počet generací	85714285
Selektivní tlak	CT, DT
Počet kroků MDIL-systému na iteraci	1

Tabulka 6.5: *Hodnoty parametrů užité pro nalezení řadicí sítě uvedené na obrázku 6.8.*

Přestože bylo dále provedeno množství experimentů s vývojem řadicích sítí rostoucích po třech a čtyřech vstupech, žádný z těchto experimentů nebyl úspěšný (tj. nepřinesl výsledek s lepšími nebo shodnými parametry, než kterých dosahují srovnatelné metody). Získané sady pravidel sice pro velikosti sítí, pro které byly vyvíjeny, generovaly rozumný počet komparátorů, pro řadicí sítě o větších velikostech však při testu růstu počet generovaných komparátorů rapidně vzrůstal na extrémní hodnoty. Ani po odstranění všech přebytečných komparátorů nepřekonaly tyto sítě doposud známé parametry, proto bylo přikročeno k experimentům s vývojem inkrementů řadicích sítí.

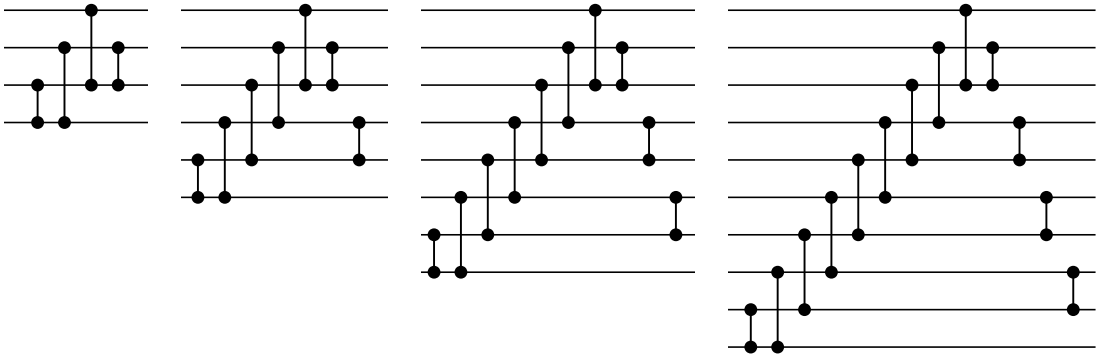
6.3 Návrh inkrementů řadicích sítí

Experimenty v oblasti návrhu k-inkrementů řadicích sítí přinesly nejlepší z výsledků této práce. Díky předpokladu, že prvních $n - k$ vstupů je již seřazeno, lze drasticky snížit časovou složitost ověření inkrementu sítě, viz sekce 5.5. Díky tomu lze rychleji spočítat fitness jedinců, a tak vyvinout za stejný čas více generací, či do fitness zahrnout sítě o větších velikostech. S tím přirozeně vzrůstá kvalita nalezených řešení.

Experimenty byly prováděny na superpočítači Anselm, a to nejdříve s návrhem sítí rostoucích po dvou vstupech, později po třech, čtyřech a více vstupech. Byl přidán parametr aplikace, který určuje ukončující podmínku nikoliv podle počtu generací, ale podle uplynulého času, a to z důvodu efektivního využití výpočetního času na superpočítači. Opět byl použit nástroj GNU Parallel. Počáteční experimenty byly spouštěny po dobu jedné hodiny v 96 až 128 běžích na expresní frontě, experimenty se slibnými výsledky byly poté spuštěny na šest až 12 hodin v 1024 až 2048 běžích na produkční frontě. Nastavení parametrů vycházelo z poznatků získaných při předchozích experimentech a bylo upravováno dle potřeby.

Pokud není uvedeno jinak, bylo použito osm jedinců v populaci, turnajový výběr ze čtyř jedinců a pravděpodobnost mutace 20 %. Počet kroků MDIL-systému na iteraci byl nastaven na jeden krok.

Při experimentech s evolučním návrhem po dvou rostoucích 2-přírůstků řadicích sítí byla opět několikrát znovuobjevena metoda nalezená M. Bidlem [4]. V tomto případě však nalezená sada pravidel generovala zcela identickou strukturu sítě jako původní metoda, a to bez redundantních komparátorů. Jeden z těchto znovuobjevených rostoucích inkrementů je uveden na obrázku 6.9. První z přírůstků rozšiřuje platnou dvouvstupou řadicí síť na platnou čtyřvstupou řadicí síť, po jednom kroku růstu rozšiřuje platnou čtyřvstupou řadicí síť na platnou šestivstupou řadicí síť, atd. Parametry experimentu, ve kterém byl tento rostoucí přírůstek nalezen, jsou uvedeny v tabulce 6.6.



$$
\begin{aligned}
2\langle D\rangle &\to UDU2U2DIDD \\
D\langle S\rangle &\to DSD \\
\langle S\rangle &\to UDSDIU2U2DI
\end{aligned}
$$

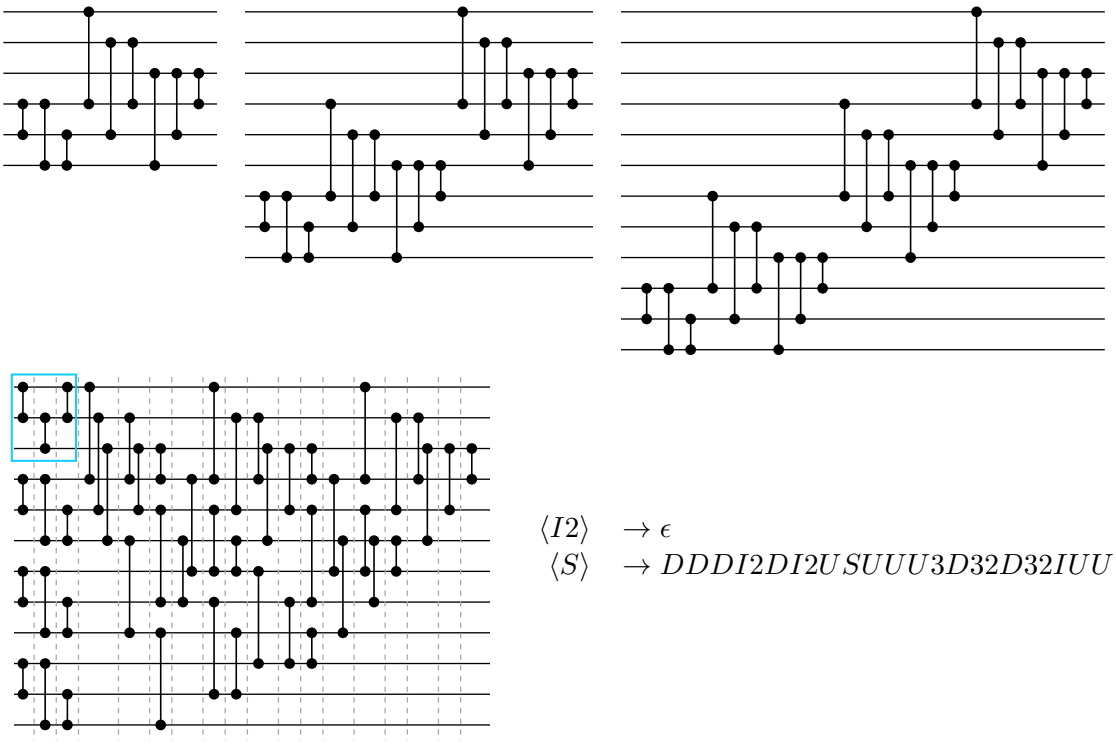
Obrázek 6.9: *Nalezený po dvou rostoucí inkrement, který generuje řadicí sítě se zcela identickou strukturou jako metoda objevená M. Bidlem, a to bez redundantních komparátorů.*

Parametr	**Hodnota**
Axiom	S
Reprezentace	SUDI2
Velikosti sítí pro fitness	4, 6, 8, 10, 12
Typ pravidel	$1\langle 1\rangle 0 \to 12$
Počet pravidel	3
Max. čas běhu	3595 s
Selektivní tlak	CT, DT
Zakázané podřetězce	„IUI“, „IDI“, „2DD2“, „2UU2“

Tabulka 6.6: *Hodnoty parametrů užité pro nalezení rostoucího inkrementu uvedeného na obrázku 6.9.*

Během experimentů však nebyl nalezen žádný po dvou rostoucí 2-přírůstek, který by dosahoval lepších výsledků než tato metoda. Bylo tedy přikročeno k návrhu inkrementů rostoucích po třech a po čtyřech vstupech.

Nejlepší nalezený 3-inkrement rostoucí po třech vstupech je uveden na obrázku 6.10. Jedná se o jeden z nejlepších výsledků této práce, neboť překonává nejlepší srovnatelnou metodu [4], a to v počtu komparátorů. Pravidla generující tento inkrement navíc nevytváří žádné redundantní komparátory. Parametry řadicích sítí vzniklých spojením optimální třívstupé řadicí sítě a takto vzrostlých inkrementů jsou uvedeny v tabulce 6.7. Inkrementy tvořené tímto postupem byly testovány pro všechny jejich aplikovatelné velikosti n v rozmezí $6 \leq n \leq 1002$ a ve všech případech byly plně funkční. Pro účely této práce lze tedy předpokládat, že metoda je obecná pro libovolné $n = 3 + 3 \cdot i$, kde $i \in \mathbb{N}$. Parametry experimentu, při kterém byla tato pravidla nalezena, jsou uvedeny v tabulce 6.8.



$\langle I2 \rangle \quad \rightarrow \epsilon$

$\langle S \rangle \quad \rightarrow DDDI2DI2USUUU3D32D32IUU$

Obrázek 6.10: *Nahoře: Nejlepší nalezený 3-inkrement rostoucí po třech vstupech. Inkrement generuje řadicí sítě zcela bez redundantních komparátorů, s nižším počtem komparátorů než nejlepší srovnatelná metoda [4].*
Vespod vlevo: Řadicí síť vytvořená spojením optimální třívstupé sítě (zarámovaná) a výše uvedených inkrementů. Komparátory jsou rozmístěny do příslušných paralelních hladin.
Vespod vpravo: Nalezená pravidla pro konstrukci uvedených rostoucích 3-inkrementů.

n	9	12	15	18	21	24	27
$*s_n$	*27*	*48*	*75*	*108*	*147*	*192*	*243*
s_n	29	51	79	113	153	199	251
$*t_n$	*12*	*18*	*24*	*30*	*36*	*42*	*48*
t_n	12	17	22	27	32	37	42

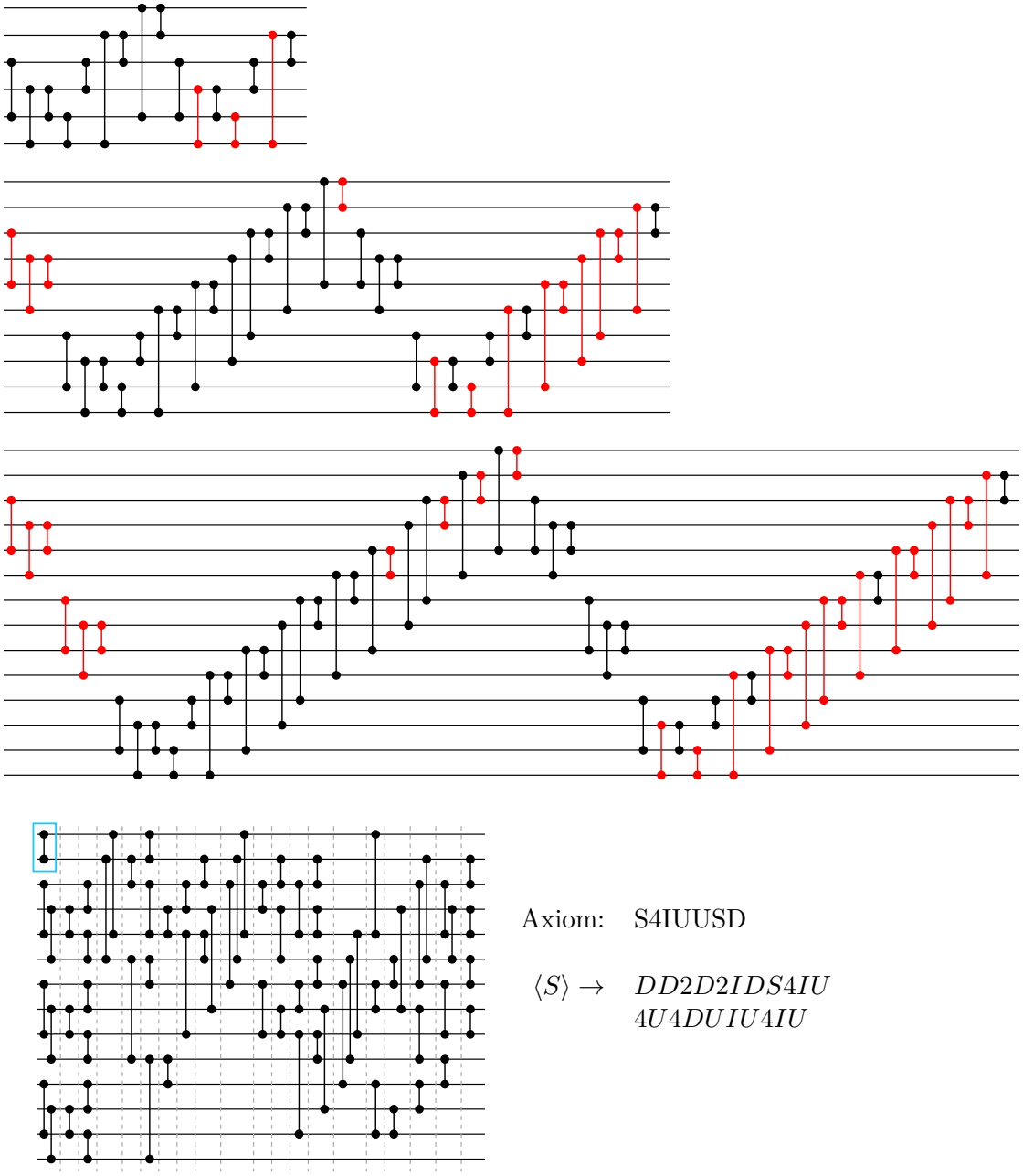
Tabulka 6.7: *Srovnání počtu komparátorů a zpoždění sítí tvořených pomocí nalezeného rostoucího 3-inkrementu ($*s_n$ a $*t_n$, řádky označené kurzívou) s nejlepší srovnatelnou metodou (s_n a t_n) [4] pro velikosti sítí $9 \leq n \leq 27$.*

Parametr	**Hodnota**
Axiom	S
Reprezentace	SUDI23
Velikosti sítí pro fitness	6, 9, 12
Typ pravidel	$2\langle 2\rangle 2 \rightarrow 24$
Max. počet pravidel	8
Počet jedinců v populaci	8
Počet jedinců v turnaji	4
Pravděpodobnost mutace	20 %
Max. čas běhu	3595 s
Selektivní tlak	CT, DT, RN
Zakázané podřetězce	„IUI“, „IDI“, „2DD2“, „2UU2“

Tabulka 6.8: *Hodnoty parametrů užité pro nalezení rostoucího inkrementu uvedeného na obrázku 6.10.*

Nalezená metoda pro tvorbu 3-inkrementů řadicích sítí dosahuje sice lepších výsledků v rámci počtu komparátorů, dosahuje však horších zpoždění než metoda uvedená v [4]. Dále bylo přikročeno k experimentům s návrhem 4-inkrementů rostoucích po čtyřech vstupech.

Evoluční návrh 4-inkrementů rostoucích po čtyřech vstupech přinesl nejlepší výsledek této práce. Jedná se o rostoucí 4-inkrement, který výrazně předčí nejlepší srovnatelnou metodu konstrukce řadicích sítí rostoucích po čtyřech vstupech [4]. Nalezená sada pravidel sice generuje jistý počet redundantních komparátorů, ty se však nalézají na předem známých pozicích, a tak je lze již během konstrukce inkrementu odstranit, podobně jako ve [4]. Nejdříve byl nalezen 4-inkrement, který obsahoval výrazně nižší počet komparátorů než uvedená metoda, dosahoval však horších parametrů zpoždění. V dalším textu bude tento nalezený inkrement označován jako *prvotní výsledek*. Tento prvotní výsledek byl následně ručně i evolučně optimalizován. Prvotní výsledek je uveden na obrázku 6.11. Parametry experimentu použité pro nalezení tohoto výsledku jsou uvedeny v tabulce 6.10. Parametry (zpoždění a počet komparátorů) prvotního výsledku jsou uvedeny v tabulce 6.9.



Obrázek 6.11: *Nahoře: Prvotní nalezený 4-inkrement rostoucí po čtyřech vstupech, který výrazně předčí nejlepší srovnatelnou metodu konstrukce rostoucích řadicích sítí [4]. Růst je ukázán na inkrementech velikostí 6, 10 a 14 vstupů. Redundantní komparátory jsou označeny červeně. Inkrement generuje řadicí sítě s výrazně nižším počtem komparátorů než nejlepší srovnatelná metoda.*
Vespod vlevo: Řadicí síť pro $n = 14$ vytvořená spojením optimální dvouvstupé sítě (zarámovaná) a výše uvedených inkrementů. Komparátory jsou rozmístěny do příslušných paralelních hladin.
Vespod vpravo: Nalezená pravidla MDIL-systému pro konstrukci uvedeného rostoucího 4-inkrementu.

n	10	14	18	22	26
s'_n	*37*	*68*	*107*	*154*	*209*
s_n	53	109	185	281	397
t'_n	*14*	*20*	*26*	*32*	*38*
t_n	22	35	49	63	77

n	10	14	18	22	26
s'_n	35	70	117	176	247
s_n	49	97	161	241	337
t'_n	12	18	24	30	36
t_n	20	30	40	50	60

Tabulka 6.9: *Vlevo: Počty komparátorů a zpoždění řadicích sítí vzniklých spojením optimální dvouvstupé řadicí sítě a nalezených rostoucích 4-inkrementů prvotního výsledku pro* $10 \leq n \leq 26$. *Řádky* s'_n *a* t'_n *označují po řadě počet komparátorů a zpoždění sítě s odstraněnými redundantními komparátory.*
Vpravo: Srovnání s [4].

Parametr	**Hodnota**
Mutace axiomu	povolena
Max. délka axiomu	16
Reprezentace	SUDI24
Velikosti sítí pro fitness	6, 10, 14
Typ pravidel	$2\langle 2\rangle 2 \rightarrow 24$
Max. počet pravidel	8
Mutace aktivity pravidel	povolena
Max. čas běhu	21590 s
Selektivní tlak	CT, DT, RN
Zakázané podřetězce	„IUI“, „IDI“, „2DD2“, „2UU2“

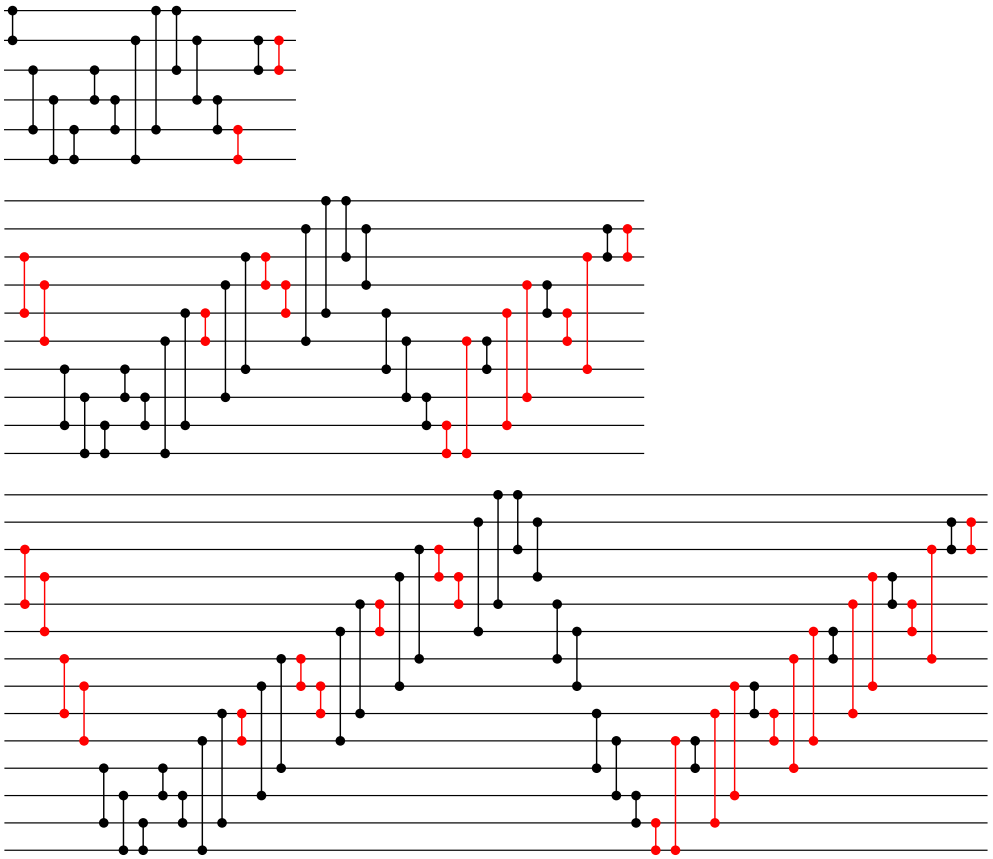
Tabulka 6.10: *Hodnoty parametrů užité pro nalezení prvotního rostoucího inkrementu uvedeného na obrázku 6.11.*

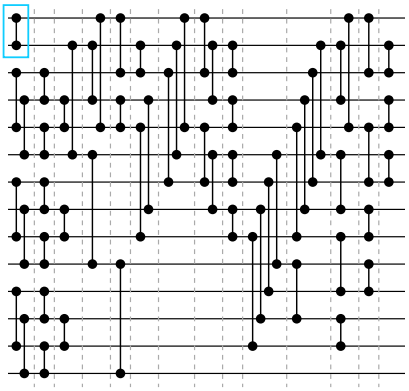
Uvedený prvotní nalezený inkrement byl dále podroben ruční analýze a pokusům o optimalizaci. Jelikož tyto pokusy neuspěly, bylo přikročeno k optimalizování nalezené sady pravidel pomocí genetického algoritmu.

Uvedená sada pravidel (o jednom pravidlu a axiomu) byla použita jako výchozí bod pro prohledávání stavového prostoru (všichni jedinci v populaci byli inicializováni touto sadou). Tato optimalizace přinesla ještě lepší výsledek, než kterým je prvotní nalezený 4-inkrement. Tento výsledek po odstranění redundantních komparátorů výrazně překonává nejlepší srovnatelnou metodu [4], a to jak v počtu generovaných komparátorů, tak ve zpoždění.

Výsledek je uveden na obrázku 6.12. Srovnání jeho parametrů s parametry nejlepší srovnatelné metody je uvedeno v tabulce 6.12. Parametry experimentu, při kterém byl tento výsledek získán, jsou uvedeny v tabulce 6.11.

Nalezený 4-inkrement na počátku rozšiřuje dvouvstupou řadicí síť na šestivstupou, po prvním kroku růstu rozšiřuje šestivstupou síť na desetivstupou, atd. Jelikož jsou pozice redundantních komparátorů plně předvídatelné, lze z nalezených pravidel extrahovat obecný princip tvorby 4-inkrementu zcela bez redundantních komparátorů pro libovolnou velikost sítě. Tento princip je uveden v podobě algoritmu, uvedeného jako algoritmus 3. Inkrementy tvořené tímto algoritmem byly testovány pro všechny jejich aplikovatelné velikosti n v rozmezí $6 \leq n \leq 2002$ a ve všech případech byly plně funkční. Pro účely této práce lze tedy předpokládat, že metoda je obecná pro libovolné $n = 2 + 4 \cdot i$, kde $i \in \mathbb{N}$.





Axiom: S42UUUUD2DSII

$\langle S \rangle \rightarrow \quad DD2D2DS4IU4U4IDUDIUUUD4U$

Obrázek 6.12: *Nahoře: Evolučně optimalizovaný 4-inkrement rostoucí po čtyřech vstupech, který výrazně předčí nejlepší srovnatelnou metodu konstrukce rostoucích řadicích sítí [4]. Růst je ukázán na inkrementech velikostí 6, 10 a 14 vstupů. Redundantní komparátory jsou označeny červeně. Inkrement generuje řadicí sítě s výrazně nižším počtem komparátorů a nižším zpožděním než nejlepší srovnatelná metoda.*
Vespod vlevo: Řadicí síť pro $n = 14$ vytvořená spojením optimální dvouvstupé sítě (zarámovaná) a výše uvedených inkrementů. Komparátory jsou rozmístěny do příslušných paralelních hladin.
Vespod vpravo: Nalezená pravidla MDIL-systému pro konstrukci uvedeného rostoucího 4-inkrementu.

Algoritmus 3: Konstrukce 4-inkrementu bez redundantních komparátorů

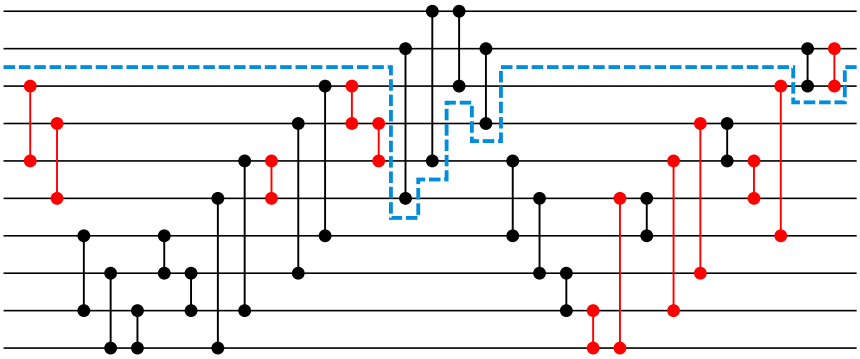
Input: $n \geq 14$ ... velikost 4-inkrementu

```
1   addComp(n − 4, n − 2)
2   addComp(n − 3, n − 1)
3   addComp(n − 2, n − 1)
4   addComp(n − 4, n − 3)
5   addComp(n − 3, n − 2)
6   for (i = n − 5; i >= 0; i = i − 1) do
7       addComp(i,i+4)
8   end for
9   for (i = 0; i <= n − 3; i = i + 4) do
10      addComp(i,i + 2)
11      addComp(i + 1,i + 3)
12  end for
13  for (i = n − 3; i >= 0; i = i − 2) do
14      addComp(i,i + 1)
15  end for
```

Odstraněním horních dvou vstupů inkrementů i s jim příslušejícími komparátory lze získat validní 4-inkrementy o velikostech 8, 12, 16, atd. Tuto možnost znázorňuje obrázek 6.13. V algoritmu 3 lze tuto úpravu realizovat záměnou číslice 0 za číslici 2 na řádku 9. Algoritmus je pak platný pro $n \geq 8$. Platnost inkrementů vzniklých tímto upraveným algoritmem byla testována pro všechna aplikovatelná n pro $8 \leq n \leq 2000$, přičemž každý z testovaných inkrementů byl vyhodnocen jako platný. Upravený algoritmus tak lze pro účely této práce považovat za obecný pro libovolné $n = 4 + 4 \cdot i$, kde $i \in \mathbb{N}$. Parametry dosažené řadicími sítěmi konstruovanými oběma verzemi algoritmu 3 ve srovnání s nejlepší srovnatelnou metodou [4] jsou uvedeny v tabulce 6.12.

Parametr	**Hodnota**
Mutace axiomu	povolena
Max. délka axiomu	26
Reprezentace	SUDI24
Velikosti sítí pro fitness	6, 10, 14
Typ pravidel	$8\langle 8\rangle 8 \rightarrow 48$
Max. počet pravidel	8
Mutace aktivity pravidel	povolena
Max. čas běhu	43180 s
Selektivní tlak	CT, DT, RN
Zakázané podřetězce	„IUI“, „IDI“, „2DD2“, „2UU2“

Tabulka 6.11: *Hodnoty parametrů užité pro nalezení rostoucího inkrementu uvedeného na obrázku 6.11.*



Obrázek 6.13: *Znázornění odstranění prvních dvou vstupů a vytvoření 4-inkrementu velikosti $n = 8$, tedy inkrementu, který čtyřvstupou síť rozšiřuje na osmivstupou.*

n	8	10	12	14	16	18	20	22	24	26	28
$*s_n$	*19*	*31*	*41*	*58*	*71*	*93*	*109*	*136*	*155*	*187*	*209*
s_n	22	35	51	70	92	117	145	176	210	247	287
$*t_n$	*6*	*9*	*10*	*14*	*15*	*18*	*19*	*23*	*24*	*27*	*28*
t_n	9	12	15	18	21	24	27	30	33	36	39

Tabulka 6.12: *Srovnání objevené metody konstrukce rostoucích řadicích sítí s nejlepší srovnatelnou metodou. Objevená metoda nejlepší metodu předčí jak v kritériu počtu komparátorů, tak v kritériu zpoždění vzniklých řadicích sítí. Počty komparátorů $*s_n$ a zpoždění $*t_n$ odpovídají řadicím sítím vzniklým spojením optimální dvou či čtyřvstupé řadicí sítě a příslušných rostoucích inkrementů generovaných algoritmem 3 či jeho výše uvedenou úpravou. Uvedeny jsou hodnoty pro velikosti řadicích sítí $8 \leq n \leq 28$. Řádky označené hvězdičkou a vysázené kurzívou odpovídají výsledkům této práce, zbylé řádky odpovídají nejlepší aktuálně známé metodě uvedené v [4].*

Kapitola 7

Závěr

V rámci této práce byla popsána teorie řadicích sítí, byly uvedeny aktuálně známé meze optimálních řadicích sítí a popsány některé známé metody pro obecnou konstrukci řadicích sítí větších velikostí. Dále byly diskutovány problémy testování platnosti řadicích sítí z pohledu jeho výpočetní složitosti a byly představeny příslušné použitelné optimalizace tohoto procesu.

Rovněž byla popsána problematika evolučních algoritmů. Nejdříve byl představen obecný genetický algoritmus a následně detailně popsány jeho části. Byla objasněna problematika selekce, křížení i mutace. V závěru kapitoly o evolučních algoritmech byly přiblíženy principy biologií inspirovaného vývinu.

S ohledem na biologií inspirovaný vývin byla také přiblížena problematika přepisovacích systémů, zejména L-systémů. Bylo navrženo, popsáno a implementováno rozšíření třídy IL-systémů, nazvané MDIL-systémy, jejichž použití se dále v evolučním návrhu řadicích sítí ukázalo být vhodné.

V další kapitole byla představena navržená metoda pro evoluční návrh obecných struktur. Tato metoda je založena na biologií inspirovaném vývinu realizovaném prostřednictvím výše uvedeného MDIL-systému. Dále byla metoda konkretizována pro užití v evolučním návrhu řadicích sítí. Detailně byla představena navržená reprezentace řadicích sítí pro použití v genetickém algoritmu, jakožto i zvolená fitness funkce. Následně byly představeny tři možnosti užití konkretizované metody pro návrh statických (tj. nerostoucích) i rostoucích řadicích sítí, jakožto i tzv. *k-inkrementů* řadicích sítí.

Navržená metoda byla následně úspěšně uplatněna v oblasti návrhu řadicích sítí, jak statických, tak rostoucích, a v oblasti návrhu rostoucích inkrementů řadicích sítí. V poslední zmíněné oblasti pak metoda zaznamenala významný úspěch, neboť výrazně překonala výsledky nejlepší známé srovnatelné metody uvedené v [4], a to v návrhu inkrementů pro řadicí sítě rostoucí po třech a po čtyřech vstupech. Nově objevená metoda tak například pro velikost řadicí sítě $n = 28$ generuje síť obsahující o 27 % méně komparátorů a dosahující o 28 % nižšího zpoždění oproti nejlepší srovnatelné metodě [4].

V rámci další práce by mohlo být zkoumáno uplatnění představené metody také v dalších oblastech mimo návrh řadicích sítí. V oblasti návrhu řadicích sítí by pak metoda mohla být rozšířena například návrhem nové reprezentace řadicích sítí či novým přístupem k růstu sítí v rámci výpočetního developmentu.

Literatura

[1] Back, T.: Selective pressure in evolutionary algorithms: a characterization of selection mechanisms. In *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, Jun 1994, s. 57–62 vol.1, doi:10.1109/ICEC.1994.350042.

[2] Batcher, K. E.: Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, ACM, 1968, s. 307–314.

[3] Bezem, M.; Klop, J.; de Vrijer, R.; aj.: *Term Rewriting Systems*. Cambridge Tracts in Theoretica, Cambridge University Press, 2003, ISBN 9780521391153.

[4] Bidlo, M.: *Evolutionary Design of Generic Structures Using Instruction-Based Development*. Faculty of Information Technology BUT, 2010, ISBN 978-80-214-4210-8, 124 s.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=9459

[5] Brabazon, A.; O'Neill, M.; McGarraghy, S.: *Natural Computing Algorithms*. Springer Publishing Company, Incorporated, první vydání, 2015, ISBN 3662436302, 9783662436301.

[6] Bundala, D.; Zavodny, J.: Optimal Sorting Networks. *CoRR*, ročník abs/1310.6271, 2013.
URL http://arxiv.org/abs/1310.6271

[7] Codish, M.; Cruz-Filipe, L.; Ehlers, T.; aj.: Sorting Networks: to the End and Back Again. *CoRR*, ročník abs/1507.01428, 2015.

[8] Ehlers, T.: Merging almost sorted sequences yields a 24-sorter. *Information Processing Letters*, ročník 118, 2017: s. 17 – 20, ISSN 0020-0190, doi:http://doi.org/10.1016/j.ipl.2016.08.005.
URL http://www.sciencedirect.com/science/article/pii/S0020019016301193

[9] Holland, J.: *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. A Bradford book, M.I.T.P., 1992, ISBN 9780262581110.

[10] Juillé, H.: Evolution of Non-Deterministic Incremental Algorithms As a New Approach for Search in State Spaces. In *Proceedings of the 6th International Conference on Genetic Algorithms*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, ISBN 1-55860-370-0, s. 351–358.

[11] Knuth, D. E.: *The Art of Computer Programming, Volume 3: (2Nd Ed.) Sorting and Searching*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1998, ISBN 0-201-89685-0.

[12] Koza, J. R.: *Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems*. Stanford University, Department of Computer Science, 1990.

[13] Kumar, S.; Bentley, P.: *On Growth, Form and Computers*. Elsevier Academic Press, 2003, ISBN 9780124287655.

[14] Lindenmayer, A.: Mathematical models for cellular interaction in development: Parts I and II. *Journal of Theoretical Biology*, ročník 18, 1968: s. 280–315.

[15] Miller, J. F.; Thomson, P.: Cartesian genetic programming. In *European Conference on Genetic Programming*, Springer, 2000, s. 121–132.

[16] Parberry, I.: A Computer Assisted Optimal Depth Lower Bound for Sorting Networks with Nine Inputs. In *Proceedings of the 1989 ACM/IEEE Conference on Supercomputing*, Supercomputing '89, New York, NY, USA: ACM, 1989, ISBN 0-89791-341-8, s. 152–161, doi:10.1145/76263.76280.
URL http://doi.acm.org/10.1145/76263.76280

[17] Paterson, M. S.: Improved sorting networks withO(logN) depth. *Algorithmica*, ročník 5, č. 1, 1990: s. 75–92, ISSN 1432-0541, doi:10.1007/BF01840378.
URL http://dx.doi.org/10.1007/BF01840378

[18] Prusinkiewicz, P.; Lindenmayer, A.: *The Algorithmic Beauty of Plants*. New York, NY, USA: Springer-Verlag New York, Inc., 1996, ISBN 0-387-94676-4.

[19] Rozenberg, G.; Salomaa, A.: *The Mathematical Theory of L Systems*. Pure and Applied Mathematics, Elsevier Science, 1980, ISBN 9780080874067.

[20] Rozenberg, G.; Salomaa, A.: *The Mathematical Theory of L Systems*. Pure and Applied Mathematics, Elsevier Science, 1980, ISBN 9780080874067.

[21] Sekanina, L.; Bidlo, M.: Evolutionary Design of Arbitrarily Large Sorting Networks Using Development. *Genetic Programming and Evolvable Machines*, ročník 6, č. 3, 2005: s. 319–347, ISSN 1573-7632, doi:10.1007/s10710-005-2987-8.
URL http://dx.doi.org/10.1007/s10710-005-2987-8

[22] Trefzer, M.; Tyrrell, A.: *Evolvable Hardware: From Practice to Application*. Natural Computing Series, Springer Berlin Heidelberg, 2015, ISBN 9783662446164.

[23] Valsalam, V. K.; Miikkulainen, R.: Using Symmetry and Evolutionary Search to Minimize Sorting Networks. *J. Mach. Learn. Res.*, ročník 14, č. 1, Únor 2013: s. 303–331, ISSN 1532-4435.
URL http://dl.acm.org/citation.cfm?id=2567709.2502591

Přílohy

Příloha A

Manuál

Obsah přiloženého CD je rozdělen do následujících adresářů:

- **results** Obsahuje výsledky experimentů s návrhem rostoucích řadicích sítí a s návrhem inkrementů řadicích sítí.
- **src** Obsahuje zdrojové soubory implementované aplikace a jejího prototypu.
- **thesis** Obsahuje digitální podobu tohoto textu.
- **thesis-src** Obsahuje zdrojové soubory tohoto textu.

Produkční aplikace se sestává z následujících zdrojových souborů:

- **sortnet/Sortnet.{h,c}** implementuje řadicí síť, včetně metod pro její evaluaci.
- **rewsys/Mdil.{h,c}** implementuje přepisovací MDIL-systém.
- **Genome.{h,c}** reprezentuje jedince populace.
- **main.c** obsahuje hlavní smyčku evolučního cyklu.
- **settings.h** obsahuje uživatelské nastavení aplikace.
- **constants.h** definuje globální konstanty.
- **scrap.h** byl použit k testování inkrementů vzniklých aplikací algoritmu 3 ze sekce 6.3.

K aplikaci byla vytvořena dokumentace ve formátu doxygen. Tato dokumentace se nachází v adresáři `src/production/doc-doxygen/`.

A.1 Nastavení parametrů aplikace

Produkční implementace představené metody pro evoluční návrh řadicích sítí se nachází v adresáři `src/production/`. Uživatelské parametry aplikace je možno nastavit v souboru `settings.h`, a to definicí příslušných maker pomocí direktiv `#define`. Tento způsob umožňuje provést během překladu optimalizace specifické pro dané nastavení. Obdobné parametry lze nalézt také u prototypu, a to v souboru `src/prototype/settings.py`. Přiložený soubor `settings.h` obsahuje nastavení určené pro ukázku aplikace, které limituje dobu běhu na 1 minutu. Význam jednotlivých nastavitelných maker je uveden níže v tabulce A.1.

Název makra	Význam
`DEBUG`	Pokud je makro definováno, jsou vypisovány ladicí údaje o změnách fitness během výpočtu.
`MAX_TIME`	Určuje maximální čas v sekundách, po který má evoluční cyklus běžet.
`SN_WIDTH`	Velikosti řadicích sítí $(n_1, \ldots n_i)$, které budou použity pro výpočet hodnoty fitness. Délka tohoto pole určuje počet iterací i.
`EVOLVE_K_INCR_STEP`	Pokud je definováno toto makro, dochází k vývoji k-inkrementů, tj. při výpočtu fitness sítě velikosti n_x je prvních $n_x - k$ vstupů považováno za seřazené. Hodnota k je udána hodnotou tohoto makra.
`SN_WIDTH_GROWTH`	Velikosti řadicích sítí $(n_{i+1}, \ldots n_j)$, které budou po skončení evoluce použity pro ověření, zda výsledek generuje platné řadicí sítě o větších velikostech, než byly použity v rámci evoluce.
`RESET_NONGROWING`	Pokud je toto makro nastaveno na hodnotu `True`, dojde k reinicializaci jedinců, kteří dosáhnou plné funkcionality při vytváření sítí velikostí `SN_WIDTH`, ale selžou při vytváření některé ze sítí velikostí `SN_WIDTH_GROWTH`.
`AXIOM`	Definuje řetězec, který bude použit jako axiom.
`AXIOM_LEN`	Pokud je toto makro definováno, je do chromosomu jedinců zařazen také axiom, který tak bude evolučně vyvíjen. Hodnota tohoto makra pak omezuje jeho maximální délku. Pokud je zároveň definováno makro `AXIOM`, je na počátku evoluce axiom všech jedinců inicializován na hodnotu makra `AXIOM`.
`TERMINALS`	Definuje symboly (abecedu Σ), které mohou být použity v přepisovacích pravidlech, a tedy i v generovaných řetězcích. Omezením množiny použitých symbolů lze měnit použitou reprezentační funkci, jak je popsáno v sekci 5.2.3
`LEFTCTX_LEN`	Definuje maximální délku levého kontextu pravidla přepisovacího systému.
`LEFT_LEN`	Definuje maximální délku levé strany pravidla přepisovacího systému.
`RIGHTCTX_LEN`	Definuje maximální délku pravého kontextu pravidla přepisovacího systému.
`RIGHT_LEN`	Definuje maximální délku pravé strany pravidla přepisovacího systému.
`RULES`	Počet pravidel přepisovacího systému. V případě, že `MUT_RULE_ACTIVITY` je nastaveno na `True`, definuje `RULES` maximální počet pravidel.

Název makra	**Význam** *Pokračováno z předchozí strany...*
`MUT_RULE_ACTIVITY`	Pokud je nastaveno na `True`, povoluje mutaci příznaku jednotlivých pravidel, který určuje, zda je pravidlo aktivní. Neaktivní pravidla nejsou použita pro přepis a nejsou uvažována jako součást přepisovacího systému, mohou však být mutována a stát se znovu aktivními. Lze tak například optimalizovat počet použitých pravidel.
`PRELOAD_RULES`	Pokud je definováno toto makro, je jeho hodnota použita jako řetězec popisující pravidla přepisovacího systému, která mají být použita pro inicializaci jedinců na počátku evoluce. Pokud je zároveň definováno makro `AXIOM_LEN`, musí tento řetězec obsahovat na počátku axiom následovaný dvojtečkou, v opačném případě axiom ani dvojtečku obsahovat nesmí. Tvar řetězce je následující: `[axiom:] Lctx1 < Lhs1 > Rctx1 -> Rhs1, Lctx2 < ...`
`POPSIZE`	Definuje počet jedinců v populaci.
`TOUR`	Definuje počet jedinců v turnaji.
`MSEQ`	Specifikuje použitou variantu operátoru mutace, jak je popsáno v sekci 5.2.2. Pokud je nastaveno na `True`, je použit tradiční přístup k mutaci a pravděpodobnost mutace ($p_{\mathbf{mut}}$) je dána hodnotou makra `MUT_PROB`. Pokud je `MSEQ` nastaveno na `False`, je použita varianta operátoru mutace optimalizovaná pro nízké hodnoty $p_{\mathbf{mut}}$, přičemž počet provedených mutací je určen hodnotou makra `MGENES`.
`MUT_PROB`	Pravděpodobnost mutace ($p_{\mathbf{mut}}$), $0 < p_{\mathbf{mut}} < 1$.
`MUT_PROB_RAMP`	Pokud je toto makro definováno, mění v průběhu generací hodnotu `MUT_PROB` na základě rampové funkce.
`MGENES`	Počet genů, které budou mutovány, pokud není použito `MSEQ`.
`MUT_ADAPT_K`	Definice tohoto makra povolí adaptaci pravděpodobnosti mutace na základě úspěšnosti algoritmu v nalézání úspěšnějších řešení. V praxi tento přístup dosahoval horších výsledků než pevně stanovená hodnota `MUT_PROB`. Hodnota tohoto makra definuje délku historie, po jakou je sledována úspěšnost algoritmu, v počtu generací.
`MUT_ADAPT_G`	Konstanta g pro adaptivní změnu pravděpodobnosti mutace. Musí být definována, pokud je definována `MUT_ADAPT_K`.
`CROSS`	Pokud je nastaveno na `True`, je prováděno experimentální křížení jedinců. V praxi přinášelo křížení horší výsledky než algoritmus bez křížení a nebylo dále uvažováno.
`GENERATIONS`	Definuje maximální počet generací, po který má evoluční cyklus běžet.

Název makra	**Význam**	*Pokračováno z předchozí strany...*
`DEVEL_STEPS_INITIAL`	Definuje počet kroků přepisovacího systému, které budou provedeny v rámci developmentu před vstupem do první iterace.	
`DEVEL_STEPS_DELTA`	Definuje počet kroků k přepisovacího systému, které budou provedeny v každé iteraci.	
`OPTIMIZE_FOUND`	Pokud je nastaveno na `False`, běh evolučního cyklu bude ukončen ihned po nalezení plně funkčního jedince. Pokud je nastaveno na `True`, běh evolučního cyklu bude pokračovat i po nalezení plně funkčního jedince. Zpravidla tak dojde k optimalizaci tohoto funkčního jedince podle parametrů penalizace fitness funkce.	
`OPTIMIZE`	Specifikuje, které vlastnosti kandidátních řešení budou zahrnuty do penalizace fitness. Formálně specifikuje prvních sedm prvků vektoru $\vec{\sigma}$ ze sekce 5.2.4. Hodnotu tohoto makra volí uživatel jako bitové `OR` některých z maker `OPT_DELAYTOTAL` (**DT**), `OPT_DELAYUSED` (**DU**), `OPT_COMPSTOTAL` (**CT**), `OPT_COMPSUSED` (**CU**), `OPT_UNUSEDCOMPS` (**CR**), `OPT_STRLEN` (**SL**) a `OPT_RULES` (**RN**). `OPT_RULES` přitom může být použito pouze pokud je zároveň `MUT_RULE_ACTIVITY` nastaveno na `True`.	
`PENALTY_BIAS`	Specifikuje konstantu b ve výpočtu fitness funkce, viz sekce 5.2.4.	
`SWAP_PROB`	Při použití `MSEQ = False` určuje pravděpodobnost, s jakou bude provedena vzájemná výměna priorit dvou náhodně zvolených pravidel, oproti pravděpodobnosti mutace symbolů v pravidlech. Pravděpodobnost výměny priority při mutaci pak bude 1 : `SWAP_PROB`.	
`SWAPSAME`	Pokud je nastaveno na `False`, brání při mutaci priority pravidel výměně priority pravidla sama se sebou (mutace bez efektu).	
`WRAP_SN_POS`	Ovlivňuje reprezentační funkci, konkrétně chování při přetečení ukazatele za hranice sítě. Pokud je nastaveno na `False`, k přetečení nedojde a ukazatel zůstane na původním místě. Pokud je nastaveno na `True`, ukazatel je umístěn na opačnou stranu sítě.	
`ADD_SYMMETRIC_COMPS`	Pokud je nastaveno na `True`, při přidání komparátoru (a, b) do řadicí sítě šířky n do této sítě také přidá k němu symetrický komparátor $(n - b, n - a)$.	

Název makra	Význam *Pokračováno z předchozí strany...*
`ADD_PRECOMPS`	Dovoluje pevně stanovit komparátory na počátku řadicí sítě (tj. před zahájením interpretace řetězce kódujícího síť), a to podle jednoho z předdefinovaných schémat. Implementovaná schémata zahrnují `COMPS_ONESEVEN` (komparátory velikosti 1 začínající na sudých vstupech), `COMPS_ONESODD` (komparátory velikosti 1 začínající na lichých vstupech) a `COMPS_GREEN` (tzv. Greenův filtr).
`ADD_POSTCOMPS`	Dovoluje pevně stanovit komparátory na konci řadicí sítě(tj. po ukončení interpretace řetězce kódujícího síť), a to podle jednoho z předdefinovaných schémat. Implementovaná schémata jsou stejná jako v případě výše uvedeného makra `ADD_PRECOMPS`.
`FORBIDDEN_SUBSTRS`	Obsahuje pole řetězců, jejichž výskyt jako podřetězce v řetězcové reprezentaci řadicí sítě způsobí zvýšení penalizace fitness funkce této sítě. Formálně nastavuje $\vec{o}_8 = 1$ a definuje způsob výpočtu hodnoty **FS**, viz sekce 5.2.4.

Tabulka A.1: *Přehled a význam jednotlivých uživatelských parametrů aplikace.*

A.2 Překlad aplikace

K produkční aplikaci je přiložen soubor `Makefile` připravený pro fakultní server Merlin, využívající kompilátor g++. Dále je přiložen soubor `Makefile-Anselm`, který byl použit pro překlad aplikace kompilátorem icpc a který využívá instrukčních sad SSE2/AVX2 prostřednictvím direktiv OpenMP 4. Překladem vznikne aplikace `sne` (Sorting Network Evolution), sestavená podle parametrů uvedených uživatelem v souboru `settings.h`.

A.3 Spuštění a výstup aplikace

Přeloženou aplikaci `sne` lze spustit jednoduše bez parametrů, jelikož všechny parametry již byly specifikovány před překladem. Formát výstupu aplikace je inspirován serializačním formátem JSON. Přesněji řečeno, výstupem jednoho spuštění aplikace je text odpovídající tělu struktury serializačního formátu JSON. Jednotlivé položky této struktury jsou:

- **seed** – seed náhodného generátoru,
- **success** – `true`, pokud bylo nalezeno funkční řešení,
- **gens** – počet vyhodnocených generací,
- **fitness** – hodnota fitness nejlepšího jedince,
- **growing** – `true`, pokud nalezené řešení tvoří platné řadicí sítě pro všechny velikosti sítí z `SN_WIDTH_GROWTH`,

- **compsU/T** – počty komparátorů nejlepšího nalezeného řešení pro velikosti sítí dle parametrů `SN_WIDTH` a `SN_WIDTH_GROWTH`, T označuje celkový počet komparátorů, U označuje počet využitých komparátorů,
- **delayU/T** – zpoždění nejlepšího nalezeného řešení pro velikosti sítí z `SN_WIDTH` a `SN_WIDTH_GROWTH`, T označuje zpoždění generované sítě, U označuje zpoždění sítě s odstraněnými redundantními komparátory,
- **strlen** – délka řetězce, kterým nejlepší nalezené řešení reprezentuje řadicí síť, pro velikosti sítí z `SN_WIDTH` a `SN_WIDTH_GROWTH`,
- **substrPenalty** – hodnota penalizace **FS**,
- **rules** – řetězec obsahující sadu pravidel nejlepšího nalezeného řešení.

V závislosti na uživatelských parametrech mohou některé z položek chybět. Příklad výstupu aplikace je uveden na obrázku A.1.

```
"seed":4116666858,
"success":true,
"gens":36000000,
"fitness":151147,
"growing":true,
"compsU":[5, 24, 62, 103, 169, 252],
"compsT":[5, 31, 96, 516, 2697, 14037],
"delayU":[3, 13, 25, 34, 50, 70],
"delayT":[3, 21, 46, 253, 1331, 6810],
"strlen":[12, 87, 449, 2386, 12472, 65324],
"substrPenalty":0,
"rules":"<S> -> US2DS2IDUS3D, <3> -> DDDUDDD, <D>2 -> S3DDD32S32USD2UD,
<2I> -> DD3SU, <D>D -> D, <D> -> 3D2SUU2IUUIU"
```

Obrázek A.1: *Ukázka výstupu aplikace.*

Ve výstupu aplikace chybí parametry experimentu, jelikož je předpokládáno, že pro stejné uživatelské nastavení bude spuštěno mnoho běhů aplikace. Přidání parametrů experimentu by tak způsobilo značnou redundanci. Doporučený postup použití aplikace je nejdříve nastavit parametry, přeložit aplikaci, spustit požadovaný počet experimentů a výsledky jednotlivých experimentů uložit do oddělených souborů. Poté lze spustit skript `mkjson.sh`, který výsledky shromáždí do jediného souboru formátu JSON a přidá informace o nastavení uživatelských parametrů experimentu. Pro ukázku tohoto postupu obsahuje `Makefile` cíl `run`, který provede 10 spuštění programu `sne` a výsledky uloží do jednotlivých souborů v adresáři `src/production/results/`. Poté lze použít skript `mkjson.sh` k transformaci výsledků do jediného souboru `json/results.json`. Tento postup je uveden na obrázku A.2. Ukázka struktury výsledného souboru `json/results.json` je uvedena na obrázku A.3.

```
$ make run
./run.sh
RUN 1
...
RUN 10
$ ./mkjson.sh
results: 0 tasks timed out, 10 completed successfully.
```

Obrázek A.2: *Ukázka doporučeného způsobu použití aplikace.*

```
{   "MAX_TIME":60,
    "SN_WIDTH":[4,6,8],
    "SN_WIDTH_GROWTH":[10,12,14,16],
    ...
    "FORBIDDEN_SUBSTRS":["IUI", "IDI"],
    "results":[
        {
            "seed":3371318761,
            "success":false,
            ...
            "rules":"<2> -> UDUUUIU, <S> -> SU2SDUID2D2, <U> -> D2UDIDUD"
        },{
            "seed":3114895025,
            "success":false,
            ...
        }...
]}
```

Obrázek A.3: *Ukázka struktury výsledného souboru ve formátu JSON.*

A.4 Analýza výsledků

Pro analýzu výsledků slouží program `restool`, nacházející se na přiloženém CD v adresáři `results/`. První funkcí tohoto programu je převedení výsledků do formátu `csv`. Soubor `csv` je snadnější manuálně analyzovat, například tabulkovým procesorem, jak ilustruje obrázek A.4. Pro vytvoření souboru `csv` z dat uložených v souboru `json/results.json` lze použít příkaz `../../results/restool.py json/results.json csv > results.csv`.

	A	B	C	D	E	F	G	H	I	J	K
1	seed	fitness	success	gens	delayT 4	delayT 6	delayT 8	delayU 4	delayU 6	delayU 8	compsT 4
2	4238293791	102285	True	200000000	3	8	12	3	7	11	5
3	3613339499	102271	True	200000000	3	7	11	3	7	11	5
4	2977086741	102258	True	200000000	3	7	14	3	7	11	5
5	1419767710	102257	True	200000000	4	10	17	4	9	16	6
6	2118661404	102257	True	200000000	4	10	17	4	9	16	6
7	172245589	102254	True	200000000	4	10	17	4	9	16	6

Obrázek A.4: *Ukázka manuálně analyzovaného csv souboru s výsledky.*

Druhou funkcí programu `restool` je zobrazení rostoucích řadicích sítí nalezených během experimentů. Pro vizualizaci řadicích sítí je použita knihovna Matplotlib pro Python 2.7, je tedy třeba, aby byla nainstalována. Zobrazení řadicích sítí lze provést příkazem `./restool experiment.json disp[p][s] <seed>`, kde `seed` je seed běhu programu, jehož výsledek má být zobrazen, získatelný manuální analýzou `csv` souboru. Parametr `disp[p][s]` je pak mód, který má být použit pro vykreslení výsledku. Jeho význam je následující:

- `disp` (display) – Zobrazí výsledné sítě se všemi generovanými komparátory, rozmístěnými do příslušných paralelních hladin. Redundantní komparátory budou vyznačeny červenou barvou.
- `dispp` (display pruned) – Zobrazí výsledné sítě po odstranění redundantních komparátorů. Komparátory budou rozmístěny do příslušných paralelních hladin.
- `disps` (display sequential) – Zobrazí výsledné sítě se všemi generovanými komparátory, rozmístěnými jeden po druhém v pořadí, v jakém byly generovány reprezentační funkcí z řetězcové reprezentace. Redundantní komparátory budou vyznačeny červeně.
- `dispps` (display pruned sequential) – Zobrazí výsledné sítě po odstranění redundantních komparátorů. Komparátory budou rozmístěny jeden po druhém v pořadí, v jakém byly generovány reprezentační funkcí z řetězcové reprezentace.

Po spuštění pak `restool` zobrazí vždy jedno okno s vizualizací generované řadicí sítě příslušné velikosti, počínaje nejnižší velikostí. Po zavření tohoto okna je otevřeno nové okno s vizualizací řadicí sítě vzniklé developmentem z předchozí sítě. Lze tak pozorovat růst sítě.

Při zobrazování výsledků experimentů s vývinem k-inkrementů řadicích sítí přitom není zobrazen pouze inkrement dané velikosti, nýbrž celá síť složená z optimální sítě velikosti n_0 a všech doposud vygenerovaných inkrementů. Při analýze tak lépe vynikne celková struktura výsledné sítě. Hodnoty v `csv` souboru však odpovídají pouze inkrementům, nikoliv celým sítím.

Program `restool` zároveň před zobrazením každé sítě provádí její ověření. Na standardní výstup přitom vypisuje řetězcovou reprezentaci zobrazené sítě, informace o počtu komparátorů a zpoždění sítě a zda je síť platná (`WORKING`) či ne (`FAILED`). Program `restool` lze ukončit stiskem `Ctrl+C`. Ukázka výstupu programu `restool` je uvedena na obrázku A.5.

A.5 Přiložené výsledky

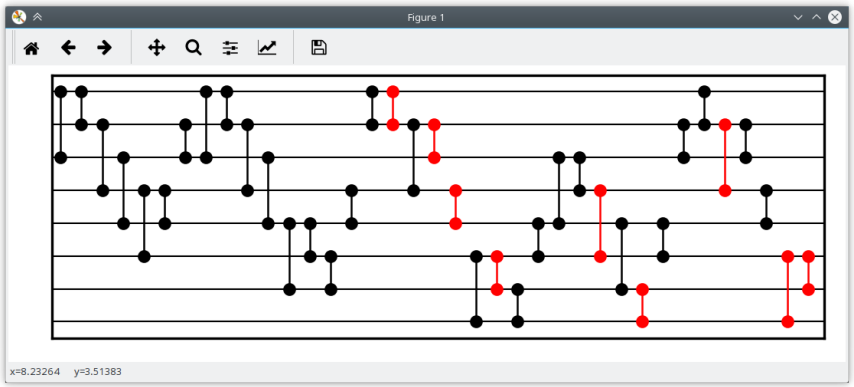
V adresáři `results/` na CD jsou přiloženy výsledky vybraných experimentů. Jedná se o následující experimenty:

- `grow2.json` – experiment, ze kterého vzešla rostoucí řadicí síť uvedená na obrázku 6.8. Tuto síť lze zobrazit následujícím příkazem:

  ```
  ./restool.py grow2.json disps 511838004
  ```

- `incr2.json` – experiment, ze kterého vzešel rostoucí 2-inkrement uvedený na obrázku 6.9. Rostoucí síť tvořenou tímto inkrementem lze zobrazit následujícím příkazem:

  ```
  ./restool.py incr2.json disps 1827734686
  ```




```
WIDTH = 8
String SU2SDUID2D2D2UDIDUDUDUUUIUSU2SDUID2D2DD2UDIDUDIDUDUUUIUDUDUU...
WORKING
Comparators: 28 / 37
Delay: 16 / 18
```

Obrázek A.5: *Ukázka grafického a konzolového výstupu programu* `restool`.

- `incr3.json` – experiment, ze kterého vzešel rostoucí 3-inkrement uvedený na obrázku 6.10. Rostoucí síť tvořenou tímto inkrementem lze zobrazit následujícím příkazem:

  ```
  ./restool.py incr3.json disps 263280613
  ```

- `incr4-primary.json` – experiment, ze kterého vzešel prvotní rostoucí 4-inkrement uvedený na obrázku 6.11. Rostoucí síť tvořenou tímto inkrementem lze zobrazit následujícím příkazem:

  ```
  ./restool.py incr4-primary.json disps 3003060788
  ```

- `incr4-opti.json` – experiment, ze kterého vzešel evolučně optimalizovaný rostoucí 4-inkrement uvedený na obrázku 6.12. Rostoucí síť tvořenou tímto inkrementem lze zobrazit následujícím příkazem:

  ```
  ./restool.py incr4-opti.json disps 1345456
  ```