StuCoSReC

Proceedings of the 2016 3rd Student Computer Science Research Conference

University of Primorska Press

StuCoSReC
Proceedings of the 2016 3rd Student Computer Science Research Conference

Edited by
Iztok Fister jr. and Andrej Brodnik

Reviewers and Programme Committee
Zoran Bosnić ▪ University of Ljubljana, Slovenia
Borko Bošković ▪ University of Maribor, Slovenia
Janez Brest ▪ University of Maribor, Slovenia
Andrej Brodnik ▪ University of Primorska
and University of Ljubljana, Slovenia
Mojca Ciglarič ▪ University of Ljubljana, Slovenia
Jani Dugonik ▪ University of Maribor, Slovenia
Iztok Fister ▪ University of Maribor, Slovenia
Iztok Fister Jr. ▪ University of Maribor, Slovenia
Goran Hrovat ▪ University of Maribor, Slovenia
Andres Iglesias ▪ Universidad de Cantabria, Spain
Branko Kavšek ▪ University of Primorska, Slovenia
Miklos Kresz ▪ University of Szeged, Hungary
Niko Lukač ▪ University of Maribor, Slovenia
Uroš Mlakar ▪ University of Maribor, Slovenia
Peter Rogelj ▪ University of Primorska, Slovenia
Xin-She Yang ▪ Middlesex University, United Kingdom
Aleš Zamuda ▪ University of Maribor, Slovenia

Published by
University of Primorska Press
Titov trg 4, SI-6000 Koper

Editor-in-Chief
Jonatan Vinkler

Managing Editor
Alen Ježovnik

Koper, 2016

ISBN 978-961-6984-37-9 (pdf)
www.hippocampus.si/ISBN/978-961-6984-37-9.pdf
ISBN 978-961-6984-38-6 (html)
www.hippocampus.si/ISBN/978-961-6984-38-6/index.html

CIP - Kataložni zapis o publikaciji
Narodna in univerzitetna knjižnica, Ljubljana

004(082)(0.034.2)

STUDENT Computer Science Research Conference (3 ; 2016 ; Ljubljana)
Proceedings of the 2016 3rd Student Computer Science Research Conference - StuCoSReC [Elektronski vir] / [edited by Iztok Fister and Andrej Brodnik]. - El. knjiga. - Koper : University of Primorska Press, 2016

Način dostopa (URL): http://www.hippocampus.si/isbn/978-961-6984-37-9.pdf
Način dostopa (URL): http://www.hippocampus.si/isbn/978-961-6984-38-6/index.html

ISBN 978-961-6984-37-9 (pdf)
ISBN 978-961-6984-38-6 (html)
1. Fister, Iztok, ml.
286799104

Preface

Dear reader,

After two successful editions of the Student Computer Science Research Conference that were organized in Maribor and Ljubljana, here are the proceedings of the 3rd Student Computer Science Research Conference that is being hosted this year by the University of Primorska in Koper.

The present Conference has again attracted many undergraduate and graduate students presenting their research ideas. As in the past, there are contributions from many Computer Science research areas and even some contributions from other research areas such as, for example, Mechatronics. This edition consists of 11 papers that are divided into National and International sessions. The National session consists of 4 papers, while the International session consists of 7 papers. All papers were reviewed by at least two members of the International Program Committee, who made comments that were sent back to authors in order to improve their papers.

In a nutshell, The 3rd Student Computer Science Research Conference 2016 once again confirms the main missions that were set up back in 2014 when we wanted to establish a platform where Computer Science students could come together, make new friends and connections, discuss their research projects and ideas and encourage each other on their way to becoming Computer Scientists.

Contents

Zaznavanje plagiatov s pomočjo n-gramov

Žiga Leber
Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko
Smetanova ulica 17, SI-2000
Maribor, Slovenija
ziga.leber1@student.um.si

Luka Horvat
Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko
Smetanova ulica 17, SI-2000
Maribor, Slovenija
luka.horvat@student.um.si

Patrik Kokol
Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko
Smetanova ulica 17, SI-2000
Maribor, Slovenija
patrik.kokol@student.um.si

Marko Očko
Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko
Smetanova ulica 17, SI-2000
Maribor, Slovenija
marko.ocko@student.um.si

POVZETEK

V članku predstavimo postopek avtomatske detekcije plagiatov s pomočjo n-Gramov in izboljšavo z algoritmom Stupid Backoff. Algoritem deluje na podlagi primerjave povedi iz dokumenta z referenčnim korpusom. Iz rezultatov je razvidno, da izboljšan algoritem deluje bolje. Na koncu še podamo zaključke in naše mnenje.

Ključne besede

Plagiati, n-grami, Stupid Backoff

1. UVOD

Plagiat je objava del, ki se pripisujejo drugemu avtorju, brez da bi citirali vir iz katerega smo jih pridobili. To predvsem velja v primeru, kjer lahko legitimno pričakujemo, da je delo originalno. Dejanje je storjeno, z namenom pridobitve koristi, kreditnih točk ali druge oblike dobička [4]. Pred 18. stoletjem v Evropi kopiranja oz. prepisovanja in posnemanja niso dojemali kot nemoralno ali prepovedano početje, temveč so celo spodbujali k posnemanju velikih mojstrov in zgledovanju pri njih. V umetnosti je tudi danes težko določiti, če gre za plagiat ali ne, saj se nekateri umetniki zgledujejo drug pri drugem, lahko pa tudi nezavedno vplivajo drug na drugega. V znanosti žene plagiatorje želja po ugledu ali zaslužku brez lastnega truda, lahko pa pride do kraje intelektualne lastnine tudi samo zaradi pomanjkanja svojega znanja in sposobnosti [9]. Plagiatorstvo se še posebaj pojavlja v sodobnem času, saj lahko preprosto dostopamo do del drugih avtorjev s pomočjo elektronskih virov. Zato so raziskovalci razvili veliko različnih postopkov za avtomatsko detekcijo plagiatov.

V praksi se pojavlja več načinov nelegitimne ponovne uporabe besedila, ki jih lahko na grobo kategoriziramo v naslednji skupini. Plagiati, ki temeljijo na dobesednem kopiranju, kjer so deli besedila v celoti prevzeti v originalni obliki. Te je praviloma najlažje zaznati. V drugo skupino sodijo plagiati, kjer so bili dobesedno kopirani segmenti besedila zamaskirani. V tem primeru plagiator namenoma prevzame tuje ideje in jih prepiše v svojem slogu brez omembe originalnega avtorja. Pri takšni obliki so besede zamenjane s sinonimi, prav tako pa je pogosto zamenjan tudi njihov vrstni red. V teh primerih je posledično plagiat dosti težje zaznati [5, 1].

V tem članku smo za uporabo plagiatov preizkusili metodo opisano v [1]. Ta temelji na primerjavi večih povedi z referenčnim korpusom na podlagi n-gramov (zaporednih n elementov v dani sekvenci npr. zaporednih n besed v povedi). Postopek smo dodatno izboljšali z uporabo algoritma Stupid Backoff. Delovanje metode preizkusimo za različne velikosti n-gramov. Iz pridobljenih rezultatov je razvidno, da tako originalna, kot referenčna metoda delujeta boljše pri nizkih vrednostih n. Predlagan algoritem izboljša F_1-mero za 0.44 %.

Članek je organiziran v naslednje odseke. V odseku 2 predstavimo sorodna dela. Nato referenčno metodo in naše izboljšave opišemo v poglavju 3. Sledijo rezultati in diskusija v odseku 4. Članek s kratkim povzetkom zaključimo v odseku 5.

2. SORODNA DELA

Trenutno obstaja že kar nekaj metod za ugotavljanje plagiatov. Sistem PPChecker [6] razdeli sumljivo besedilo v manjše dele, kot so odstavki in povedi, ter nato posamezne manjše dele primerja z referenčnim korpusom. Problem te metode je, spreminjanje vrstnega reda besed v stavkih, kajti tega ta sistem ne bo zaznal. V Ferret [7] sistemu za detekcijo plagiatov, se celoten testni dokument razdeli v različne

n-grame in potem se primerjajo število unikatnih ter ujemajočih n-gramov. Večje je ujemanje, večja je verjetnost prisotnosti plagiata. Slabost tega pristopa je, da se išče podobnost oz. primerjava celotnega dokumenta in ne posameznih manjših delov. Obstajajo tudi posebne metode pri katerih dejansko ne iščemo ujemanja celotnih stavkov oz. besedila, ampak samo ujemanje določenih besed. V Stamatatos-ovem [3] sistemu se uporablja iskanje ujemanje tako imenovanih prekinitvenih besed, ki so seznam frekvenčno zelo velikokrat uporabljenih besed v besedilu. Izkaže se, da lahko z ujema-njem teh prekinitvenih besed zelo dobro ugotavljamo plagiate, tudi ko so besedila zelo spremenjena.

Sami smo se odločili, da bomo uporabili metodo, ki so jo uporabili v originalnem članku [1], to je uporaba srednje poti med PPChecker in Ferret metodama. Namesto, da bi primerjali n-grame celotnega dokumenta, se dokument najprej razdeli na manjše dele, to je povedi, in nato se nad njimi ugotavlja ujemanje z referenčnim korpusom.

3. METODA

Z naslednjimi metodami smo poskušali odgovoriti na vprašanje, ali je poved s_i plagiat glede na dokument d iz korpusa. Najprej smo implementirali referenčno metodo iz članka, nato pa smo jo poskusili izboljšati s pomočjo algoritma Stupid Backoff.

3.1 Originalna metoda

Imamo dokument r, za katerega sumimo, da je plagiat. r razdelimo na povedi s_i. Imamo tudi referenčni korpus D, ki je sestavljeni iz dokumentov d. Torej nas zanima ali je poved s_i plagiat, ki izvira iz dokumenta d. Prepisane povedi so lahko spremenjene oz. se je kopiral samo del povedi, posamezne prepisane besede pa so lahko v pomešanem vrstnem redu, kar pa oteži samo ugotavljanje plagiata. Ta problem smo rešili z uporabo n-gramov, kajti malo verjetno je, da se bodo n-grami iz različnih dokumentov ujemali. Verjetnost seveda pada z višanjem n-ja. Ker pa so prepisane povedi sestavljene iz delčkov besedila iz različnih delov originalnega dokumenta pa smo celotni dokument r razdeliti na n-grame in ne na povedi.

Postopek algoritma je naslednji:

1. Dokument, ki vsebuje plagiate razdelimo na posamezne povedi s_i.
2. Poved s_i je razdeljena na n-grame, ki sedaj predstavljajo poved.
3. Referenčni dokument d je razdeljen na n-grame brez predhodnega deljenja na povedi, saj lahko plagiat vsebuje različne dele originalnega besedila.
4. S primerjanjem n-gramov povedi s_i z dokumentom ugotovimo ali je poved plagiat, kot je opisano v nadaljevanju.

Za preverjanje plagiata, smo primerjali dobljene n-grame s tistimi iz referenčnega besedila s pomočjo enačbe (1).

$$C(s_i|d) = \frac{|N(s_i) \cap N(d)|}{|N(s_i)|} \tag{1}$$

kjer so $N(.)$ n-grami. Če je vrednost $C(s_i|d)$ nad določenim pragom, potem je s_i klasificiran kot plagiat. Z učenjem na označeni učni množici smo poiskali prag za določene n-grame.

3.2 Izboljšana metoda

Referenčna metoda za ugotavljanje plagiata uporablja n-grame fiksne velikosti in tako ne upošteva manjših n-gramov. Pri večjih n-gramih zato pade učinkovitost algoritma, saj so možnosti enakih zelo majhne. Zato smo dodali pri ugotavljanju vsebnosti algoritem Stupid Backoff [2] s katerim smo začeli upoštevati tudi manjše n-grame.

Predlagan algoritem ima prva dva koraka enaka kot referenčni, nato pa sledijo:

1. Referenčni dokument d razdelimo na vse n-grame pri čemer je $n \in \{1, 2, ..n\}$.
2. Vsak n-gram povedi s_i pri izbranem n testiramo vsebnost z enačbo (2). Če je n-gram vsebovan prištejemo 1, drugače pa rekurzivno preverjamo manjše n-grame.
3. Naredimo vsoto vsebnosti vseh n-gramov povedi, jo normaliziramo in nato preverimo glede na prag, ali je poved plagiat ali ne.

$$\begin{aligned} C(b^i_{i-k+1}) &= \begin{cases} 1, & f(b^i_{i-k+1}) > 0 \\ 0.4C(b^i_{i-k+2}), & \text{else} \end{cases} \\ C(b_i) &= \begin{cases} 1, & f(b_i) > 0 \\ 0, & \text{else} \end{cases} \end{aligned} \tag{2}$$

Pri čemer je b^i_{i-k+1} iskan n-gram in $f(b^i_{i-k+1})$ frekvenca iskanega n-grama.

4. REZULTATI IN DISKUSIJA

V sledečem podpoglavju opišemo zgradbo in obliko izbranega korpusa. Nato sledi podpoglavje v katerem podrobneje predstavimo eksperimentalne rezultate.

4.1 Korpus

Za potrebe testiranja algoritmov smo uporabili METER korpus, ki je zgrajen za analizo po-uporabe besedila. Na voljo je v več različnih formatih, mi smo uporabili korpus v formatu XML. Podatki v korpusu so deljeni na dva dela. Prvi del vsebuje 771 novic, ki jih je napisala Press Association (PA) in večinoma vsebuje članke o dogajanjih na sodiščih. Drugi del korpusa so članki različnih Britanskih novic [8]. Britanski članki so v korpusu razdeljeni na povedi, ki so nato označeni z *rewrite, verbatim* ali *new*. Te oznake nam povedo, ali je del povedi prepisan, povzet ali na novo napisan glede na referenčne PA članke. Za posamezno poved velja, da je plagiat, če je večji del besed označen kot prepisan ali povzet. Na splošno se uporablja enačba $|s_{iV} \cup s_{iR}| > 0.4|s_i|$ pri čemer s_{iV} in s_{iR} predstavljata število povzetih in prepisanih besed, s_i pa število vseh besed v povedi. Vse skupaj je v korpusu 3600 povedi Britanskih novic, od tega jih je po tem pravilu 61% označenih kot plagiat.

4.2 Eksperimentalni rezultati

Na podlagi eksperimenta smo določili optimalne vrednosti parametrov in preverili natančnost delovanja obeh metod. Eksperiment smo izvedli tako, da za vse n-grame, $n \in [1, 5]$, izvedli izčrpno iskanje optimalnega praga, pri katerem je poved klasificirana kot plagiat. Za izmero natančnost metod smo uporabili metrike preciznost, priklic in F_1-mero.

Pri unigramih je priklic pri obeh metodah, za vse vrednosti praga pod 0,7, zelo visok (skoraj 1,0). Posledično zaznamo skoraj vse plagiate. Klub temu pa je preciznost nizka, saj je verjetnost, da se posamezna beseda iz povedi pojavi v referenčnem dokumentu dokaj velika. Pri večjih n-gramih je priklic nižji, saj na stopnjo zaznavanja vpliva tudi zamenjan vrstni red. Najboljši rezultat smo dobili pri bigramih pri pragu 0,25, kjer je vrednost F_1-mere 0,76, kar je razvidno tudi iz slike 1.

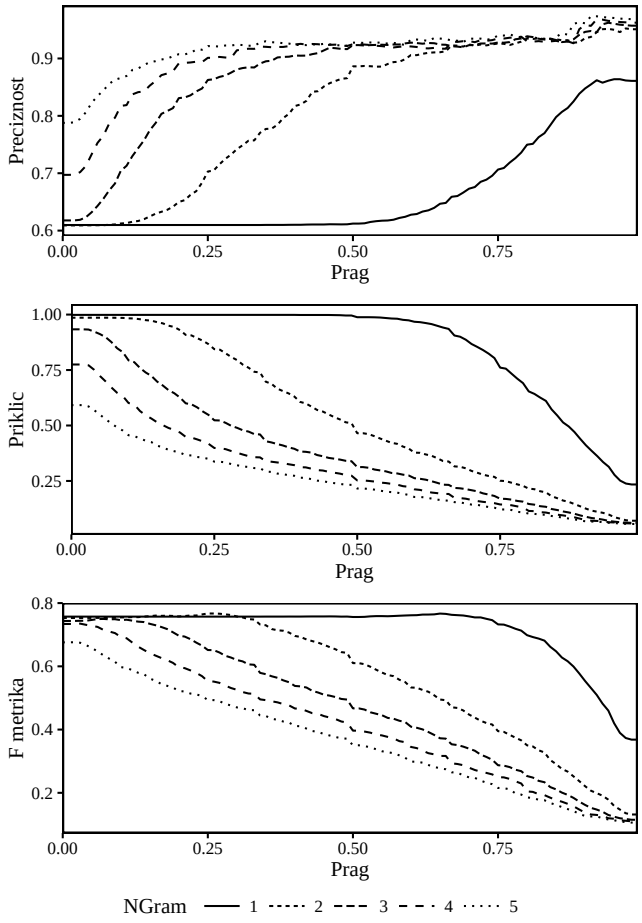


Figure 1: Rezultati za referenčno metodo

Rezultati, ki smo jih dobili so v vseh primerih za približno 12 % boljši od tistih iz [1]. Primerjava izboljšanih rezultatov s tistimi, ki jih je dosegla referenčna metoda je predstavljena v tabeli 1. V [1] so se, podobno kot pri nas, kot najboljši izkazali bigrami, vendar je v njihovem članku F_1-mera 0,68, pri nas pa 0,76. Predvidevamo, da je to zato ker smo pri predprocesiranju odstranili vsa ločila. Posledično je število vseh možnih besed v našem primeru dosti nižje, kar poveča verjetnost, da se bo beseda v povedi pojavila v referenčnem dokumentu.

	naša		referenčna	
n	prag	F_1-mera	prag	F_1-mera
1	0,65	0,7668	—	—
2	0,40	0,7705	0,34	0,68
3	0,22	0,7641	0,17	0,66
4	0,10	0,7581	—	—
5	0,04	0,7505	—	—

Table 1: Primerjava rezultatov

Z uporabo izboljšane metode se F_1-mera za vse n-grame nekoliko izboljša. Učinek je najbolj očiten pri večjih n-gramih, kjer je izboljšava skoraj 10 %. To je posledica dejstva, da v primeru, ko algoritem ne najde ustreznega n-grama uporabi naslednji (n-1)-gram. Ponovno imajo najboljšo F_1-mero bigrami, vendar, kot lahko vidimo na sliki 2, trigrami ne zaostajajo za veliko. Najboljša F_1-mera (0,77) je pri pragu 0,4, kar je za 0,44 % boljše od prejšnega rezultata (0,76).

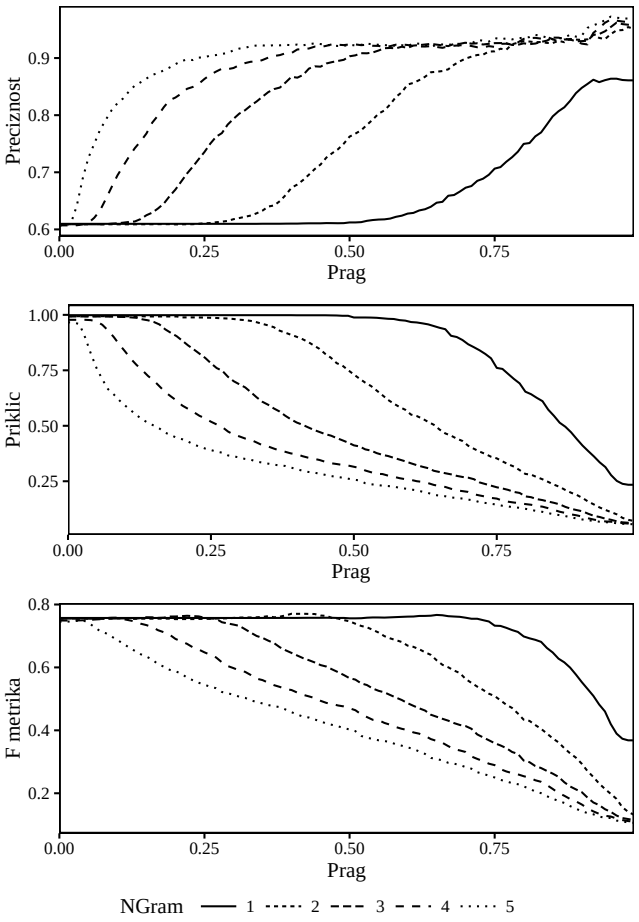


Figure 2: Rezultati za izboljšano metodo

5. ZAKLJUČEK

Dobili smo boljše rezultate, kot v referenčnem članku. Referenčno metodo smo dodatno izboljšali s pomočjo algoritma Stupid Backoff. Pri izbiri n-jev med 2 in 5 za n-grame je

izboljšana metoda dosegla boljše rezultate. Na primer pri izbiri petgramov smo dobili 10% boljše rezultate. Pri izbiri bigramov pa so se ti izboljšali za 0,44 %. Ugotovili smo tudi, da je najbolj optimalna izboljšava bila pri n=5.

6. REFERENCES

[1] A. Barrón-Cedeño and P. Rosso. *Advances in Information Retrieval: 31th European Conference on IR Research, ECIR 2009, Toulouse, France, April 6-9, 2009. Proceedings*, chapter On Automatic Plagiarism Detection Based on n-Grams Comparison, pages 696–700. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[2] T. Brants, A. C. Popat, P. Xu, F. J. Och, and J. Dean. Large language models in machine translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 858–867, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[3] S. E. Plagiarism detection using stopword n-grams. *J. Am. Soc. Inf. Sci., 62: 2512–2527. doi: 10.1002/asi.21630*, 2011.

[4] T. Fishman. "we know it when we see it" is not good enough: toward a standard definition of plagiarism that transcends theft, fraud, and copyright. 2009.

[5] B. Gipp. Citation-based plagiarism detection–idea, implementation and evaluation. 2009.

[6] G. A. Kang N. Ppchecker: Plagiarism pattern checker in document copy detection. *TSD 2006. LNCS, vol. 4188,pp. 661–667. Springer, Heidelberg*, 2006.

[7] M. Lyon C., Barrett R. A theoretical basis to the automated detection of copying between texts, and its practical implementation in the ferret plagiarism and collusion detector. *Plagiarism: Prevention, Practice and Policies Conference, Newcastle*, 2004.

[8] S. L. P. Paul Clough, Robert Gaizauskas. Building and annotating a corpus for the study of journalistic text reuse. *3rd International Conference on Language Resources and Evaluation*, 2002.

[9] Wikipedia. Palgiatorstvo — Wikipedia, the free encyclopedia, 2016. [Online; accessed 25-Sept-2016].

Detekcija jezika v besedilih s šumom

Tilen Škrinjar
Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko
Smetanova 17
2000 Maribor, Slovenija
skrinjar.tilen@gmail.com

Matej Trop
Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko
Smetanova 17
2000 Maribor, Slovenija
trop.matej@gmail.com

Filip Urh
Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko
Smetanova 17
2000 Maribor, Slovenija
filipurh@gmail.com

POVZETEK

Večina besedil na internetu je slovnično nepravilnih (oziroma vsebujejo šum), saj uporabniki pri komentiranju in na raznih forumih ne uporabljajo knjižnega jezika. V članku predstavljamo pet metod za razpoznavo jezika iz besedil s šumom. Večina metod, ki se ukvarjajo s to problematiko, imajo težave pri razpoznavi jezika, ki so kratka ali pa vsebujejo napake.

Eksperimenti, ki smo jih izvedli nad testnimi množicami, so pokazali, da so lahko omenjene metode dokaj uspešne pri identifikaciji jezikov. Naša metoda pa uspešnost poveča še za 3.6 %.

1. UVOD

Z napredovanjem tehnologije, ki omogoča vedno hitrejše deljenje informacij med ljudmi, se je povečala količina tekstovnih podatkov na internetu in v podatkovnih bazah. Zaradi tega je črpanje informacij iz takih podatkov postalo težavno, še posebej če mora to opravljati človek. Na to temo je bilo opravljenih veliko raziskav, ki so stremele k avtomatični ekstrakciji podatkov iz teksta. Ponavadi pa moramo pri teh metodah vnaprej vedeti, v katerem jeziku so informacije, ki jih analiziramo. Zaradi tega postane identifikacija jezika v nekem dokumentu zelo pomemben del analiznega procesa. Zaradi pomembnosti veliko raziskovalcev preučuje avtomatično identifikacijo jezikov.

Izziv predstavlja predvsem problem, kako prepoznati zelo kratka sporočila, kakršna na primer uporabljamo na raznih forumih ali v tekstovnih SMS sporočilih. Velika večina metod namreč temelji na statističnem modelu. Za izgradnjo takšnega modela potrebujemo relativno dolgo testno besedilo, če želimo jezik pravilno identificirati. V nasprotnem primeru algoritem ne zna odločiti, v katerem jeziku je testno besedilo.

Dandanes poznamo več kategorij podatkov, ki so lahko spletne novice, izobraževalne spletne strani, forumi ipd. Za raziskovalce so najbolj zanimiva socialna omrežja in forumi, kjer se izmenja največ mnenj o raznih produktih, filmih, glasbi, kar so še posebej zanimivi podatki za oglaševalce, vendar se pri razpoznavi besedil iz teh dveh kategorij pojavijo težave. Ta besedila namreč pogosto vsebujejo okrajšave, "tage" ali pa so v celoti zapisana v SMS stilu in jih je nemogoče razpoznati z običajnimi postopki.

V sklopu te raziskave smo preučili in implementirali štiri predlagane metode, ki so poimenovane CBA (angl. Character Based identification Algoritm), ki temelji na znakih iz učnega korpusa, WBA (Word Based identification Algorithm), ki temelji na besedah ter dva hibridna algoritma. Pri tem zaporedno ali pa paralelno izvedemo algoritma CBA in WBA [3].

V drugem poglavju bomo na kratko predstavili nekaj podobnih del na področju identifikacije jezika. Nato bomo predstavili naš učni korpus in identifikacijske metode, ki smo jih implementirali ter izboljšavo le-teh. V zadnjem podpoglavju bodo predstavljeni rezultati in primerjave uporabljenih metod.

2. SORODNA DELA

V članku [5] so za korpus uporabljali Wikipedio [2] in EuroParl [1]. Za detekcijo jezika so uporabili trigrame, ki delujejo zelo dobro na daljših besedilih (skoraj 100%), pri kratkih pa detekcija pade pod 50% in metoda postane neuporabna. Zanimivo je tudi, da je bila detekcija bolj uspešna z učno množico besedil iz Wikipedie, medtem ko je EuroParl korpus ustvarjen za strojno učenje imel tudi do 90% napačno identifikacijo.

V članku [7] so se ukvarjali z N-grami in razdaljo med rezultati. Iz učne množice ustvarijo znakovne in besedne N-grame, nato z enačbo za navzkrižno entropijo ali drugih enačb za razdalje izračunajo razdaljo med najboljšimi učnimi N-grami in testnim besedilom. Glede na razdaljo se izračuna končna ocena, na podlagi katere se identificira jezik besedila. Najbolje je delovala metoda z navzkrižno entropijo, vse metode pa so imele verjetnost detekcije nad 90%.

Drugi algoritmi so delovali na podlagi strojnega učenja [8] in na podlagi modela HMM (angl. Hidden Markov Models). Model se med drugim uporablja v razpoznavanju govora, biologiji, meteorologiji, ipd. Iz učne množice ustvarimo vektorje besed in z njimi spreminjamo parametre modela. Re-

zultate so izmerili samo na petih jezikih in jih primerjali z N-grami spremenljive dolžine. Rezultati so dokaj podobni, HMM model v nekaterih primerih deluje malo bolje.

Algoritem za detekcijo jezika je tudi metoda z N-grami [4], kjer si ustvarimo profile jezikov in nato iz testnega besedila ustvarimo N-grame ter preverimo kateremu profilu se najbolj prilegajo. Ta metoda ima kar nekaj težav pri detekciji v primerjavi s prejšnjimi ne glede na dolžino besedila.

V članku [6] so predlagali identifikacijo z odločitvenimi drevesi. Eksperiment so izvedli le na dveh arabskih jezikih: arabščini in perzijščini. Za oba jezika je metoda delovala zelo dobro.

3. METODE ZA IDENTIFIKACIJO JEZIKOV

V tem podpoglavju bomo opisali metode, ki so jih razvili v članku [3]. Te metode smo tudi implementirali. Detekcija poteka v več korakih. Najprej v program naložimo referenčne profile za vse jezike, tj. učni korpus besed in znakov, nato preberemo datoteko ali več datotek. Iz njih odstranimo nepotrebne znake, ki bi lahko vplivali na detekcijo in besedilo razrežemo na besede in znake. Nad temi podatki nato opravimo detekcijo.

Prvi algoritem se imenuje CBA (angl. Character Based identification Algorithm) in temelji na znakih v jeziku. Učni korpus uporablja zbirko predefiniranih znakov za vsak jezik. Algoritem izračuna frekvenco vsakega znaka v besedilu. Nato sešteje vse frekvence po enačbi (1). Na koncu klasificiramo besedilo glede na najvišjo frekvenco.

$$Sum_i = \sum_{j=1} frequency_{ij} \tag{1}$$

Problem se pojavi, ko več jezikov deli skupni nabor učnih znakov in zaradi tega dobi isto frekvenco pri seštevku znakov. To rešimo tako, da sledimo posebnemu testnemu zaporedju za vsak jezik:

if $max\_freq = slo\_freq$ and $max\_freq \neq bos\_freq$
then Return Slovenian
else Pass to next test
end if

Kjer je max_freq največji seštevek vseh frekvenc, slovenian_freq in bos_freq pa sta frekvenci za posamezen jezik.

Zaradi različnih jezikov smo sami definirali vrstni red jezikov. Algoritem ponavljamo v naslednjem zaporedju: hrvaščina, srbščina (latinica), bosanščina, slovenščina, češčina, poljščina, slovaščina, makedonščina, bolgarščina. Ta način ni najboljši, saj lahko različne vhodne jezike identificiramo različno dobro.

WBA (angl. Word Based identification Algorithm) je zelo podoben prejšnjemu algoritmu, le da deluje nad besedami besedila, ki ga identificiramo. Besedilo razčlenimo na besede in izračunamo frekvence za vsako besedo. Nadaljnja klasifikacija deluje enako, kot je opisano v metodi CBA.

Predlagana sta tudi dva hibridna algoritma, ki bi naj izboljšala natančnost zaznavanja jezika. Prvi hibridni pristop je sekvenčni in deluje tako, da najprej izvedemo algoritem CBA in če ta kot najboljši rezultat vrne enake frekvence za več jezikov, izvedemo še WBA. V tem primeru upoštevamo rezultate algoritma WBA.

Druga hibridna metoda temelji na obeh algoritmih CBA in WBA. Pri drugem hibridnem pristopu oba algoritma izvedemo paralelno in ko opravita izračune za vse jezike, seštejemo posamezne frekvence po enačbi (2) in nato razpoznamo, za kateri jezik gre na podlagi postopka opisanega pri CBA.

$$Sum = freqCBA + freqWBA \tag{2}$$

Naš doprinos je tretji hibridni algoritem, pri katerem smo uporabili drugačen izračun kot je pri ostalih algoritmih. Algoritem CBA smo polovično utežili, saj daje slabše rezultate. Prilagodili smo algoritem WBA. Tu smo upoštevali vrstni red najpogostejših besed. Frekvence za besede se izračunajo po enačbi (3):

$$(N - rank)/N, \tag{3}$$

kjer rank predstavlja red pogostosti besede v določenem jeziku, N pa število besed v učnem korpusu, ki pri vseh jezikih znaša dvajset. Red najpogostejše besede se začne z 0 in se pri manj pogostejših besedah nadalje poveča za 1. Za primer vzamemo tri besede. Prva beseda se nahaja na prvem mestu, druga na petem, tretja pa na dvajsetem mestu najpogostejših besed v jeziku. Število vseh najpogostejših besed je dvajset. Za prvo besedo izračunamo frekvenco kot (20 - 0) / 20 = 1, za drugo (20 - 4) / 20 = 0.8 in za tretjo (20 - 19) / 20 = 0.05. Če skupaj strnemo metodo, dobimo naslednjo enačbo:

$$Sum = \frac{1}{2} * freqCBA + modFreqWBA, \tag{4}$$

kjer se modFreqWBA izračuna po enačbi (3).

4. EKSPERIMENT

4.1 Učni in testni korpus

Za učni korpus in testna besedila smo uporabili kodiranje UTF-8, saj podpira največ znakov, ki jih vsebujejo testirani jeziki.

V naši raziskavi smo se osredotočili na jezike, ki so si med seboj podobni. Zaradi tega smo sestavili svoj učni in testni korpus. Potrebne informacije smo pridobili na svetovnem spletu. Pri učnem korpusu smo definirali besede in znake, ki zaznamujejo jezik. V članku [3] so za razpoznavo Uporabili dvajset najpogostejših besed za posamezen jezik in nekaj znakov, na osnovi katerih se jezik razlikuje od drugih. Na enak način smo se tudi mi lotili izdelave korpusa. Ta vsebuje podatke za devet jezikov. Večina jezikov si je med seboj podobna, saj le-ta spadajo v skupino slovanskih jezikov. Razpoznavali smo naslednje jezike: bolgarščina, bosanščina,

češčina, hrvaščina, srbščina (latinica), makedonščina, poljščina, slovaščina in slovenščina. Bosanščina, hrvaščina in srbščina so si med seboj zelo podobni, medtem ko se bolgarščina precej razlikuje od ostalih jezikov. Testni korpus vsebuje kratka besedila do stopetdeset besed. Ta besedila niso sestavljena iz pravilno napisanih besedil, ampak vsebujejo napake, okrajšave, nepomembne znake in citate itd. Za testni korpus smo uporabili petindvajset testnih besedil za posamezni jezik. Vsak testni primer je dolg nekaj stavkov. Besedila smo dobili iz različnih spletnih forumov, kot so športni forum, elektrotehniški forum in ostali.

4.2 Analiza in rezultati

Vse metode, ki smo jih implementirali, smo nato tudi preizkusili na praktičnih primerih. Za vsak jezik smo ustvarili matriko klasifikacijskih napak in v njej prikazali dobljene rezultate. Naše rezultate smo poskušali primerjati z orodji za razpoznavo jezika, kot sta *Google Prevajalnik* in *Prevajalnik Bing*. Prvi ne prepoznava srbščine (latinica), drugi pa bosanščine in makedonščine. Zaradi tega primerjava rezultatov med orodji ni smiselna.

Klasifikacijo smo opravili nad vsemi jeziki. V tabeli 3 in 4 so prikazani rezultati teh meritev. Tabela predstavlja matriko klasifikacijskih napak. Prikazani so jeziki testnega besedila in uporabljene metode. V eni celici so štiri vrednosti. Vsaka vrednost pripada eni od metod. Posebej so za vsako metodo, po vrsti: CBA, WBA, HA1, HA2 in HA3. Za primer si vzemimo celico v zadnjem stolpcu in vrstici v tabeli 4. Ta znaša 18/22/18/24. Prva vrednost 18 je število pravilno razpoznanih besedil po algoritmu CBA, vrednost 22 je število razpoznanih besedil po metodi WBA, nadalje 18 po metodi HA1, 22 po metodi HA2 in 24 po naši metodi oziroma HA3. V tabeli 1 in tabeli 2 lahko zasledimo tudi uspešnost posameznih metod v jezikih in skupaj. Če pogledamo za primer zadnjo vrstico v tabeli 1, opazimo, da je najbolj pravilno klasificiral besedila naš algoritem - z 96 % natančnostjo, najslabše pa CBA in HA1 algoritem z 72 % natančnostjo. Iz rezultatov vidimo, da metode slabše delujejo med jeziki, ki so si podobni (srbščina, hrvaščina in bosanščina). Metoda CBA deluje odvisno od predefiniranih znakov. Vidimo, da so rezultati enaki pri metodi CBA in HA1. Razlog je v tem, da pri testih nismo dobili enakih frekvenc. Dobro deluje uravnotežena metoda HA2. V naši metodi smo bolj utežili metodo WBA, saj daje boljše rezultate. CBA smo manj utežili. Zaradi teh dveh izboljšav smo dobili 3.6 % izboljšavo. Boljšo razpoznavo besedil smo dobili pri češčini, slovaščini in slovenščini.

Tabela 1: Uspešnost metod v posameznem jeziku %

	Uspešnost metod %				
	CBA	WBA	HA1	HA2	HA3
Bolgarščina	100	100	100	100	100
Bosanščina	80	4	80	4	4
Češčina	48	88	48	92	96
Hrvaščina	0	92	0	92	84
Srbščina (latinica)	0	16	0	16	40
Makedonščina	0	100	0	100	96
Poljščina	72	100	72	100	100
Slovaščina	88	80	88	92	96
Slovenščina	72	88	72	88	96

Tabela 2: Uspešnost metod v %

	Uspešnost metod %				
	CBA	WBA	HA1	HA2	HA3
Vsi jeziki	51.1	74.2	51.1	76	79.6

5. ZAKLJUČEK

V članku smo analizirali obstoječe metode pri zaznavanju jezika in predlagali svojo metodo. Algoritem, ki je deloval na principu znakov, je slabo zaznaval jezike, saj je težko narediti nekakšen statistični model, ki bi prinesel dovolj raznolike vzorce v učni korpus. Več sreče smo imeli z besedami, kjer je algoritem zaznaval jezike precej bolje. Tudi hibridna metoda in naša metoda sta vračali boljše rezultate. Naš algoritem deluje najbolje, saj smo pogostost pojavljanja besed v jeziku dodatno obtežili.

Učni korpus, ki smo ga uporabili, je bil sestavljen iz več 100 besedil iz spleta, ki so bila pretvorjena v besedne in znakovne učne baze podatkov v obliki tekstovnih datotek. Na podlagi prejšnjih metod smo poskušali izboljšati detekcijo, ki se razlikuje na drugačen izračun od prejšnjih hibridnih metod. Ugotovili smo, da naša metoda deluje boljše kot druge hibridne metode, vendar pa je to odvisno od besedila, ki ga procesiramo.

Tabela 3: Klasifikacija jezikov, prvi del

		Razpoznan razred (CBA, WBA, HA1, HA2, HA3)				
		bg-BG	bos-BOS	cs-CZ	hr-HR	Lt-sr-SP
Dejanski razred	bg-BG	25/25/25/25/25	0/0/0/0/0	0/0/0/0/0	0/0/0/0/0	0/0/0/0/0
	bos-BOS	5/0/5/0/0	20/1/20/1/1	0/0/0/0/0	0/17/0/17/13	0/6/0/6/10
	cs-CZ	0/0/0/0/0	0/1/0/0/0	12/22/12/23/25	0/0/0/0/0	0/0/0/0/0
	hr-HR	1/0/1/0/0	11/1/11/1/ 0	0/0/0/0/0	0/23/0/23/21	0/1/0/1/4
	Lt-sr-SP	1/0/1/0/0	20/1/20/1/0	0/0/0/0/1	0/20/0/20/14	0/4/0/4/10
	mk-MK	25/0/25/0/ 1	0/0/0/0/0	0/0/0/0/0	0/0/0/0/0	0/0/0/0/0
	pl-PL	0/0/0/0/0	0/0/0/0/0	0/0/0/0/0	0/0/0/0/0	0/0/0/0/0
	sk-SK	3/0/3/0/0	0/0/0/0/0	0/1/0/1/1	0/0/0/0/0	0/0/0/0/0
	sl-SI	0/0/0/0/0	7/0/7/0/0	0/1/0/1/1	0/2/0/2/0	0/0/0/0/0

Tabela 4: Klasifikacija jezikov, drugi del

		Razpoznan razred (CBA, WBA, HA1, HA2, HA3)				
		mk-MK	pl-PL	sk-SK	sl-SI	
Dejanski razred	bg-BG	0/0/0/0/0	0/0/0/0/0	0/0/0/0/0	0/0/0/0/0	
	bos-BOS	0/0/0/0/0	0/0/0/0/0	0/0/0/0/0	0/1/0/0/1	
	cs-CZ	0/0/0/0/0	0/1/0/0/0	11/1/11/2/0	2/0/2/0/ 0	
	hr-HR	0/0/0/0/0	0/0/0/0/2	0/0/0/0/0	13/0/13/0/0	
	Lt-sr-SP	0/0/0/0/0	0/0/0/0/9	0/0/0/0/0	4/0/4/0/0	
	mk-MK	0/25/0/25/24	0/0/0/0/0	0/0/0/0/0	0/0/0/0/0	
	pl-PL	0/0/0/0/0	18/25/18/25/25	7/0/7/0/0	0/0/0/0/0	
	sk-SK	0/0/0/0/0	0/4/0/1/0	22/20/22/23/24	0/0/0/0/0	
	sl-SI	0/0/0/0/0	0/0/0/0/0	0/0/0/0/0	18/22/18/22/24	

6. REFERENCE

[1] European parliament proceedings parallel corpus 1996-2011. `http://www.statmt.org/europarl/`. Accessed: 2016-05-23.

[2] Wikipedia. `https://www.wikipedia.org/`. Accessed: 2016-05-23.

[3] K. Abainia, S. Ouamour, and H. Sayoud. Robust language identification of noisy texts: Proposal of hybrid approaches. In *2014 25th International Workshop on Database and Expert Systems Applications*, pages 228–232, Sept 2014.

[4] W. B. Cavnar and J. M. Trenkle. N-gram-based text categorization. In *In Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175, 1994.

[5] R. M. Milne, R. A. O'Keefe, and A. Trotman. A study in language identification. In *Proceedings of the Seventeenth Australasian Document Computing Symposium*, ADCS '12, pages 88–95, New York, NY, USA, 2012. ACM.

[6] A. Selamat, N. C. Ching, and Y. Mikami. Arabic script web documents language identification using decision tree-artmap model. In *Convergence Information Technology, 2007. International Conference on*, pages 721–726, Nov 2007.

[7] A. K. Singh. Study of some distance measures for language and encoding identification. In *Proceedings of the Workshop on Linguistic Distances*, LD '06, pages 63–72, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

[8] A. Xafopoulos, C. Kotropoulos, G. Almpanidis, and I. Pitas. Language identification in web documents using discrete {HMMs}. *Pattern Recognition*, 37(3):583 – 594, 2004.

Stiskanje slik z algoritmi po vzorih iz narave

Gregor Jurgec
Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko
Smetanova 17, Maribor
gregor.jurgec@gmail.com

Iztok Fister
Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko
Smetanova 17, Maribor
iztok.fister@um.si

POVZETEK

V članku predstavljamo postopek stiskanja slik na inovativen način s pomočjo optimizacijskih algoritmov iz dveh različnih družin, tj. evolucijskih algoritmov in algoritmov na osnovi inteligence rojev. V naši raziskavi za reševanje problema stiskanja slik uporabljamo diferencialno evolucijo in optimizacijo s kukavičjim iskanjem. Osnovna algoritma izboljšamo z adaptacijskimi in hibridizacijskimi metodami. Stiskanje slik izvedemo s pomočjo množice primitivnih geometrijskih oblik trikotnikov. Razvita algoritma testiramo na realnih primerih stiskanja slik in kakovost rezultatov stiskanja primerjamo med seboj. Z izboljšanima algoritmoma dobimo zelo dobre rezultate stisnjenih slik, saj podobnost teh slik v primerjavi z izvornimi slikami v najslabšem primeru presega 89,70 %, v najboljšem primeru pa dosežemo 92,08 %.

Ključne besede

evolucijski algoritmi, inteligenca rojev, diferencialna evolucija, kukavičje iskanje, stiskanje slik

1. UVOD

Z razvojem naprednih senzorjev za zajem slik visoke ločljivosti se povečuje velikost zajetih posnetkov. Pri tem pridemo do problema shranjevanja slik, saj nestisnjene slike zavzemajo ogromno prostora na spominskih medijih. Da prihranimo prostor, uporabljamo različne optimizacijske algoritme, ki stisnejo podatke na račun izgube informacij o sliki. Za te izgube seveda želimo, da so čim manjše. Algoritmi stiskanja slik iščejo podobnosti v podatkih, oz. skušajo podatke, zajete s senzorji, spraviti v čim kompaktnejšo obliko.

Glavna razlika med evolucijskimi algoritmi in algoritmi na osnovi inteligence rojev je, da evolucijski algoritmi [4] pri optimizaciji simulirajo naravno evolucijo z operatorji mutacije, križanja in selekcije [3], medtem ko delovanje algoritmov na osnovi inteligence rojev temelji na obnašanju bioloških vrst (npr. živali ali žuželk), ki rešujejo vsakodnevne probleme porazdeljeno in kolektivno [1]. S kombinacijo značilnosti obeh družin lahko razvijemo hibridne algoritme, ki so sposobni reševanja tudi kompleksnejših optimizacijskih problemov. S hibridizacijo različnih delov algoritma dodamo osnovnemu algoritmu domensko specifično znanje domene [4, 5].

V tem članku rešujemo problem stiskanja slik z izboljšanima optimizacijskima algoritmoma diferencialne evolucije [10] in kukavičjega iskanja [11] sposobna generiranja slik, ki zasedajo manj prostora, ampak slabše kakovosti. Sliko predstavljamo s trikotniki narisanimi na platno, ki so določeni s koordinatami oglišč in barvo. Trikotniki so učinkoviti za to nalogo saj pokrivajo več slikovnih pik kot ena sama točka in omogočajo prekrivanje in prosojnost barv v določeni regiji. Naš cilj je, da bi bila izrisana slika čim bolj podobna originalni sliki. V tem članku se zgledujemo in nadgrajujemo delo iz [13, 7], kjer s pomočjo trikotnikov in samo adaptivne diferencialne evolucije oziroma genetskega algoritma poskušajo zgraditi približen model slike.

S tem delom želimo pokazati, da lahko z izboljšanima algoritmoma na osnovi obnašanja kukavic in diferencialne evolucije uspešno stisnemo izvorno sliko. Pri tem lahko ugotovimo, da sta oba algoritma primerna za stiskanje slik s trikotniki, tj. na inovativen način in s pomočjo novih algoritmov, po vzoru iz narave. Tak način stiskanja slik lahko postane dobra alternativa obstoječim algoritmom stiskanja slik. Seveda ti algoritmi potrebujejo še dodatne izboljšave, vendar so se kot osnova izkazali zelo dobro, kar dokazujejo tudi rezultati v nadaljevanju.

Struktura članka v nadaljevanju je naslednja. Drugo poglavje se ukvarja s podanimi splošnimi informacijami o kodiranju slik. V tretjem in četrtem poglavju predstavimo oba izboljšana algoritma za stiskanje slik. Peto poglavje primerja rezultate stiskanja realnih slik z izboljšanima algoritmoma.

2. KODIRANJE SLIKE

V našem delu uporabljamo izgubni način stiskanja slik, ker imajo naše stisnjene slike nepravilnosti in so popačene. Naša slika je predstavljena z množico barvnih trikotnikov, narisanih na črno podlago, ki je iste velikosti kot originalna slika. Množica trikotnikov predstavlja sliko, podobno originalni. Vsak trikotnik je sestavljen iz treh točk s koordinatami x, y in barvo, s katero je obarvan, v barvnem prostoru rdeče-zeleno-modro-alfa (angl. Red, Green, Blue, Alpha, krajše RGBA). Zelo pomembno je, da uporabimo prosojnost tri-

kotnikov, s čimer dosežemo prekrivanje trikotnikov in s tem dobimo veliko različnih barv, ki sestavljajo sliko.

Množico barvnih trikotnikov, ki sestavljajo sliko, kodiramo v vektor na naslednji način: vektor $\vec{x} = (\vec{T}_1, \vec{T}_2, \vec{T}_3, ..., \vec{T}_{\max})$ predstavlja množico trikotnikov $\vec{T}_i, \forall i \in \{1, ..., \max\}$, kjer max določa število trikotnikov, ki sestavljajo sliko. Vsak od teh trikotnikov je kodiran v vektor $\vec{T}_i = (x_1, y_1, x_2, y_2, x_3, y_3, R, G, B, A)$, ki je sestavljen iz celih števil, kjer pari $x_i, y_i, \forall i \in \{1, 2, 3\}$ predstavljajo koordinatne točke oglišč trikotnika, skalarji R, G, B barvo in A prosojnost trikotnika.

Slika 1 prikazuje množico prekrivajočih barvnih trikotnikov na črnem ozadju v zgodnji fazi delovanja optimizacijskih algoritmov.

Slika 1: Osnovni prikaz slike s trikotniki.

2.1 Optimizacijski problem in kriterijska funkcija

Optimizacija je postopek iskanja najboljših vhodnih parametrov pri znanem modelu in znanih izhodnih parametrih [6, 4]. Optimizacijski problem P definiramo kot četvorko

$$P = \langle I, S, f, cilj \rangle,$$

kjer pomenijo: I množico vseh nalog, S množico dopustnih rešitev problema, f kriterijsko funkcijo (angl. objective function), s katero ocenjujemo kakovost rešitve, in *cilj*, s katerim povemo ali kriterijsko funkcijo maksimiziramo ali minimiziramo. Vrednosti kriterijske funkcije so lahko skalarji ali v primeru več-kriterijskih problemov vektorji [9, 12].

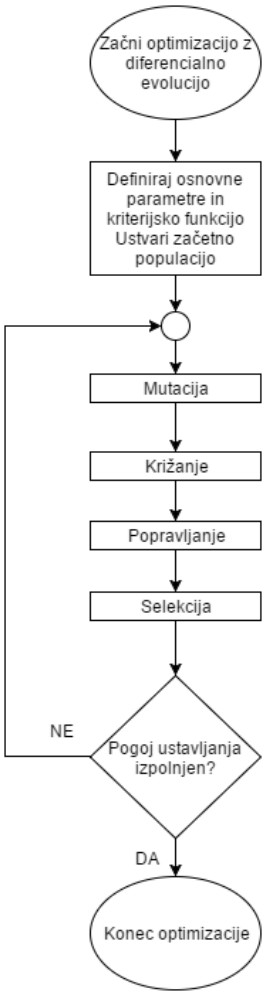
Kriterijska funkcija v našem primeru primerja vsako isto ležečo slikovno piko generirane slike s slikovno piko na originalni sliki. Vsako slikovno piko primerjamo po vseh treh barvnih komponentah R, G, B. Vsoto razlik isto ležečih slikovnih pik pri vsaki od treh komponent delimo s produktom med višino in širino slike ter z vsemi možnimi odtenki vsake od barv RGB. Z rezultatom dobimo vrednost različnosti slik, ki ga odštejemo od ena in dobimo podobnost med slikama. To pomnožimo s 100 % in ta vrednost potem predstavlja podobnost najdene slike v primerjavi z originalno v odstotkih. Naš cilj je, da je podobnost čim večja in zato moramo našo kriterijsko funkcijo maksimizirati.

Naša kriterijska funkcija $f(\vec{x})$ je opisana z enačbo 1, kjer $\vec{x}^*$ predstavlja originalno, referenčno sliko, $\vec{x}$ pa stisnjeno, generirano sliko. Indeksa $x_i, \forall i \in \{1, ...višinaSlike\}$ in $y_j, \forall j \in \{1, ...širinaSlike\}$ določata trenutni položaj opazovane slikovne pike. Oznake R, G in B pa definirajo, v kateri komponenti barve RGB se nahajamo. Konstanta 255 določa število različnih odtenkov barve.

$$f(\vec{x}) = 100\% \cdot \left(1 - \left(\frac{\sum_{i=1}^{višinaSlike} \sum_{j=1}^{širinaSlike} |\vec{x}^*_{i,j}{}^R - \vec{x}_{i,j}{}^R|}{255 \cdot višinaSlike \cdot širinaSlike} + \frac{\sum_{i=1}^{višinaSlike} \sum_{j=1}^{širinaSlike} |\vec{x}^*_{i,j}{}^G - \vec{x}_{i,j}{}^G|}{255 \cdot višinaSlike \cdot širinaSlike} + \frac{\sum_{i=1}^{višinaSlike} \sum_{j=1}^{širinaSlike} |\vec{x}^*_{i,j}{}^B - \vec{x}_{i,j}{}^B|}{255 \cdot višinaSlike \cdot širinaSlike} \right) \right) \quad (1)$$

3. DIFERENCIALNA EVOLUCIJA

Diferencialna evolucija (angl. Differential Evolution, krajše DE) rešuje optimizacijski problem z izboljševanjem posameznih rešitev iz populacije. Iz obstoječih rešitev tvori poskusne rešitve (angl. candidate solution) z dodajanjem razlike naključno izbranih vektorjev (mutacijska strategija), te pa je potrebno vrednotiti. Vrednosti elementov vektorjev poskusne rešitve, ki so izven definiranih območij popravimo na zgornje ali spodnje meje v operaciji popravljanja. Če je poskusna rešitev boljša od obstoječe, ta rešitev zamenja obstoječo v populaciji. Ta postopek ponavljamo za vsakega posameznika v populaciji rešitev v vsaki generaciji algoritma, dokler ni izpolnjen pogoj zaustavljanja [10]. Model diferencialne evolucije je prikazan na sliki 2.



Slika 2: Model diferencialne evolucije

Psevdo-kod originalnega algorithma DE prikazuje algoritem 1.

Algoritem 1 Algoritem DE

```
inicializacija;
najboljsaVrednost^0 ← oceniVseResitve;
while pogojUstavljanjaIzpolnjen do
    for i ← 1, N_p do
        x_i^p ← generirajPoskusnoResitev(x_i^t);
        poskusnaVrednost ← oceniPoskusnoResitev(x_i^p);
        if poskusnaVrednost > f(x_i^t) then
            x_i^t ← x_i^p;
            najboljsaVrednost_i^t ← poskusnaVrednost;
        end if
    end for
end while
```

3.1 Samoprilagajanje nadzornih parametrov v diferencialni evoluciji (jDE)

V literaturi najdemo veliko različnih rešitev, kako nastaviti nadzorna parametra F in C_r. Izbira primernih vrednosti nadzornih parametrov je odvisna od problema, ki ga rešujemo. Da se izognemo temu, so avtorji Brest in ostali v [2] predlagali izboljšano verzijo algoritma DE, ki nadzorna parametra F in C_r samo-prilagaja med samim izvajanjem. Poimenovali so ga algoritem jDE. Vsak posameznik v populaciji je razširjen z vrednostjo omenjenih parametrov, ki se prilagajata in izboljšujeta skozi evolucijo in posledično izboljšata posameznike v poskusni populaciji. Nova nadzorna parametra F_i^p in ${C_r}_i^p$ izračunamo pred mutacijo, tako da imata vpliv na operatorje DE in izračun poskusne rešitve v trenutni generaciji, po enačbah:

$$F_i^p = \begin{cases} F_l + U_1(0,1) \cdot F_u & \check{c}e\ U_2(0,1) < \tau_1, \\ F_i^G & drugače, \end{cases}$$

$${C_r}_i^p = \begin{cases} U_3(0,1) & \check{c}e\ U_4(0,1) < \tau_2, \\ {C_r}_i^t & drugače, \end{cases}$$

kjer je $U_j(0,1)$, $j \in \{1,2,3,4\}$ naključna uniformna vrednost iz intervala [0,1] ter τ_1 in τ_2 verjetnosti za spremembo parametrov F in C_r (tudi hitrosti učenja). Avtorji zanju predlagajo vrednosti $\tau_1 = \tau_2 = 0,1$. Vrednosti $F_l = 0,1$ in $F_u = 0,9$ zagotavljata, da je novi parameter F vedno naključno v intervalu [0.1, 1.0]. Novi parameter C_r pa zavzema vrednosti v intervalu [0,1] [13].

3.2 Modificirana diferencialna evolucija

Modificirana verzija diferencialne evolucije (krajše MoDE) za stiskanje slik vsebuje več prilagoditev in izboljšav. Najprej prilagodimo osnovni algoritem na obstoječ algoritem jDE, ki je opisan v prejšnjem poglavju. Nato razširimo jDE algoritem s strategijo $'DE/best/1/bin'$, s katero dosežemo, da se posamezniki približujejo najboljši rešitvi, okrog katere bi lahko bil globalni maksimum. V primeru, da pride do izgube raznolikosti v populaciji, uporabimo ponovno inicializacijo večine posameznikov, s čimer raznolikost populacije povečamo.

3.2.1 Kombiniranje mutacijskih strategij 'DE/rand/1/bin' in 'DE/best/1/bin'

Pri tej izboljšavi gre za izbiro mutacijske strategije. Kombiniramo med strategijo $'DE/rand/1/bin'$ in $'DE/best/1/bin'$. Privzeto uporabljamo strategijo $'DE/rand/1/bin'$. Verjetnost izbire strategije $'DE/best/1/bin'$ v vsaki generaciji pri vsakem posamezniku v populaciji je 10^{-5}, ki smo jo določili po občutku skozi testiranja. Izbiro strategije in s tem baznega vektorja uvedemo pred začetkom mutacije.

$$IzbiraStrategije = \begin{cases} 'DE/best/1/bin' & \check{c}e\ U(0,1) < 10^{-5}, \\ 'DE/rand/1/bin' & drugače. \end{cases}$$

V zgornji enačbi predstavlja $U(0,1)$ naključno število iz intervala [0,1] iz uniformne porazdelitve. Vektor mutacije $\vec{m}_i$ pri strategiji $'DE/best/1/bin'$ izračunamo po enačbi:

$$\vec{m}_i = \vec{x}_{best}^t + F \cdot \left(\vec{x}_{r1}^t - \vec{x}_{r2}^t\right),$$

kjer $\vec{x}_{best}^t$ označuje najboljšega posameznika v trenutni populaciji, F je krmilni parameter, $\vec{x}_{r1}^t$ in $\vec{x}_{r2}^t$ pa sta naključno izbrana vektorja iz trenutne populacije, kjer $r1 \neq r2$. Vektor mutacije $\vec{m}_i$ pri privzeti strategiji $'DE/rand/1/bin'$ izračunamo po enačbi:

$$\vec{m}_i = \vec{x}_{r1}^t + F \cdot \left(\vec{x}_{r2}^t - \vec{x}_{r3}^t\right).$$

3.2.2 Ponastavitev posameznika

Da se izognemo lokalnemu optimumu, uporabimo mehanizem ponastavitve posameznika (angl. random restart) z naključnimi vrednostmi elementov pri operaciji križanja. Križanje je prikazano z naslednjo enačbo:

$$k_{i,j} = \begin{cases} x_{\min,j} + \left(x_{\max,j} - x_{\min,j}\right) \cdot U(0,1) & \check{c}e\ U_j(0,1) < C_r \\ & \vee\ j_{rand} = j \wedge U_j(0,1) < 10^{-5}, \\ m_{i,j} & \check{c}e\ U_j(0,1) < C_r \vee j_{rand} = j, \\ x_{i,j}^t & drugače. \end{cases}$$

Če je pri križanju v vsaki iteraciji dimenzije problema slučajno število $U(0,1)$ pod pragom 10^{-5}, takrat celoten vektor $\vec{k}_i$ inicializiramo na začetne vrednosti in prekinemo križanje, drugače pa na pozicijo, ki jo označuje indeks j, v vsaki iteraciji kopiramo vrednost vektorja mutacije $\vec{m}_i$ ali vrednost trenutnega posameznika $\vec{x}_i^t$ z istim indeksom.

3.2.3 Razpršenost populacije in ponastavitev posameznikov

Po selekciji izračunamo razpršenost populacije. Razpršenost populacije služi kot merilo raznolikosti populacije. Izračunamo jo s standardnim odklonom od povprečne vrednosti kriterijske funkcije posameznikov v populaciji, ki ga izračunamo po enačbi:

$$povprečje = \frac{\sum_{i=1}^{N_p} fitnes_i}{N_p},$$

kjer seštejemo vse vrednosti kriterijske funkcije posameznikov v populaciji in jih delimo s številom posameznikov N_p. Povprečje uporabimo pri izračunu standardnega odklona v enačbi:

$$stdOdklon = \sqrt{\frac{\sum_{i=1}^{N_p} \left(povprečje - fitnes_i\right)^2}{N_p}}.$$

Če je standardni odklon oziroma razpršenost populacije manjša od praga, ki ga določimo pri testiranju, ponastavimo 90 % najslabših posameznikov na začetne vrednosti in s tem povečamo raznolikost populacije.

4. KUKAVIČJE ISKANJE

V nadaljevanju predstavljamo osnovni algoritem s kukavičjim iskanjem [11] (angl. Cuckoo Search, krajše CS), kjer je vhod v algoritem začetna populacija, sestavljena iz posameznikov inicializiranih z naključnimi vrednostmi elementov, ki predstavljajo rešitve. Izhod algoritma je najboljša rešitev oz. najboljši posameznik glede na kriterijsko funkcijo. Osnovni algoritem vsebuje inicializacijo in glavno zanko, ki teče do pogoja ustavljanja. Za vsakega posameznika iz populacije najprej generiramo novo rešitev z distribucijo Levy in jo ocenimo s kriterijsko funkcijo. Iz populacije izberemo naključnega posameznika, njegovo vrednost kriterijske funkcije primerjamo z vrednostjo kriterijske funkcije potencialne nove rešitve in tega zamenjamo z novo rešitvijo v primeru, da je ta boljša. Sledi zamenjava najslabšega posameznika s posameznikom, ki ga dobimo z naključno inicializacijo posameznika v primeru, da je naključno generirano število iz intervala $[0, 1] < p_\alpha$. Na koncu vsake iteracije poiščemo in shranimo najboljšo rešitev v populaciji v globalno rešitev, ki je izhod algoritma. Model algoritma s kukavičjim iskanjem je prikazan na sliki 3.

Psevdo-kod originalnega algorithma CS prikazuje algoritem 2.

Algoritem 2 Algoritem CS

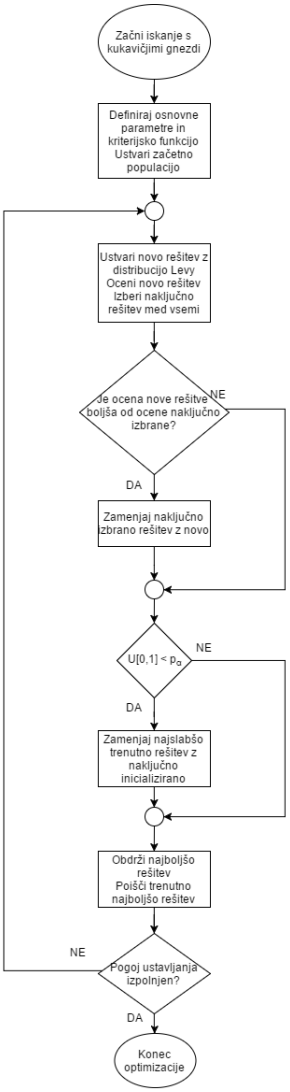
```
inicializacija;
najboljsaVrednost^0 ← oceniVseResitve;
while pogojUstavljanjaIzpolnjen do
    for i ← 1, N_p do
        x_i^p ← generirajPoskusnoResitev(x_i^t);
        poskusnaVrednost ← oceniPoskusnoResitev(x_i^p);
        j_rand ← U(1, N_p);
        if poskusnaVrednost > najboljsaVrednost^t_{j_rand} then
            x^t_{j_rand} ← x_i^p;
            najboljsaVrednost^t_{j_rand} ← poskusnaVrednost;
        end if
        if U(0,1) < p_α then
            inicializacija(x^t_{najslabsi});
        end if
        shraniGlobalnoNajboljsoResitev;
    end for
end while
```

4.1 Modificirano kukavičje iskanje

Modificirana verzija CS (krajše MoCS) vsebuje dve glavni izboljšavi. Najprej razširimo algoritem CS z generiranjem N_p kukavičjih jajc v eni iteraciji, nato uvedemo hibridizacijo z mutacijsko strategijo DE.

4.1.1 Razširitev algoritma CS

Najprej generiramo N_p novih rešitev (kukavičja gnezda) in jih ocenimo s kriterijsko funkcijo. Vrednosti kriterijske funkcije novih rešitev primerjamo z vrednostmi kriterijske funkcije rešitev iz trenutne populacije na vsakem indeksu od 1 do N_p in nove rešitve zamenjamo z rešitvami iz trenutne populacije v primeru, da so te boljše. Razlika z osnovnim



Slika 3: Model algoritma s kukavičjim iskanjem

algoritmom je v tem, da generiramo N_p novih rešitev in ne izbiramo naključne rešitve za primerjavo z novo rešitvijo, ampak primerjamo kar rešitve z istim indeksom.

4.1.2 Hibridizacija z mutacijsko strategijo DE

Odkrite posameznike modificiramo po vzoru mutacijske strategije DE [8]. Tu gre za neke vrste pristransko naključno iskanje, kjer razliko naključno izbranih posameznikov prištejemo rešitvi posameznika, ki ga trenutno zavračamo. Naključne posameznike izberemo tako, da generiramo dve polji permutacij dolžine N_p, nato izračunamo razliko vektorjev z istim indeksom in vsakokrat dobimo drugo razliko. Modificirano naključno iskanje izrazimo z naslednjo enačbo:

$$\vec{x}_i^{t+1} = \vec{x}_i^t + U(0,1) \times H(p_\alpha - \varepsilon) \times \left(\vec{x}_j^t - \vec{x}_k^t\right),$$

kjer sta $\vec{x}_j^t$ in $\vec{x}_k^t$ naključno izbrana posameznika iz populacije, $U(0,1)$ funkcija, ki vrne naključno realno število z uniformno porazdelitvijo iz intervala [0,1], $H(p_\alpha - \varepsilon)$ je funkcija Heaviside, s katero odločimo, ali zavržemo ali obdržimo novega posameznika, katere rezultat je 0, če je $p_\alpha - \varepsilon < 0$

in 1, če je $p_\alpha - \varepsilon \geq 0$, p_α je parameter zamenjave, ε naključno število iz uniformne distribucije iz intervala [0,1] in $\vec{x}_i^t$ je izbrani vektor iz populacije.

5. REZULTATI

Testiranje algoritmov smo izvedli v dveh korakih. V prvem koraku smo poiskali najboljše parametre velikosti populacije in števila trikotnikov med izbranimi vrednostmi za stiskanje vsake izmed izbranih slik pri 10^5 vrednotenjih. Ko smo najboljše parametre našli, smo stiskanje s temi parametri ponovili, le da smo v drugem koraku povečali število vrednotenj na 10^7 in tako dobili najboljše rezultate z izbranimi vrednostmi. Rezultate stiskanja slik predstavljamo v nadaljevanju.

5.1 Izbira slik in iskanje parametrov

Najprej smo določili tri referenčne, originalne slike, ki smo jih stiskali z našima algoritmoma. Zaradi zahtevnosti kriterijske funkcije smo vsako od slik pomanjšali na resolucijo 100x100. Raznolikost referenčnih slik, prikazanih na sliki 4, smo dosegli z izbiro dveh barvnih in ene sivinske slike.

Slika 4: Izbrane slike za stiskanje.

Izbira optimalnih parametrov algoritmov je zelo pomembna, saj ti vplivajo na pravilno delovanje algoritma, s tem pa na kakovost končnih slik in hitrost konvergence. Prav zaradi pomembnosti parametrov algoritmov smo najprej poiskali najboljše parametre vsakemu od algoritmov za stiskanje vsake slike posebej pri maksimalno 10^5 vrednotenjih kriterijske funkcije. Poudariti moramo, da smo kot ustavitveni pogoj v algoritmih vedno uporabili število vrednotenj kriterijske funkcije, s čimer smo zagotovili poštenost meritev in rezultatov. Naslednja dva parametra sta bila še velikost populacije, ki smo jo izbirali iz množice $N_p = \{50, 100, 150, 200\}$, in število trikotnikov, uporabljenih pri stiskanju slik. Te smo izbirali s pomočjo porazdelitve eksponentne funkcije:

$$stTrikotnikov = 20 \times 2^x, \quad \text{za } x = \{0, 1, 2, 3, 4\},$$

kjer smo dobili naslednja dopustna števila trikotnikov:

$$stTrikotnikov = \{20, 40, 80, 160, 320\}.$$

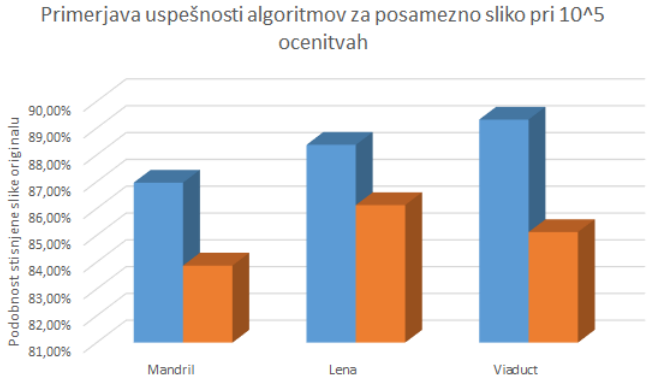
Tabela 1: Izbrani najboljši parametri

Ime slike	MoDE		MoCS	
	N_p	N_t	N_p	N_t
Mandril	50	20	50	80
Lena	50	20	50	20
Viaduct	50	80	50	40

Pri vsakem zagonu smo kombinirali vse možne kombinacije parametrov velikosti populacije in števila trikotnikov, s čimer smo izvedli 20 neodvisnih zagonov algoritma za eno sliko. Za vsako sliko smo postopek ponovili desetkrat, tj. 200 neodvisnih zagonov. Ker imamo dva algoritma in tri slike, se je skupaj izvedlo kar 1200 neodvisnih zagonov algoritma.

Vsak algoritem krmilimo z nadzornimi parametri. Pri DE smo izbrali začetne vrednosti za $F = 0,5$ in $C_r = 0,9$. Sama nastavitev teh parametrov ne igra posebne vloge, saj ima naš algoritem vključeno samo-prilagajanje teh parametrov. Nekoliko drugače je pri CS, saj moramo nastaviti parameter zamenjave p_α, ki je fiksen skozi celoten algoritem in smo ga v našem primeru nastavili na 0,25. Kot najboljše nastavljive parametre (tabela 1) smo izbrali tiste, ki so imeli največjo povprečno vrednost kriterijske funkcije in ne morebitne posamezne najboljše rešitve z največjo vrednostjo kriterijske funkcije.

Rezultati prvega dela testiranja, ko smo iskali najboljše parametre za algoritma, so pokazali dobre rezultate, ki jih lahko še izboljšamo s povečanjem števila vrednotenj kriterijske funkcije. Slika 5 prikazuje, kako uspešna sta bili naša algoritma pri stiskanju slik pri iskanju najboljših nastavljivih parametrih do 10^5 vrednotenj kriterijske funkcije.



Slika 5: Podobnost rezultatov algoritmov MoDE in MoCS za stiskanje slik pri 10^5 vrednotenjih.

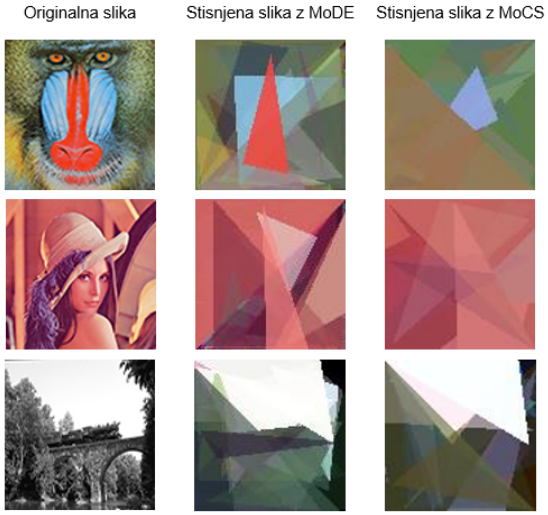
Iz slike 5 lahko razberemo, da so rezultati MoDE nekoliko boljši v primerjavi z MoCS za vsako sliko pri 10^5 vrednotenjih. Za sliko Mandril je MoDE dosegel za 3,09 % večjo podobnost originalni sliki v primerjavi z MoCS, za sliko Lena 2,25 % in za sliko Viaduct 4,20 %.

Slika 6 prikazuje originalne in najboljše stisnjene slike iz prvega koraka testiranja z algoritmoma MoDE in MoCS. Iz slik je razvidno, da v tej fazi testiranja stiskanja slik pri 10^5

Tabela 2: Rezultati podobnosti stisnjene slike in čas stiskanja z 10^7 vrednotenji.

Ime slike	MoDE		MoCS	
	Podobnost	Čas stiskanja	Podobnost	Čas stiskanja
Mandril	90,08 %	4h 41m 11s	89,70 %	8h 12m 45s
Lena	91,92 %	4h 48m 35s	90,31 %	5h 43m 38s
Viaduct	92,08 %	5h 51m 31s	90,56 %	6h 26m 24s

vrednotenjih MoCS ne dosega tako dobrih rezultatov kot MoDE.



Slika 6: Primerjava originalnih slik s stisnjenimi slikami z algoritmoma MoDE in MoCS z 10^5 vrednotenji kriterijske funkcije.

5.2 Stiskanje slik z 10^7 vrednotenji kriterijske funkcije

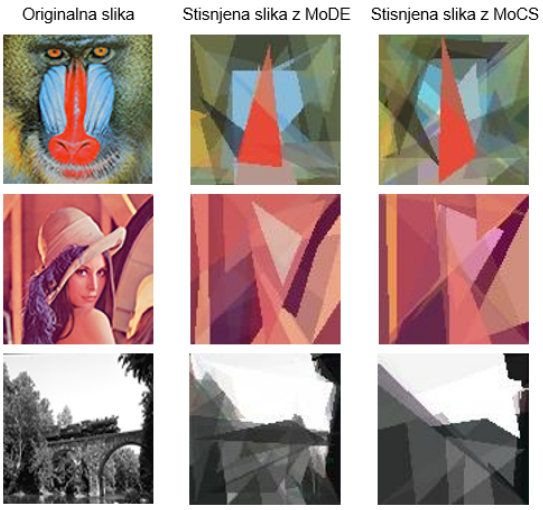
Po izvedbi testiranja, s katerim smo dobili najboljše nastavitve parametrov, smo te uporabili pri časovno zahtevnejšem testiranju stiskanja slik. Število vrednotenj kriterijske funkcije smo povečali iz 10^5 na 10^7 in ustrezno nastavili najboljše parametre velikosti populacije N_p in število trikotnikov N_T, kot jih prikazuje tabela 1.

Rezultati, prikazani v tabeli 2, so bili veliko boljši kot pri stiskanju z 10^5 vrednotenji. Algoritem MoDE je tudi pri tem številu vrednotenj boljše stisnil vsako izmed slik, kot je to storil algoritem MoCS.

Iz slike 7 je jasno razvidno, da je razvit algoritem MoDE boljši algoritem za stiskanje slik kot razvit algoritem MoCS. V stisnjenih slikah z algoritmom MoDE so jasno vidne linije objektov in njihove barve. Pri stisnjenih slikah z algoritmom MoCS so te linije nekoliko zabrisane in v primerjavi z MoDE so nekateri objekti združeni v enega samega.

6. ZAKLJUčEK

Razvili smo dva algoritma za stiskanje slik po vzorih iz narave. Podrobno smo opisali nekatere od možnih izboljšav algoritma DE in CS, ki pomagajo pri konvergenci algoritma. Rezultati izboljšanih algoritmov dokazujejo, da ti algoritmi



Slika 7: Primerjava originalnih slik s stisnjenimi slikami z algoritmoma MoDE in MoCS z 10^7 vrednotenji kriterijske funkcije.

zaenkrat še niso primerni za naš problem stiskanja slik, saj ne dajejo rezultatov, ki bi bili sprejemljivi v praksi. Pričakujemo lahko, da bosta omenjena algoritma na podlagi dodatnih izboljšav postala primerna tudi za te namene. Trenutno je stiskanje s temi algoritmi primerno za znanstvene namene in poskuse, ker lahko pokažemo, da so ti algoritmi po vzorih iz narave splošno namenski.

Rezultati stisnjenih slik so po oceni naše kriterijske funkcije glede podobnosti z originalno sliko zelo dobri. Dosegajo podobnost do dobrih 92 %, kar se zdi zelo obetavno. Ko pogledamo stisnjene slike, pa lahko opazimo, da so na slikah vidni le robovi slike in da so barve poligonov, ki jih ti robovi omejujejo, podobne istemu področju originalne slike. Iz samih stisnjenih slik lahko brez težav razberemo, kateri originalni sliki pripada, same objekte pa je že težje prepoznati.

6.1 Možnosti za izboljšave, nadgradnje

Obstaja več možnosti, s katerimi bi lahko uspešnost naših algoritmov še dodatno izboljšali. V naslednjih nekaj odstavkih predstavljamo samo nekatere izmed njih.

Lotili bi se lahko pred-procesiranja slik, kjer slike razdelimo na več pod-slik in vsako od njih obdelamo neodvisno z algoritmom stiskanja. Glede na to, da smo pri sivinskih slikah dobili zelo dobre rezultate, bi lahko bila naslednja izboljšava po vzoru sivinskih slik. Tukaj sliko razdelimo na tri barvne ravnine RGB, vsako od njih obdelamo posebej in na koncu združimo rezultate nazaj v barvno sliko.

Algoritmi po vzorih iz narave se nenehno izpopolnjujejo in izboljšujejo s strani različnih raziskovalcev. Prav gotovo obstajajo še kakšne izboljšave uporabljenih algoritmov v našem delu, ki bi jih lahko prilagodili za naš problem stiskanja slik in tako izpopolnili uspešnost uporabljenih algoritmov. Ena od izboljšav pri obeh algoritmih bi bila uporabiti dodatne mehanizme za izogibanje lokalnemu optimumu.

Velik problem pri stiskanju slik z algoritmi po vzorih iz narave je njihova časovna zahtevnost. Kot smo prikazali v članku, je ugotavljanje dobrih parametrov, testiranje in stiskanje potekalo več dni. Če ne bi bili časovno omejeni, bi bilo zanimivo narediti več zagonov algoritmov nad skupino najboljših parametrov in bi tako lahko dobili drugačne, boljše parametre za stiskanje slik.

Algoritem bi lahko nadgradili tako, da bi lahko slike stiskali v drugem barvnem modelu, kot sta recimo HSV ali CMYK.

7. REFERENCES

[1] C. Blum and D. Merkle. *Swarm Intelligence*. Springer-Verlag, Berlin, 2008.

[2] J. Brest, S. Greiner, B. Bošković, M. Mernik, and V. Žumer. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *T. Evolut. Comput.*, 10(6):646–657, 2006.

[3] C. Darwin. *On the Origin of Species*. Harvard University Press, London, 1964.

[4] A. Eiben and J. Smith. *Introduction to Evolutionary Computing*. Springer-Verlag, Berlin, 2003.

[5] I. Fister, M. Mernik, and J. Brest. *Hybridization of Evolutionary Algorithms, Evolutionary Algorithms*. InTech, 2011.

[6] I. Fister, D. Strnad, X.-S. Yang, and I. Fister Jr. Adaptation and hybridization in nature-inspired algorithms. In *Adaptation and Hybridization in Computational Intelligence*, pages 3–50, Berlin, 2015. Springer-Verlag.

[7] A. Izadi, V. Ciesielski, and M. Berry. Evolutionary non photo-realistic animations with triangular brushstrokes. In *AI 2010: Advances in artificial intelligence*, pages 283–292, Berlin, 2011. Springer-Verlag.

[8] U. Mlakar, I. F. Jr., and I. Fister. Hybrid self-adaptive cuckoo search for global optimization. *Swarm and Evolutionary Computation*, pages –, 2016.

[9] S. Sivanandam and S. Deepa. *Introduction to genetic algorithms*. Springer-Verlag, Heidelberg, 2008.

[10] R. Storn and K. Price. Differential evolution: A simple and efficient heuristic for global optimization over continuous spaces. *J. Global Optim.*, 11(4):341–359, 1997.

[11] X.-S. Yang and S. Deb. Cuckoo search via levy flights. In *World Congress & Biologically Inspired Computing (NaBIC 2009)*, pages 210–214, IEEE Publication, 2009.

[12] A. Zalzala and P. Fleming. *Genetic algorithms in engineering systems*. The institution of electrical engineers, London, UK, 1997.

[13] A. Zamuda and U. Mlakar. Differential evolution control parameters study for self-adaptive triangular brushstrokes. *Informatica*, 39(2):105–113, 2015.

A GPGPU Implementation of CMA-ES

Aleksandar Tošić
Faculty of Mathematics, Natural Sciences and Information Technologies
University of Primorska
aleksandar.tosic@upr.si

Matjaž Šuber
Faculty of Mathematics, Natural Sciences and Information Technologies
University of Primorska
matjaz.suber@student.upr.si

ABSTRACT

In the last decade, a lot of hope has been put into General Purpose computing on Graphical Processing Units with optimization algorithms being among the most researched [10, 6, 12, 13]. Unfortunately, the shift in programming paradigms and computation models makes implementation of existing algorithms non-trivial. Some algorithms are simply not suitable for implementation on a single Instruction Multiple Data (SIMD) model. To overcome the tediousness of programming GPU's a number of frameworks became available with Compute Unified Device Architecture (CUDA) [9] being the most popular proprietary and Open Computing Language (OpenCL) [8] leading in the field of open-source frameworks. With the help of OpenCL we implemented the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [4] in a Metaheuristic Algorithms in Java (jMetal) framework [1, 2]. We tested both CPU and OpenCL implementations on different hardware configurations and problem sets available in jMetal. Additionally we explain the details of our implementation and show a comparison of computation results.

Keywords

Covariance Matrix Adaptation Evolution Strategy(CMA), Numerical Optimisation, Evolutionary Algorithm, GPGPU, jMetal, OpenCL

1. INTRODUCTION

Computation on graphics processing units has received a lot of attention in the past. Mainly because of the availability of hardware that was driven mostly by the gaming industry [7]. As more complex rendering engines were developed, the need to harness the GPU capabilities grew. Manufacturers gave more access and control over their hardware trough frameworks.

In this paper we present an implementation of a popular evolutionary inspired algorithm called CMA-ES. The CMA-ES (Covariance Matrix Adaptation Evolution Strategy) is an evolutionary algorithm for non-linear or non-convex continuous optimization problems. It is considered as state-of-the-art in evolutionary computation and has been adopted as one of the standard tools for continuous optimization. It is based on the principle of biological evolution, where in each iteration new candidate solutions are generated.

Choosing the framework was relatively straight forward. Most frameworks are not mature enough or proprietary, hence we chose openCl. OpenCL (Open Computing Language) is the first truly open and royalty-free programming standard for general-purpose computations on heterogeneous systems. It allows programmers to preserve their expensive source code investment and easily target multi-core CPUs, GPUs, and the new APUs. It improves the speed and responsiveness of a wide spectrum of applications. OpenCL is defined with four models, namely Platform model, Execution model, Memory model, and Programming model [3]. The most important beeing the execution model that deals with the loading and executing kernels on the GPU and Memory model, handling the allocation of GPU memory and the transfer of data between host memory and GPU memory.

The main challenge is to adapt the kernels and memory models depending on the hardware specifications namely, work-group sizes, compute units, etc. In this paper present empirical results comparing the standard algorithm with our GPGPU implementation. Additionally, we show results comparing two platforms from different hardware manufacturers to eliminate the potential bias of openCl implementation.

2. IMPLEMENTATION

The main loop in the CMA-ES algorithm consists of three main parts and it runs until termination criterion is met. In the first part the algorithm samples new population of solutions, for $k = 1, ..., \lambda$, in the second part it reorders the sampled solutions based on the fitness function and finally it updates the covariance matrix as described in [5].

The execution model defines how the OpenCL environment is configured on the host and how kernels are executed on the device. Before a host can request that a kernel is executed on a device, a context must be configured on the host that coordinates the host-device interaction and keeps track of the kernels that are created for each device. Each kernel contain a unit of code (function) which is executed on a

specific compute unit. This units are then grouped into one, two or three dimensional set, which are called work-groups.

For the implementation we obtained the source code of the iterative CMA-ES algorithm from the jMetal Java based framework for multi-objective optimization [1]. To speed-up the algorithm we integrated OpenCL into the execution process. In the first part the algorithm selects the OpenCL device with the largest number of compute units in addiction to the work group size.

After that it initializes the OpenCL context with the OpenCL *clCreateContext* command. The OpenCL context is required for managing OpenCL objects such as command-queues, memory, program and kernel objects and for executing kernels on one or more devices specified in the context. It continues with creating the OpenCL command-queue with the command *clCreateCommandQueue*.

The main loop of the CMA-ES algorithm is divided into kernels which are enqueued to the command-queue with the OpenCL *clEnqueueNDRangeKernel* command. This kernels are than executed on the device. When the termination criterion is met, the host machine reads the best solution from the device memory with the OpenCL command *clEnqueueReadBuffer* and outputs the result.

Algorithm 1 CMA-ES (Sequential version)

```
1:  procedure EXECUTE()
2:      ...
3:      // initialize variables
4:      init()
5:      ...
6:      while counteval < maxEvaluations do
7:          // get a new population of solutions
8:          population = samplePopulation()
9:          for i ∈ 0...populationSize do
10:             // evaluate solution
11:             evaluateSolution(population.get(i));
12:             // update counteval
13:             counteval += populationSize;
14:         // returns the best solution using a Comparator
15:         storeBest(comparator);
16:         // update the covariance matrix
17:         updateDistribution();
18:     ...
19:     // store the best solution
20:     resultPopulation.add(bestSolutionEver);
21:     ...
22:     // return the best solution
23:     return resultPopulation;
```

The analysis of the iterative CMA-ES algorithm has shown that the most time expensive steps are in the *SamplePopulation* method, in the inner for loop and in the *UpdateDistribution* method. To speed-up the algorithm we implemented eight OpenCL kernels, which encapsulate the logic of those methods. In order to execute this kernels on the device, the implementation firstly initializes these kernels and their input arguments with the OpenCL *clSetKernelArg* command. After that it sends all the precomputed data from the host to the device with the *clCreateBuffer* command. All this actions are implemented in the *init()* method.

After the initialization process has finished the main loop of the algorith is executed as shown in algorithm 2. In each iteration of the main loop the *SamplePopulation* method executes the OpenCL kernels with the OpenCL command *clEnqueueNDRangeKernel*. When the *SamplePopulation* method has finished the OpenCL device holds the current best solution found according to the fitness function which is implemented in the *computeFitnessKernel*.

In the last phase of the main loop the algorithm executes all the kernels in the *UpdateDistribution* method, which updates the covariance matrix as described in [5]. When the number of evaluations is bigger than the predefined maximum evaluation number, the host machine reads the best solution from the device memory, with the OpenCL command *clEnqueueReadBuffer* and outputs the result as shown in algorithm 2.

Algorithm 2 CMA-ES (OpenCL version)

```
1:  procedure EXECUTE()
2:      ...
3:      // Host: initialize variables
4:      // Host: initializes kernels and input arguments
5:      // Host: send data to OpenCL device
6:      init()
7:      ...
8:      // Host: main loop
9:      while counteval < maxEvaluations do
10:         // Host: execute kernels to sample population
11:         samplePopulation();
12:         // Host: update counteval
13:         counteval += (populationSize*populationSize);
14:         // Host: execute kernels to update the covariance
    matrix
15:         updateDistribution();
16:     ...
17:     // Host: read the best solution from the device
18:     clEnqueueReadBuffer(...);
19:     ...
20:     // Host: store the best solution
21:     resultPopulation.add(bestSolutionEver);
22:     ...
23:     // Host: return the best solution
24:     return resultPopulation;
```

3. RESULTS

The tests were performed on two main hardware configurations. Besides testing the performance improvement of our GPU implementation, we tested the differences between Nvidia and ATI graphic cards. The first machine was equipped with an i7-3820 CPU clocked at 3.60 GHz coupled with a GeForce GTX 650. The second configuration was an AMD FX-6300 clocked at 3500 MHz with a AMD Radeon R9 280X. The details of both cards are shown in Table 1

The jMetal framework offers plentiful problems to test optimization algorithms. We have chosen four typical problems, namely Rosenbrock, Sphere, Griewank, and Rastrigin. Each problem was tested on each hardware configuration and with different sample sizes. We measured the running time of the

Property	GeForce GTX 650	AMD Radeon R9 280X
Shading Units	384	2048
Memmory	1024 MB	3072 MB
GPU Clock	1058 MHz	850 MHz
Memory Clock	1250 MHz	1500 MHz

Table 1: GPU hardware comparison

algorithm for each run and computed averages for each sample size. The results for tests ran on GeForce GTX 650 are shown in Table 2

Sample size	Griewank	Sphere	Rosenbrock	Rastrigin
10	6ms	6ms	6ms	7ms
100	9ms	9ms	9ms	9ms
500	79ms	73ms	73ms	72ms
1000	226ms	232ms	231ms	225ms
2500	1274ms	1272ms	1278ms	1271ms
5000	5260ms	5273ms	5270ms	5275ms

Table 2: Run times on Nvidia GeForce 650gtx

To illustrate the performance improvement we ran the same tests on the Intel i7-3820 CPU. The results are shown in Table 3

Sample size	Griewank	Sphere	Rosenbrock	Rastrigin
10	11ms	11ms	11ms	12ms
100	108ms	112ms	110ms	113ms
500	856ms	881ms	879ms	886ms
1000	5428ms	5533ms	5588ms	5610ms
2500	123.39s	121.57s	153.94s	127.89s
5000	1208.84s	1184.37s	1188.17s	1265.53s

Table 3: Run times on Intel i7-3820 CPU

Comparing Tables 2 and 3 we can observe the improvements of the GPU implementation over the CPU. On larger problems, the GPU implementation outperforms the CPU by a factor of 230. On smaller tests however, the improvements are not substantial. This was expected due to the fact the kernels and the data need to be loaded from RAM to VRAM. In smaller problems the latency gives expression, while in the case of larger problems it does not impact overall computation time as much. The performance increase can also be observed in Figure 1 which was plotted in logarithmic scale.

We perform the same tests on the second configuration using an AMD-FX-6300 and an ATI Radeon R9-280x GPU shown in Tables 4 The results on the ATI GPU are a bit surprising. The graphic card is generally one of the fastest currently on the market, yet, it seems the Nvidia outperformed it on smaller problems. After profiling the execution using Jprofiler [11] we noticed the bottleneck was the AMD CPU, which took much longer to load the kernels and data on to the GPU. On larger problems, where the actual computation takes longer, the slow bandwidth between CPU and GPU is not noticeable.

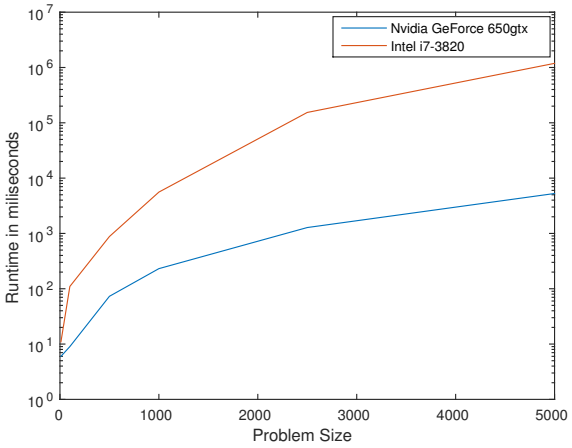


Figure 1: Runtime comparison on logarithmic scale

Sample size	Griewank	Sphere	Rosenbrock	Rastrigin
10	31ms	26ms	28ms	25ms
100	32ms	31ms	37ms	30ms
500	86ms	85ms	92ms	94ms
1000	219ms	216ms	232ms	221ms
2500	1090ms	1092ms	1102ms	1085ms
5000	4119ms	4125ms	4140ms	4134ms

Table 4: Run times on ATI Radeon R9-280x

From Table 5 we can observe the performance of the CPU implementation on the AMD CPU. Compared with the Intel CPU we can clearly see that the AMD CPU is much slower, which in turn explains the bottleneck in bandwidth between the GPU. The performance improvement of the GPU over CPU on the AMD configuration is shown in Figure 2 plotted in logarithmic scale.

Sample size	Griewank	Sphere	Rosenbrock	Rastrigin
10	17ms	17ms	17ms	17ms
100	152ms	130ms	140ms	139ms
500	1864ms	1824ms	1803ms	1871ms
1000	24128ms	24028ms	24106ms	24191ms
2500	232.52s	229.29s	223.91	234.76s
5000	3416.5s	3446s	3431.2s	3445.2s

Table 5: Run times on AMD FX-6300

4. CONCLUSION

Using an open source meta-heuristic algorithm framework we implemented a GPGPU version of CMA-ES algorithm using openCL, which is also open source. We empirically compared the computational results between the CPU and GPU version, which show improvement of the GPU over the CPU version. Additionally we compared two different configurations in an attempt to eliminate the possible bias of the framework towards certain manufacturers. Even though the configurations were not in the same price range and hence

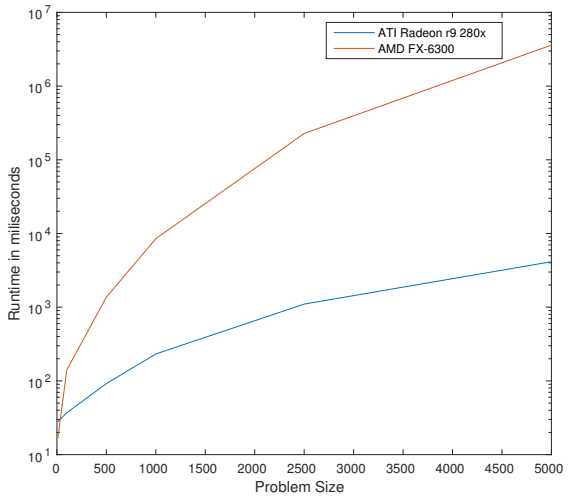


Figure 2: Runtime comparison on logarithmic scale

not directly comparable we show that the speedups obtained were in a reasonable margin. In best case our implementation is over 800 times faster then the CPU implementation. We did not notice any improvement in case of smaller problems mainly due the time penalty of transferring problem data from RAM to vRAM.

5. ACKNOWLEDGMENT

The authors would like to thank dr. Peter Korošec for advising and guiding the research.

6. FUTURE WORK

The hardware used to perform tests was not directly comparable. The Intel configuration had a much better CPU while the AMD configuration had a state of the art GPU. The next step is to run the tests on hardware configurations of the same price point. Additionally we are working on upstreaming our implementation to the Jmetal framework.

7. REFERENCES

[1] J. Durillo, A. Nebro, and E. Alba. The jmetal framework for multi-objective optimization: Design and architecture. In *CEC 2010*, pages 4138–4325, Barcelona, Spain, July 2010.

[2] J. J. Durillo and A. J. Nebro. jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42:760–771, 2011.

[3] B. Gaster, L. Howes, D. R. Kaeli, P. Mistry, and D. Schaa. *Heterogeneous Computing with OpenCL*. 2012.

[4] N. Hansen. The cma evolution strategy: a comparing review. In *Towards a new evolutionary computation*, pages 75–102. Springer, 2006.

[5] N. Hansen. The CMA evolution strategy: A tutorial. *Vu le*, 102(2006):1–34, 2011.

[6] M. Harris. Gpgpu: General-purpose computation on gpus. *SIGGRAPH 2005 GPGPU COURSE*, 2005.

[7] M. Macedonia. The gpu enters computing's mainstream. *Computer*, 36(10):106–108, 2003.

[8] A. Munshi. The opencl specification. In *2009 IEEE Hot Chips 21 Symposium (HCS)*, pages 1–314. IEEE, 2009.

[9] C. Nvidia. Programming guide, 2008.

[10] M. Pharr and R. Fernando. *Gpu gems 2: programming techniques for high-performance graphics and general-purpose computation*. Addison-Wesley Professional, 2005.

[11] J. Shirazi. Tool report: Jprofiler. *Java Performance Tuning*, 2002.

[12] S. Tsutsui and P. Collet. *Massively parallel evolutionary computation on GPGPUs*. Springer, 2013.

[13] W. H. Wen-Mei. *GPU Computing Gems Emerald Edition*. Elsevier, 2011.

Building visual domain-specific languages using the semiotic approach: A case study on EasyTime

Iztok Fister Jr.
Faculty of Electrical Engineering and Computer Science, University of Maribor
Smetanova 17, 2000 Maribor, Slovenia
iztok.fister1@um.si

Iztok Fister
Faculty of Electrical Engineering and Computer Science, University of Maribor
Smetanova 17, 2000 Maribor, Slovenia
iztok.fister@um.si

ABSTRACT

This paper presents the development and usage of the EasyTime III Visual Domain-Specific Language (VDSL) for measuring time during sports competitions that enables users to create the VDSL programs visually. Indeed, this new hybrid VDSL creating approach deals with semiotics' analysis in place of building a meta-model. Thus, each visual element, like an icon, is handled as a sign with its representation (form and shape), reference (object) and meaning (implementation of object). Each visual element is then mapped into DSL language construct as defined by object implementation. In general, this approach is suitable, when Domain-Specific Language (DSL) using a Textual User Interface (TUI) is already implemented within a certain application domain and, therefore, the developer can be focused mainly on designing a Graphical User Interface (GUI). In this study, the existing EasyTime DSL has been upgraded to EasyTime III VDSL using the hybrid VDSL creation approach. Experiments of measuring time in an aquathlon have shown that the programming in EasyTime III is simple and efficient, whilst all other features of EasyTime DSL are preserved within the new VDSL.

Keywords

domain-specific language, EasyTime, semiotics

1. INTRODUCTION

Until recently, measuring time during sports competitions was performed manually by timekeepers using time obtained from stopwatches assigned to specific competitors according to their starting numbers. The rapid expansion of Radio Frequency IDentification (RFID) [6] technology has led into the development of various measuring devices. However, measuring devices represent only one part of the story, because measuring times during sports competitions cannot be achieved without a measuring system. This system collates timed-events from different measuring devices into a central database that enables organisers to track the results of competitors faster, more efficiently, and more accurately. In order to simplify working with the measuring system for tracking the results, the EasyTime domain-specific language was proposed for the Double Triathlon in 2009 [7]. This version, based on a specific compiler/generator, was developed for compiling an EasyTime source code into an abstract machine code. Modifying and widening the EasyTime domain-specific language demanded interventions directly into the compiler/generator. Therefore, an improved version of EasyTime was proposed in [8, 11, 10, 9], where EasyTime was defined formally as (domain analysis, abstract syntax, concrete syntax, formal semantics, and code generation), whilst a LISA [18] was used for building as a suitable compiler/generator. In this paper, we move forward in order to improve user experience and, thus, propose EasyTime III Visual Domain-Specific modelling Language (VDSL). This explores the visual interface to simplify the programming tasks of the measuring system. This VDSL addresses the shortcomings of its predecessor, i.e., simplifying its development. The users' visual programming consists of constructing the user model. This model is then translated into EasyTime DSL constructs. Indeed, a semiotics theory [12] is applied. Semiotics is the study of signs [2], where each sign consists of representation, object, and meaning. The meanings of the signs are defined explicitly in the user model [5]. Thus, a meta-model step can be avoided by the traditional creation of DSMLs. This approach is, therefore, also named as a hybrid VDSL creation. The VDSL was tested on building the EasyTime III program for measuring time during the aquathlon competition. The obtained results showed the power of the created visual tool that brings a great user experience on the one hand and simplified creating the visual programs.

The structure of the paper is as follows. Section 2 deals with a description of the proposed EasyTime III VDSL. In Section 3, created VDSL was applied for building the EasyTime III VDSL program in measuring time during an aquathlon competition. The paper concludes with Section 4, which summarizes the performed work and outlines the possible directions for the future.

2. EASYTIME III VISUAL DSL

The design of visual languages is a very complex process that, besides a knowledge of computer science, demands also the knowledge of areas like Psychology, Sociology, Anthropology, Linguistics, Design, and Engineering. For the development of EasyTime III VDSL, the following phases need

to be carried out:

- Concept analysis,
- Visualization of features,
- Semiotic analysis,
- Code generation.

In the remainder of the paper all the mentioned phases are described in detail.

A concept analysis identifies the concepts and relations between concepts. The analysis of a measuring time domain summarizes the results as proposed in [8] because VDSL EasyTime III addresses the same application-domain (i.e., measuring time) as the original EasyTime DSL. The concept analysis divides the concepts into features, and then these features into sub-features. However, the features and sub-features may be mandatory or optional. In order to denote the attitudes between concepts, features and sub-features, the concept analysis defines the following relations:

- *all*: All features or sub-features are mandatory.
- *more-off*: The feature may be either one of the sub-features from a set of sub-features or any combination of sub-features from the same set.
- *one-off*: The feature may be one of the sub-features from a set of sub-features but not any combination of sub-features from the same set.

The concept diagram of the measuring time domain is depicted in Figure 1, from which it can be seen that the concept *Race* consists of seven features (i.e., *Events*, *Transition area*, *Control points*, *Measuring time*, *Begin*, *End* and *Agents*).

2.1 Visualization of features

During the visualization process, the features are mapped into appropriate graphical interfaces, as presented in Table 2.1. The Table is interpreted as follows. Icons I_{begin} and I_{end} denote the features *Begin* and *End*, respectively. Events can be represented using icons I_{swim}, I_{bike}, and I_{run} and described the sub-features, as follows: *Swimming*, *Cycling*, and *Running*. The feature *Transition areas* is identified by icon I_{ta}, while *Measuring devices* are marked using icons I_{md0}, I_{md1}, and I_{md2}.

2.2 Semiotics‘ analysis

Semiotics is the study of 'everything that can be taken as a sign' [17]. This is an interdisciplinary theory that comprises domains of meanings in a variety of other disciplines like Psychology, Anthropology, Biology, Logic, Linguistics, Philosophy, and even Software Engineering [4]. Modern semiotics consists of two independent theories, as follows: Semiology and semiotics. The former was developed in Switzerland by Ferdinand de Saussure [16], while the latter in North America by Charles Sanders Peirce [13]. De Saussure's theory of signs originated from the language theory as a system of arbitrary signifying units. The Peircean theory of signs is based on logic and epistemology [2]. He defined the sign as anything that stands for something else, to somebody, in some respect or capacity. Nothing is a sign until it is interpreted by somebody. Peirce described signs as threefold structures consisting of: Representation, reference (object), and meaning (interpretation) (Figure 2). Two characteristics of these structures are valid: Firstly, a direct connection between a representation and reference need not necessarily exist. Secondly, a meaning is always a mediator between a representation and reference. That is, the sign does not exist until some interpretation of representation is taken that has some meaning for somebody. In other words, a sign requires the concurrent presence of all three characteristics. On the other hand, there can be many meanings to a sign. In our study, the Peircean structure of signs was taken into

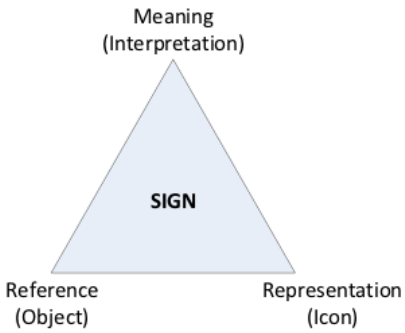


Figure 2: Structure of signs.

account in order to prescribe the unique meanings of them. Indeed, unique translation of signs can be achieved to EasyTime domain-specific language constructs. In line with this, semantic analysis is not needed for this translation. Therefore, this process of generating the EasyTime III VDSL was named a hybrid approach, and it can be applied usefully when the textual DSL is already developed in this application domain and an upgrade to the visual interface needs to be developed. The semiotics of EasyTime III can be described with the structures as illustrated in Table 2, from which it can be seen that each icon is represented by a corresponding object. In our study, objects are represented as C++ classes. The objects' meanings are defined by the implementation code. The source code in EasyTime DSL is generated as a result of the implementation.

For instance, an icon I_{begin} is referenced with an object *Begin* that is created by a parameter *Laps* which determines the number of laps. This object is responsible for generating the EasyTime DSL language construct "upd *STARTx*". Note that this character x denotes the integer value (also location counter, *lc*) necessary for distinguishing the different instances of the same variables because, in contrast, the same names of the variables will be generated for different control points. The variable *lc* is initialized during race configuration. The icons $I_{swim}, I_{bike}, I_{run}$ representing the events are represented by objects *Event* (Algorithm 1) that are responsible for generating two DSL EasyTime language constructs "dec *ROUNDx*" and "upd *INTERx*" (Algorithm 2). A class *Event* consists of three variables (*type*, *laps*, *lc*) and three methods (constructor *Event*, *initialize*,

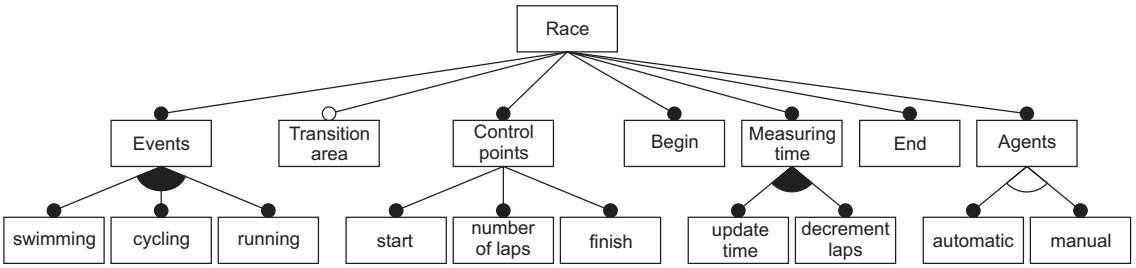


Figure 1: Concept diagram of measuring time domain.

Table 1: Translation of the application domain features to graphical elements.

Application domain features	Graphical elements
Begin race	I_{begin}
Events *(Swimming, Cycling, Running)*	$I_{swim}, I_{bike}, I_{run}$
Transition area	I_{ta}
End race	I_{end}
Measuring time	$I_{md0}, I_{md1}, I_{md2}$
Control points *(start, number of laps, finish)*	square dots (control points)
Agents *(automatic, manual)*	square dots (measuring points)

Algorithm 1 Definition of object 'Event'

```
1: class Event {
2:     int type;    // type of events Swim, Run, Bike
3:     int laps;    // number of laps
4:     int lc;      // index of generated variables
5:
6:     Event(int Type, int Laps, int Lc);
7:     void initialize();
8:     void generate();
9: }
```

generate). The variable *type* determines the type of event, i.e., *Swim*, *Run*, *Bike*, variable *laps* the number of laps that the competitor needs to accomplish a specific discipline, and variable *lc* determines the instance of a specific variable. The variable *laps* is a context information which is obtained by the user. The method *Event* constructs the object with appropriate parameters, method *initialize* generates the specific EasyTime DSL code necessary to initialize the variables within the scope, and the method *generate* generates the specific EasyTime DSL code appropriate for this event. A detailed implementation of the mentioned methods can be seen in Algorithm 2.

Algorithm 2 Implementation of object 'Event'

```
1: Event::Event(int Type, int Laps, int Lc)
2:     type = Type; laps = Laps;
3:     lc = Lc;
4: }
5: void Event::initialize() {
6:     gen(op_init, var_round, lc, laps);
7:     gen(op_init, var_inter, lc, 0);
8: }
9: void Event::generate() {
10:    gen(op_dec, var_round, lc);
11:    gen(op_upd, var_inter, lc);
12: }
```

Note that the other objects are represented and implemented in a similar manner.

2.3 Code generation

Code generation is divided into:

- the source code generation,
- the object code generation.

In the first phase, a visual representation of a real race using the graphical elements (also user model) is mapped into the EasyTime DSL source code while, in the second phase, this source code is compiled into object code using the LISA [18] compiler/generator. Semantic domains need to be defined for translating the graphical elements into the EasyTime DSL source code. In EasyTime III, two semantic domains are defined (Table 3). Both domains are used for a proper calling of the objects represented in the Table 4 in the translation phase. The former (i.e., D_{Event}) defines sub-features of the feature *Event*, while the latter (i.e., D_{Event}) sub-features of the feature *Md*. All the other parameters of the corresponding objects are of integer type. In the second phase, the EasyTime DSL source code translated from the corresponding user visual model is generated into virtual machine object code. The readers can obtain more information about this process in [8].

Table 3: Semantic domains in EasyTime III.

$D_{Event} = \{Swim, Bike, Run\}$	$Event\_Type \in D_{Event}$
$D_{Md} = \{Md0, Md1, Md2\}$	$Md\_Type \in D_{Md}$

Table 2: Translation of the application domain concepts to semiotic signs.

Icons	Objects	Meanings
I_{begin}	Begin($Laps, Lc$)	"($ROUNDx == \%Laps$) → upd $STARTx$;"
$I_{swim}, I_{bike}, I_{run}$	Event($Event\_Type, Laps, Lc$)	"dec $ROUNDx$;" "upd $INTERx$;"
I_{ta}	End(Lc) Begin($Laps, Lc$)	"($ROUNDx == 0$) → upd $FINISHx$;" "($ROUNDx == \%Laps$) → upd $STARTx$;"
I_{end}	End(Lc)	"($ROUNDx == 0$) → upd $FINISHx$;"
$I_{md0}, I_{md1}, I_{md2}$	Md($Md\_Type, Mp, Ag, [Ip\|Fn]$)	"mp[%Mp] ← agnt[%Ag] {" | "}"

3. MEASURING TIME IN AN AQUATHLON COMPETITION USING EASYTIME III VDSL

Aquathlon is a relatively young sport, the first National Competition being held in the USA in 1965. It belongs to a class of multi-sports, where the race consists of continuous two-stage disciplines involving swimming followed by running [14, 15]. Between both disciplines, however, a so-called transition area exists, where the competitors who finish the swimming prepare themselves for running. The time spent in the transition area is added to the final result of each specific competitor. Aquathlons are similar to triathlons. Triathlons, however, have cycling in addition to swimming and running. As a result, an aquathlon covers triathlon distances as well. For instance, a 1 km swim is followed by a 5 km run, etc. Distances vary depending upon the race venue and race organisers. For building the EasyTime III visual programs, an EasyTime III visual editor was developed using the Qt [1, 3], where a lot of bindings with other languages also exist, e.g., Java, Python, Ruby, etc. The visual program in an EasyTime III visual editor for measuring time during an aquathlon competition is illustrated in Figure 3. The visual editor is the graphical editor for describing the race to be measured. It consists of a graphical editor area, where the race should be described and a control area consisting of the buttons necessary for editing. In fact, the buttons represent either the application domain features or miscellaneous controls devoted to managing the visual programs. These buttons are divided into three subsets representing:

- the race configuration, i.e., buttons for dragging the icons representing the control points (e.g., I_{begin}, I_{end}, I_{swim}, $I_{bike}, I_{run}, I_{ta}$) and dropping them into specific positions within the visual editor area,
- situated measuring devices, i.e., buttons for dragging the icons representing the measuring points (e.g., I_{md1}, I_{md2}, I_{md0}) and dropping them into specific positions within the visual editor area,
- controls, namely the buttons necessary for opening, saving and erasing the visual programs, and generating the object code from the visual program.

Note that the graphical editor area consists of two fields in which icons are placed sequentially. The upper is sensitive to the buttons designated control points, whilst the lower to the buttons described measuring points. In fact, the upper fields of icons determine the configuration of a race, whilst the lower fields are where measuring devices have to be situated. Each icon also includes a square dot that enables the control point to be connected with the measuring point. Moreover, the transition area icon, similar to the measuring device with two mats, includes two square dots. Connection has to be made by dragging the source control point and dropping it into the destination measuring point or vice versa. Furthermore, the EasyTime III visual editor also includes certain context dependent information, like the number of laps, IP address of the measuring device, or an input file name. Zero laps on the swimming icon means that the swimming will not be measured on this measuring device. This information can be obtained by pressing the right-hand mouse button. The measuring time during an aquathlon competition is configured as follows. The upper graphical editor field determines the race configuration (i.e., start of race, swimming, transition area, running and finish of race), whilst the lower graphical editor field denotes situated measuring devices (i.e., realised by a measuring device with two measuring points). The connections between the control points and measuring points determine where the particular event has to be measured. For instance, the finish time of swimming has to be measured at measuring point 1 (antenna mat 1), whilst the other three running events (start time, intermediate times, and finish time) have to be measured at measuring point 2 (antenna mat 2).

3.1 Source code generation for measuring time in an aquathlon competition

Source code generation starts with translating the user visual model (consisting of icons and links) into semiotics objects. Indeed, an area of central points *CP* and area of measuring points *MP* together with an adjacent matrix representing connections between control and measuring points are generated. The results of this translation are illustrated in Table 4. It can be seen from the Table that the control points' area *CP* consists of six semiotics objects representing two disciplines (e.g., swimming and running) embraced between *Begin* and *End* semiotics objects. The generated names of the variables in these semiotics objects are distinguished according to their location counter. For instance, all variables referring to the first discipline are generated with $lc = 1$, whilst the variables referring to the second discipline with $lc = 2$. There are two measuring points within an area *MP*, and four links where the time should be measured. Note that the adjacent matrix *LN* designates the connections between the control and measuring points. The program presented in the Algorithm 3 is generated according to the race configuration. This generation is straightforward when someone follows the code generation as defined by the

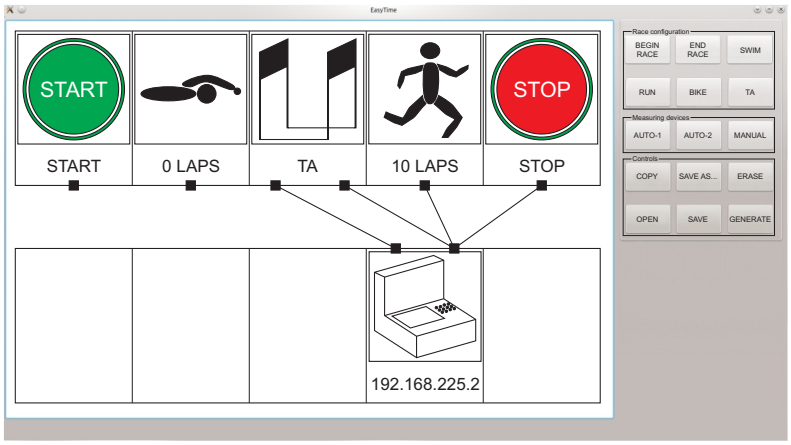


Figure 3: Visual program for measuring time in an aquathlon competition.

Table 4: Result of algorithm 'config_race'.

CP = { $Begin$(0,1),$Event$(Swim, 0, 1),End(1),$Begin$(0,2),$Event$(Run, 10, 2),End(2) }
MP = { Md(Auto-2,1,1,"192.168.225.2"), Md(Auto-2,2,1,"192.168.225.2") }
LN = { (3,1),(4,2),(5,2),(6,2) }

Algorithm 3 Source code for measuring time during on aquathlon.

```
 1: 1 auto 192.168.225.2;
 2:
 3: START1 = 0;
 4: ROUND1 = 0;
 5: INTER1 = 0;
 6: FINISH1 = 0;
 7: START2 = 0;
 8: ROUND2 = 10;
 9: INTER2 = 0;
10: FINISH2 = 0;
11:
12: mp1[1] → agnt[1] {
13:   (ROUND1 == 0) → upd FINISH1;
14: }
15: mp1[2] → agnt[1] {
16:   (ROUND2 == 10) → upd START2;
17:   (true) → upd INTER2;
18:   (true) → dec ROUND2;
19:   (ROUND2 == 0) → upd FINISH2;
20: }
```

meanings of the semiotics objects. Note that this code is functionally equivalent to the code written by the domain expert manually. Later, EasyTime LISA compiler/generator is used for object code generation [8].

4. CONCLUSION

This paper proposes a new hybrid VDSL creation approach that is suitable when the DSL already exists in certain application-domain. In this manner, a programmer exploits existing DSL and focuses on designing a graphical user interface. Thus, he/she avoids constructing the meta-model and, in line with this, the usage of the complex development frameworks, like Eclipse. The modelling step is, here, substituted with semiotic analysis, where each visual element, like icon or link, is handled as a sign with its representation (object) and meaning (implementation of the object). From these so-called semiotics objects, the source code is then generated, which can be translated into object code using the existing compiler/generator. The proposed approach was tested by the development of EasyTime III VDSL for measuring time during sports competitions and starting with the concepts of an application domain obtained through domain analysis. These concepts serve as the basis for building EasyTime III graphical user interfaces on the Qt based visual editor. The visual program (also user model) built using this editor is then mapped into EasyTime DSL constructs. The result of this translation is the EasyTime DSL source program. Translating this source program using the LISA compiler/generator generates an object AM code for the EasyTime measuring system. As a result, the developed EasyTime III VDSL enables ordinary users (e.g., organisers of sports competitions) to create programs for measuring time application-domain visually. That is, instead of writing the program in text editor, only 'point-and-click' is needed with the icons on the screen. With the proposed EasyTime III, the programming of a measuring system within visual environments is easier, faster, and more effective.

5. REFERENCES

[1] J. Blanchette and M. Summerfield. *C++ GUI programming with Qt 4*. Prentice Hall PTR, 2006.

[2] D. Chandler. *Semiotics: The Basics*. Routledge, New York, US, 2007.

[3] M. Dalheimer. *Programming with QT: Writing portable GUI applications on Unix and Win32*. O'Reilly Media, Incorporated, 2002.

[4] C. de Souza. *The semiotic engineering of human-computer interaction*. MIT Press, Cambridge, England, 2005.

[5] C. de Souza. Semiotic perspectives on interactive languages for life on the screen. *Journal of Visual Languages & Computing*, 24(3):218 – 221, 2013.

[6] K. Finkenzeller. *RFID handbook: fundamentals and*

applications in contactless smart cards, radio frequency identification and near-field communication. Wiley, 2010.

[7] I. Fister Jr. and I. Fister. Measuring time in sporting competitions with the domain-specific language Easytime. *Electrotechnical review*, 78(1–2):36–41, 2011.

[8] I. Fister Jr., I. Fister, M. Mernik, and J. Brest. Design and implementation of domain-specific language Easytime. *Computer Languages, Systems & Structures*, 37(4):151–167, 2011.

[9] I. Fister Jr, T. Kosar, I. Fister, and M. Mernik. Easytime++: A case study of incremental domain-specific language development. *Information Technology And Control*, 42(1):77–85, 2013.

[10] I. Fister Jr., M. Mernik, I. Fister, and D. Hrnčič. Implementation of Easytime formal semantics using a LISA compiler generator. *Computer Science and Information Systems*, 9(3):1019–1044, 2012.

[11] I. Fister Jr., M. Mernik, I. Fister, and D. Hrnčič. Implementation of the domain-specific language Easytime using a LISA compiler generator. In *Computer Science and Information Systems (FedCSIS), 2011 Federated Conference on*, pages 801–808. IEEE, 2011.

[12] C. Peirce. Collected papers of charles sanders peirce. volume 1–8, Cambridge, MA, 1931–1958. Harward University Press.

[13] C. Peirce and V. Welby. *Semiotic and significs: the correspondence between Charles S. Peirce and Lady Victoria Welby*. UMI-books on demand. Indiana University Press, 1977.

[14] S. Petschnig. *10 Jahre IRONMAN Triathlon Austria*. Meyer & Meyer Verlag, 2007.

[15] S. Rauter and M. Doupona Topič. Perspectives of the sport-oriented public in slovenia on extreme sports. *Kinesiology*, 43(1):82–90, 2011.

[16] F. Saussure. *Course in General Linguistics*. Duckworth, 1976.

[17] E. Umberto. *A theory of semiotics*. Advances in semiotics. Indiana University Press, 1976.

[18] University of Maribor. Lisa 2.2. `http://labraj.uni-mb.si/lisa/`, 2013. Accessed 17 August 2016.

A new population-based nature-inspired algorithm every month: Is the current era coming to the end?

Iztok Fister Jr., Uroš Mlakar, Janez Brest, Iztok Fister
Faculty of Electrical Engineering and Computer Science, University of Maribor
Smetanova 17, 2000 Maribor, Slovenia
iztok.fister1@um.si

ABSTRACT

Every month at least one new population-based nature-inspired algorithm has been released in literature. Until recently, there were probably more than 200 algorithms of this kind in books, papers and proceedings. Many researchers discuss that this research area is becoming flooded with new algorithms that are in fact the old algorithms in a new disguise. Potentially, such behavior could be leading into the emergence of pseudoscience. In this paper, we try to find some answers to the questions what lead authors to propose and develop the new nature-inspired algorithms and what their benefits in doing so are. We propose ways in which to stop the emergence of new algorithms. In line with this, we have found that proposing the new population-based nature-inspired algorithms is actually similar to the swarm intelligence behavior in nature, where the role of population members is acted by authors of the new algorithm with the goal to publish a paper, thus promoting its algorithm and spreading it all over the world.

Keywords

metaphors, nature-inspired algorithms, swarm intelligence

1. INTRODUCTION

Approximately 50 years ago, the time emerged when scientists began applying algorithms solving the problems on digital computers by mimicking the human brain widely. These methods were called **Artificial neural networks** [13]. Artificial neural networks were proposed in the 40s in the previous century, but it took some time before the community began to use them widely for scientific and first practical usage. These networks were really interesting methods and many scientists claimed that artificial neural networks would power the world in the near future. Artificial neural networks were counted into pure artificial intelligence and now there are many various types of these networks for solving particular tasks in theory and practice. That historical time was also the time where people were limited with hardware and software. Thus, people were unable to test and develop their methods widely. However, some years after the creation of artificial neural networks, another discipline (or alternative to artificial neural networks) was developed actively by the scientific community. The name of this discipline, that was coined later, was **Evolutionary computation**. Evolutionary computation was based on the natural evolution of species and respected the theory of Charles Darwin. Initially, the Evolutionary Algorithms (EAs) simulated the operators of mutation and crossover, where the individuals to survive were selected according to their fitness values. The fitness value was determined according to the evaluation of the fitness function that corresponded to the problem to be solved. Nevertheless, over the years the EAs were divided into the following kind of algorithms: Genetic algorithms [18], evolution strategies [1], genetic programming [16] and evolutionary programming [26]. The main differences between these sub-families were basically in the representation of individuals, e.g., binary representation was used by genetic algorithms, floating point representation by evolution strategies, finite state automata by evolutionary programming and programs in Lisp by genetic programming. Additionally, it is important to mention that in the 80s other metaheuristics were also designed [23, 8, 9, 10, 15]. The period when these methods appeared in the literature was a little bit calmer compared with nowadays. It was a time without the Internet and also access to the papers was limited. Additionally, in these times people did not yet know the term **Publish or perish** [19, 2]. Scientists should not have to be forced to publish for any price in order to their hold position at the university or scientific institute. But things were changed quickly. The years of 90s came rapidly. In this scientific area a new paradigm was proposed that incorporated the social behavior of many agents that guided them into complex behavior. The roots of this method, which is named **Swarm intelligence**, can be found in the dissertation of Marco Dorigo [4]. His method proposed the colonies of ants for solving discrete optimization problems. A little bit later, in 1995, Kennedy and Eberhart [14] applied the behavior of bird swarms and fish schools into an algorithm with the name **Particle swarm optimization**. These two methods were the beginners of the new community movement, i.e. the so-called swarm intelligence community. However, in the 90s and early 2000s the community did not think that these two powerful algorithms were the stepping stones for the development of uncountable nature-inspired algorithms and, potentially, the flood of algorithms that led into pseudoscience. In this paper, we

try to get answers to the following questions:

- What is actually considered as a new nature-inspired algorithm?
- What motivates researchers to propose new algorithms?
- What they have if they propose a new algorithm?
- What is a generic recipe for proposing a new algorithm?
- What are the implications of new algorithms?
- How to stop the invasion of new algorithms?
- Is proposing new algorithms basically swarm intelligence behavior itself?

Insights on these questions will be highlighted in the remainder of the paper.

2. NATURE-INSPIRED ALGORITHMS

At the start, it is very hard to define exactly what the population-based nature-inspired algorithms actually are. However, there are many definitions and most of these definitions say that population-based nature-inspired algorithms are a kind of algorithms that are inspired by natural, biological and even social systems, and are intended to solve problems in a similar way to what nature does. Even today, there are a few taxonomies that try to deal algorithms in different groups. One of the taxonomies is a taxonomy published in 2013 by Fister et al. [5] where algorithms were split into 4 groups [5]:

- Swarm intelligence based algorithms,
- Bio-inspired that are not swarm intelligence based,
- Physics and chemistry based and
- Other algorithms.

Algorithm 1 Generic pseudo-code of most of the population-based nature-inspired algorithms

```
initialize individuals within bounds using a particular randomization generator
evaluate all individuals
while termination criteria not met do
    move all individuals according to proposed formulas
    evaluate all individuals
    find the best individuals
end while
return the best individual and vizualize
```

Generic pseudo-code for most of the algorithms in this taxonomy, especially for the first and second group, is presented in Algorithm 1.

3. THE NEW POPULATION-BASED NATURE-INSPIRED ALGORITHMS

The previous section gave readers a short overview of the population-based nature-inspired algorithms, while this section will be concentrated on the implications of the new population-based nature-inspired algorithms.

3.1 Current publishing era

Without doubt we can say that this era, after the year 2000, led to the hard race between scientists and even institutions. People that work in universities and institutes are forced to publish works that will achieve a lot of citations, since good works with many citations help universities in global rankings. Better universities have attracted more, better students. In line with this, they could obtain the government and industrial projects more easily. Projects mean money, while more students mean more tuition fees. Fortunately, this is not true for each country. For example, there are no tuition fees for students of Government Institutions in Slovenia. Only, doctoral studies require students to pay a tuition fee that is actually not as high as it is abroad. In order to meet these goals universities could make pressure on researchers to publish more and better works. This manner is also connected with the term **Publish or perish**. However, to satisfy these goals is a little bit easier for mature and well-known researchers, while newbies have mostly enormous problems. For example, some students in China are even prepared to give a kidney for having a paper accepted in a journal with impact factor. Is not publishing becoming similar to any fight sport (win at any price)? Let us leave politics and go back to our population-based nature-inspired algorithms. So, after the year 2000 when Ant colonies and Particle Swarms became popular, interesting and widely used algorithms, some researchers began thinking whether there was a possibility to develop or just propose a new algorithm that should be based on any other inspiration from nature. If we look into our bioshpere, we can easily find a lot of animal species, different trees, natural processes, social behaviors for developing the optimization algorithms. When researchers found an inspiration, they then needed to coin some operators that mimic the behavior of their inspiration and, later, put this magic into the universal recipe that was presented in the previous chapter. Most of the algorithms only used a different formula for moving individuals and that was all the magic behind an algorithm. Paradigm was the same, but only some minor changes were incorporated in the developing of the brand new population-based nature-inspired algorithm. After developing, the time has started for publishing the new algorithm. Usually, all researchers need to validate their new algorithms on some well-known benchmark functions or on some engineering problems. Of course, all algorithms have beaten other well-known algorithms without any problem, although nobody cared if researchers used special benchmarks and tested on special dimensions and compared with basic well-known algorithms and not with their improved versions. At the beginning, they were successful and achieved good publications in journals with nice impact factors and even at good conferences. Until 2010, almost nobody had yet cared about new algorithms but, after 2010, many scientists began to doubt about the originality of works. However, it was too late. Nowadays, we are in 2016. According to the list on Github (**www.github.com/fcampelo/EC-Bestiary**), we counted (as of 5 August 2016) that there are 101 nature-inspired algorithms (Fig. 1). Anyway, there are many others that are not on this list.

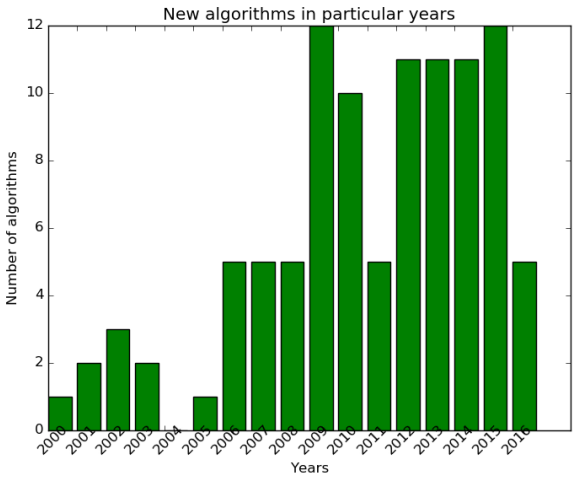


Figure 1: The emergence of new algorithms (according to Github repository EC-Bestiary.)

4. MOTIVATION FOR THE DEVELOPMENT OF THE NEW POPULATION-BASED NATURE-INSPIRED ALGORITHMS

In order to find what is actually the motivation behind the development of a new algorithm, we should take a more global view. As many papers have already shown [21, 22, 25, 24], that the new algorithms are actually old algorithms in new disguises (similar as Spy character in Team Fortress 2 game) we should discover why researchers created a new algorithm artificially and masked it within the new inspiration. We believe that motivation is connected with publishing and citations. This research area is really wide and offers excellent opportunities for publication. Along with this statement, publication also opens a pool for citations. Thus, one of the main motivations behind new algorithms is more or less the current publishing situation.

5. GENERIC RECIPE FOR PROPOSING THE NEW POPULATION-BASED NATURE-INSPIRED ALGORITHM

After studying some of the new population-based nature-inspired algorithms, we can simply propose a generic recipe that captures all the ingredients of the successful creation of a new nature-inspired algorithm. The generic recipe is presented in Algorithm 2. At the beginning, researchers are looking for an idea. While searching is in progress, when they get an idea, they need to reconcile the name of the new algorithm. If the name is still free, then the researchers need to develop formulas, choose test functions, run experiments and publish a paper. At the end, they also need to spread the word about the algorithm. This could be done easily by various social networks.

6. IMPLICATIONS OF THE NEW NATURE-INSPIRED ALGORITHMS

Mostly, the new population-based nature-inspired algorithms do not affect older researchers (however, there are some exceptions), while new algorithms of this kind are excellent bait for younger researchers, especially students. Students do not have a global view on a particular research area and, thus, they simply succumb to new algorithms. Many papers that propose new algorithms are written in tempting style and it really attracts students. Moreover, even various researchers from other totally different research areas (geology, space exploration, leisure studies, etc.) sometimes use new algorithms for research. They do not care about roots, they just want to solve their problems (no matter what method solves the problem). At the end, industry is the last thing here. People from industry need a solution for their problem. If they see that one algorithm is good for their problem they take it.

Algorithm 2 Generic recipe for the new nature-inspired algorithm proposal

```
Watch TV, browse internet, go for a walk in nature in order to get inspiration for your new algorithm
while searching in progress do
 if you get an idea about a new algorithm:
  check if your proposed name is still free (browse via major search engines and databasess)
   if name is free:
    develop formulas
    choose benchmark functions
    run experiments
    write and publish a paper
    spread the word about your algorithm
   else repeat until you find another inspiration
```

7. HOW TO STOP THE INVASION OF THE POPULATION-BASED NATURE-INSPIRED ALGORITHMS?

We believe that the invasion of the new population-based nature-inspired algorithms could be stopped within the next five years. All trends in evolution are the same. At the beginning there is a high rise, when it comes to the top then it goes down slowly. At the moment the trend is not rising any more and many Editors and Reviewers are informed about this problem. Recently, many papers that show the problems of this area have been released [22, 7, 6, 21, 5]. Some Journal Editorial Boards have even revised their rules and they do not accept papers where questionable metaphors are presented [11]. By the same token, the Matthew effect [20, 17] that depicts "the rich tend to be richer" almost always works. Hence, old and famous algorithms will always be more powerful than artificially created algorithms.

7.1 Swarm intelligence behavior in the population-based nature-inspired algorithm development

The definition of the swarm intelligence based algorithms were outlined in the previous sections. The swarm intelligence based algorithms family are, these days, more popular and there are also many journals that are devoted to these kinds of algorithms. As a matter of fact, swarm intelligence based algorithms propose **many individuals** that execute **simple actions** and their **behavioral actions** leads into a **complex** and **decentralized** system. Can we find any parallel points with the process of new nature-inspired al-

Table 1: Parallel points between the definition of swarm intelligence and the process of creation of new nature-inspired algorithms.

Definition of swarm intelligence	New nature-inspired algorithm creation
many individuals	many authors
simple actions	watching the inspiration in nature, giving a new name for the algorithm, developing a formula
behavioral actions	publishing a paper
complex	name motivates other individuals, new hybrid and adaptive variants
decentralized	algorithm is spread all over the world, impossible to stop spreading this algorithm – the same as viruses for example

gorithm creation? The Table 1 shows point to point comparison among these two processes. What is the most interesting is that the process of new algorithm creation possesses the behavior of swarm intelligence. Swarm intelligence based algorithms consist of many individuals. On the other hand, the process of population-based nature-inspired algorithms is guided by many authors. Simple actions (for example foraging in bees or pheromone tracking in ants or even home building by termites) are, in the process of new algorithm creation, defined as simple actions where authors try to find an inspiration in nature, give their algorithm a bombastic name and even develop a formula that will mostly guide an evolutionary process. Behavioral actions are, basically, connected with publishing a paper in a journal or at a conference, while complex behavior is connected with spreading the algorithm all over the world with the help of social media [12, 3] (Twitter, Facebook, Researchgate, Academia, Google groups, etc.), search engines (Google, Yahoo), emails (many authors send emails to others attached with the source code and pdfs), mouth sharing (via conferences and social meetings) and so on. Decentralized behavior is expressed as an algorithm that is spread all over the world and it is also impossible to stop it spreading. Especially, new researchers take a new algorithm and create new variants (mostly hybrid variants or apply this algorithm on industrial problems) and then, again, we obtain complex behavior.

8. CONCLUSION

In this paper we took a view on the new population-based nature-inspired algorithms‘ development. The new population-based nature-inspired algorithms are released every month and, basically, they have nothing special and no novel features for science. Thus, the development of the new population-based nature-inspired algorithms may be becoming very dangerous for science. We found that there are many social tensions that lead authors towards the new population-based nature-inspired algorithm creation, especially the wish for quick paper publishing and citations on published papers. Additionally, our research revealed that the process of the new population-based nature-inspired algorithm possesses the behavior of the swarm intelligence paradigm. Thus, it would not be easy to stop the invasions of the new population-based nature-inspired algorithms in the near future (only a systematic approach can help). However, awareness of the research community will help drastically in preventing the emergence of new population-based nature-inspired algorithms on new proposal attempts and make this research area clean again. Finally, the evolution of everything has not been finished in one night, but it took a lot of time. Eventually, it could also be the same for population-based nature-inspired algorithms.

9. REFERENCES

[1] H.-G. Beyer and H.-P. Schwefel. Evolution strategies–a comprehensive introduction. *Natural computing*, 1(1):3–52, 2002.

[2] P. Clapham. Publish or perish. *BioScience*, 55(5):390–391, 2005.

[3] T. D. Cosco. Medical journals, impact and social media: an ecological study of the twittersphere. *Canadian Medical Association Journal*, 187(18):1353–1357, 2015.

[4] M. Dorigo and T. Stützle. Ant colony optimization: overview and recent advances. *Techreport, IRIDIA, Universite Libre de Bruxelles*, 2009.

[5] I. Fister Jr., X.-S. Yang, I. Fister, J. Brest, and D. Fister. A brief review of nature-inspired algorithms for optimization. *Elektrotehniški vestnik*, 80(3):116–122, 2013.

[6] S. Fong, X. Wang, Q. Xu, R. Wong, J. Fiaidhi, and S. Mohammed. Recent advances in metaheuristic algorithms: Does the makara dragon exist? *The Journal of Supercomputing*, pages 1–23, 2015.

[7] S. Fong, R. Wong, and P. Pichappan. Debunking the designs of contemporary nature-inspired computing algorithms: from moving particles to roaming elephants. In *2015 Fourth International Conference on Future Generation Communication Technology (FGCT)*, pages 1–11. IEEE, 2015.

[8] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549, 1986.

[9] F. Glover. Tabu search–part i. *ORSA Journal on computing*, 1(3):190–206, 1989.

[10] F. Glover. Tabu search–part ii. *ORSA Journal on computing*, 2(1):4–32, 1990.

[11] F. Glover and K. Sörensen. Metaheuristics. *Scholarpedia*, 10(4), 2015.

[12] N. Hall. The kardashian index: a measure of discrepant social media profile for scientists. *Genome Biology*, 15(7):1–3, 2014.

[13] A. K. Jain, J. Mao, and K. M. Mohiuddin. Artificial neural networks: A tutorial. *IEEE computer*, 29(3):31–44, 1996.

[14] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948. IEEE, 1995.

[15] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simmulated annealing. *Science*, 220(4598):671–680, 1983.

[16] J. R. Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

[17] R. K. Merton et al. The matthew effect in science. *Science*, 159(3810):56–63, 1968.

[18] M. Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[19] G. Parchomovsky. Publish or perish. *Michigan Law Review*, 98(4):926–952, 2000.

[20] M. Perc. The matthew effect in empirical data. *Journal of The Royal Society Interface*, 11(98):20140378, 2014.

[21] A. P. Piotrowski, J. J. Napiorkowski, and P. M. Rowinski. How novel is the "novel" black hole optimization approach? *Information Sciences*, 267:191–200, 2014.

[22] K. Sörensen. Metaheuristics–the metaphor exposed. *International Transactions in Operational Research*, 22(1):3–18, 2015.

[23] K. Sörensen, M. Sevaux, and F. Glover. A history of metaheuristics. In *ORBEL29–29th Belgian conference on Operations Research*, 2015.

[24] D. Weyland. A rigorous analysis of the harmony search algorithm: how the research community can be misled by a "novel" methodology. *International Journal of Applied Metaheuristic Computing*, 1(2):50–60, 2010.

[25] D. Weyland. A critical analysis of the harmony search algorithm–how not to solve sudoku. *Operations Research Perspectives*, 2:97–105, 2015.

[26] X. Yao, Y. Liu, and G. Lin. Evolutionary programming made faster. *IEEE Transactions on Evolutionary computation*, 3(2):82–102, 1999.

Visualization of cycling training

Dušan Fister
Univerza v Mariboru
Fakulteta za strojništvo
Smetanova 17, 2000 Maribor
dusan.fister@student.um.si

Iztok Jr. Fister
Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko
Smetanova 17, 2000 Maribor
iztok.fister1@um.si

Iztok Fister
Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko
Smetanova 17, 2000 Maribor
iztok.fister@um.si

ABSTRACT

In the era of big data, a flood of information obtained from pervasive computing devices is being available. Using latest data mining technologies, lots of information can also be processed. Since many of them do not present any significant meaning to users, information could be ranged in so-called significant classes, where these are selected according to their significance. This paper presents an analysis of data obtained during the cycling training sessions made by wearable sports tracker and later visualization of their most significant parameters. Every sports training activity should be presented in a simple, self-speaking and understandable figure, from which it is possible to deduce difficulty, strain, effort, power, conditions and pace of the visualized sports training session.

Keywords

visualization, cycling, cycling elements, .TCX

1. INTRODUCTION

A use of GPS sports trackers (trackers) for sports training purposes in cycling is increased every day. More and more athletes use trackers to measure, accompany and control data obtained during their trainings, as well as perform later analysis and online planning. Trackers are today embedded into sports-watches (e.g. Garmin Connect, Strava, Movescount, Endomondo, Sports tracker, etc.) or mobile devices offering an upload of datasets with tracked data to the mentioned sports tracker producer websites. Every cyclist collects his activity datasets in a so called calendar, showing daily training sessions.

Sport-watches and mobile devices provide a brand-new approach of training, not only for cycling, but also for running, swimming, canoeing, hiking, roller skating, skiing, fitness, and other sports. Some of the specialized trackers support additional functions, like summing the number of daily steps and predicting calories burnt (therefore measuring the daily consumption), analysing daily weight and measuring average sleep quality by measuring movement of the arm [5].

All those properties, which became accessible to every athlete, lead to huge and complex online trackers, offering lots of data. But these data should be correctly interpreted in order to became useful information for an athlete. In line with this, different applications for visualization in sports have been emerged. For instance, the more frequently used visualization tasks are described in [7], but these are especially suited for the team sports. The analysis of the cycling data is well described in [3], while the advanced data-mining approach in [6]. The cycling training sessions and its planning can be seen in [4].

This paper proposes visualization of cycling elements, where athlete's data obtained from an archive of the activity datasets are interpreted. Based on athlete's effort, specific figures are built, from which it is possible to obtain the most significant class information about the performed sports training activity.

The organization of the paper is as follows. In second chapter, basic elements of cycling training are described in details. Third chapter offers background and description of proposed visualization, while the fourth chapter deals with results. The paper ends with conclusions and outlines the directions for future work.

2. ELEMENTS OF CYCLING TRAINING AND VISUALIZATION BACKGROUND

In this paper, we are focused on visualization of cycling training datasets. Cycling datasets are, as mentioned in Chapter 1, created during a sports training session. Cycling trackers usually record properties from the most significant class consisting of:

- position,
- distance,
- duration,
- velocity,
- altitude,
- temperature,
- heart rate and

- power.

First six properties are recorded by tracker itself when wearing or carrying during the activity, while least two are provided by wearing accessories. Heart rate monitor and power meter crank are today's classic equipment for cyclists and are usually being used during the training. Data is commonly recorded every second into special .GPX or .TCX dataset. This means that many useful information (e.g., different graphs, coefficients, average values, fatigue prediction, feeling and performance evaluation) can be extracted from the activity datasets when interpolating data through time. Tracker analysers usually compete against each other to provide most data for athletes. Therefore, more and more information are accessible when updating the tracker (extracting data is still in research).

The results of analyser are forwarded to visualizer, whose main task is to interpret them. For specific athlete, data should be treated specifically, i.e., no generalization is desired. For correct interpreting, cyclist needs to collaborate with visualizer's personal dataset in order to properly range data for each athlete. One of the tasks of visualizer is teaching himself in the matter of improving athlete's performance. Therefore, regular updates of personal dataset are crucial before the visualization. Correctly updated visualizer should then correctly predict cyclist's efforts, no matter of age, experiences or personal habits. Following values are specified in personal dataset:

- daily norm of duration,
- daily norm of distance,
- daily norm of altitude,
- minimum and maximum velocity of an athlete,
- minimum and maximum hearth rate and
- Functional Threshold Power (FTP).

First three properties depend on the athlete's desire to what he wants to reach, while last three show his objective performance. FTP setting of power is a property, which describes athlete's ability to produce the specific amount of power in one hour.

For visualization figure, it is common to show athlete's daily feeling. Therefore, the cyclist should enter into personal dataset his/her feeling, or condition, on the day of training. Thus, three different feeling badges can be selected, i.e., good, bad and medium. Additionally, weather data can be downloaded from weather websites and added to visualization figure. In order to commence research and implementation of automatic visualizer, the prototype was manually schemed in the picture drawing studio (Fig. 1).

Fig. 1 presents the prototype of visualization designed in figure editing studio that serves as a basis for automatic visualization. It is worth to mention that cycling parameters shown in prototype are untrue and biased, but purpose can be clearly seen from it. Figure should be read by the color

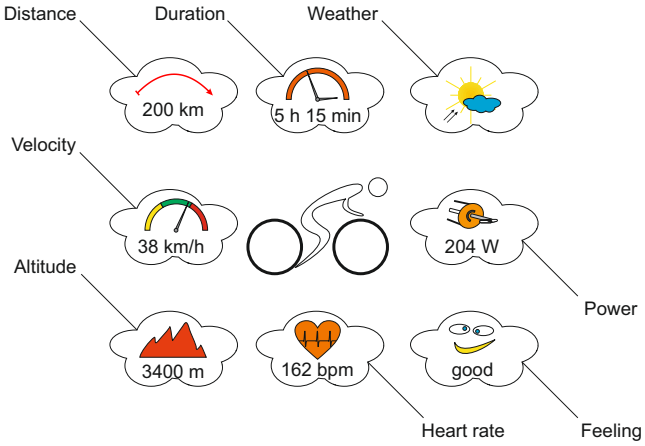


Figure 1: Visualization elements of the prototype.

of the cycling elements, as well as with an objective value. The curve representing the distance currently in bright red means that athlete almost reached the wished daily norm. The number below means total distance cycled and serves as supporting value of visualization. Speed meter represents velocity and is divided into three sections, presenting relaxation ride, tempo ride and competition ride. A deterministic value is added as well in order to prevent the misconception of reading the velocity meter slope. The same applies to duration, altitude, heart rate and power meter element. Colors are automatically chosen and adjusted to athlete's performance. They vary between bright yellow, as low-effort, and dark red, as high-effort (Fig. 3).

Fig. 2 presents the flow of events and data graphically. Starting by cycling, process continues to uploading and analysing of recorded data. Transformations from real to computer world are clearly seen as well as visualizing section with appropriate displaying results to cyclist.

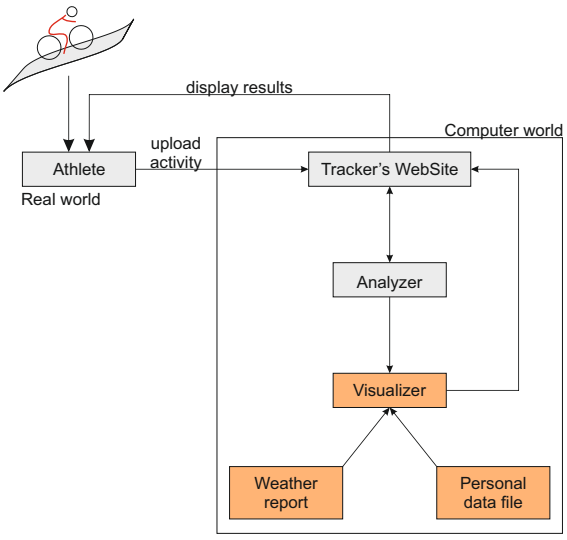


Figure 2: Visualization process.

As can be seen from Fig. 2, the visualization process is divided into three parts:

- completing the training session,
- uploading and processing data, as well as visualizing results,
- informing cyclist.

The first part is dedicated to the cyclist in real world. After completing his training, recorded data is transmitted to computer world in the second part. There, they are being processed, visualized and displayed on the website as a result. In third part cyclist can inform himself about the training's effort and therefore struggle even more next time.

3. CYCLING TRAINING ELEMENTS VISUALIZATION

The visualization process is performed using a software suite ImageMagick and Ruby programming language [2]. ImageMagick is the open source text-mode editing software for pictures, which excellently collaborates with the Ruby language. This combination of software was applied in our study because of high adaptability, simplicity and integration. Algorithm 1 presents the basic visualization process.

Algorithm 1 Visualization principle

1: Athlete uploads cycling activity dataset to tracker's website;
2: Analysis of uploaded dataset begins;
3: Analyser forwards extracted data to visualizer;
4: Visualizer reads data and retrieves personal dataset;
5: **if** Personal dataset should be updated **then**
6: Update personal dataset;
7: **end if**
8: Visualizer downloads actual weather report;
9: Visualizer interprets data to become information;
10: Visualizer generates a figure and forwards it to website;
11: Website presents the visualized figure of the training activity session;

Algorithm 1 presents the standard visualization process, taking into account both of the accessories (i.e., heart rate monitor and power-meter) by default. It should be noted, that the visualization process is adjusted, when cyclist does not use any of them and visualization figure is different than shown one (Fig. 1).

3.1 Visualization of distance, altitude, duration, heart rate and power

Let's say, that analysis part has been completed and extracted data are ready to be interpreted. Interpretation actually means comparing extracted data from tracker and values obtained from personal dataset. An example shows the comparison for the distance element:

$$INTERPRET\_DISTANCE = \frac{actual\_distance}{max\_distance}, \quad (1)$$

where $INTERPRET\_DISTANCE \in [0, 1]$.

After obtaining result $INTERPRET\_DISTANCE$, which lies in the interval [0,1] a color for the distance curve is selected. Ten colors are available for ranging the intensity of cycling elements. If the result of distance is $INTERPRET\_DISTANCE = 0.45$, then the corresponding color in interval $[0.4, 0.5]$ will be selected, i.e., bright orange as can be seen from Fig. 3.

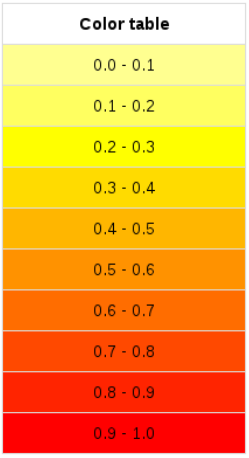


Figure 3: Color table.

Interpretation of altitude and duration is performed on the very similar way, while interpretation of heart rate and power is a bit more sophisticated. It should be mentioned, that it would be silly to interpret, for instance heart rate $HR = 56$ bpm on the training, because this can mean that cyclist did not actually move. To prevent appearance of such errors, we suggest to add besides the set of maximum values also the set of minimum values. Consequently, actual data should be spread between minimum and maximum values, e.g. heart rate should be drawn from interval $HR \in [120, 180]$ bpm. Accordingly, Eq. (1) transforms into Eq. (2):

$$INTERPRET\_POWER = \frac{actual\_power - min\_power}{max\_power - min\_power}, \quad (2)$$

where $INTERPRET\_POWER \in [0, 1]$. If the $actual\_power$ is lower than min_power, the min_power is taken into account and $actual\_power$ is disregarded.

As a result, listed elements are colored in the appropriate color and finally added (imported) into the primary figure.

3.2 Visualization of velocity

As seen in prototypical Fig. 1, velocity, weather and feeling are not presented by color, as other elements. Therefore, a different approach was studied for those elements. One of the most self-explaining principles of presenting velocity in general is using a simple indicator with three background arc colors. In fact, reading the slope of the indicator is the easiest way to determine velocity and that is the reason for employing it in our application. Background arc colors can be pre-drawn and pre-imported into file to make whole process easier, but indicator has to be programmed and processed at the moment of visualization, indeed.

The scheme of velocity indicator consists of three mutually connected points. Connections create so-called polygon, which has to be properly positioned and rotated to express wanted velocity. Position can be fixed, due to fixed color arc in primary figure, but rotation is a matter of two read values, written in personal dataset - expected minimum and maximum value. Those extend the velocity range, e.g. 25 km/h as the lowest velocity and 45 km/h as the highest velocity. As stated, those conditionals call for the automatized calculation of indicator's tip position. In Ruby, programming of indicator and its movement was executed by simple mathematical equation, consisting of trigonometrical functions. First, the slope in degrees is calculated using following equation:

$$SLOPE = \left(actual\_velocity - \frac{min\_velocity + max\_velocity}{2}\right) \cdot 9^{\circ}, \quad (3)$$

where *actual_velocity* is parsed from .GPX, or .TCX file and *min_velocity* and *max_velocity* read from personal dataset. 9° is used as section interval, meaning that 1 km/h presents 9° in inaugural form and is updated if updating personal dataset.

After calculating the needed slope, the tip's position is being calculated using trigonometrical functions in following equations:

$$x\_position = \sin(SLOPE) \cdot 202.5, \quad (4)$$

$$y\_position = \cos(SLOPE) \cdot 202.5, \quad (5)$$

where 202.5 presents the length of the indicator from the center of color arc to the tip end in pixels (Fig. 4).

Figure 4: Velocity indication.

3.3 Visualization of weather and feeling

Visualization of weather is divided into two parts. First part presents the general weather conditions - sunny, rainy, cloudy weather, while the second part deals with the magnitude and direction of wind. Data is downloaded and parsed from the aviation weather's site, called Ogimet [1], which offers aviation weather reports, called METAR (Meteorological Aviation Report). The METAR is a simple, text-based weather report, from which weather conditions can be determined. For us, it is particularly important to be aware of clouds and possible wind blowing, therefore only little of report is parsed. To get proper weather report from website it is also necessary to locate closest weather station to activity position, what is still under research.

Meanwhile, weather condition figures are drawn in advance, so only compositing of them to the right place is necessary to expose the weather conditions. If sunny weather is reported, only sun is exposed and opposingly, cloudy weather is seen from figure, if clouds are described in METAR (Fig. 5):

Figure 5: Weather conditions.

Fig. 5 shows basic weather conditions, which serve as the first part in describing weather. Describing the wind, or second part, is more sophisticated. First, the direction and magnitude are gathered from weather report. Next, the length of the wind flag is initialized. Position of the wind flag tip and its end are calculated to continue the calculation process and finally the line between them is outlined. To certainly specify wind direction, the wind flag's ending tips are drawn (Fig. 6). The calculus part consists of many trigonometrical function and is very long, therefore we excluded it from paper.

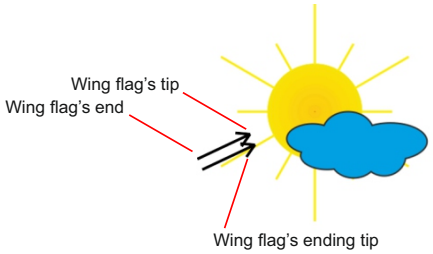


Figure 6: Describing wind.

Three different magnitudes of wind are applicable in our work:

- wind velocity from 0 km/h to 5 km/h: no wind flag,
- wind velocity from 5 km/h to 15 km/h: one wind flag,
- wind velocity from 15 km/h and above: two wind flags.

Visualizing feeling bases on the athlete's manual input. Three different feelings are included in program, which vary from bad, medium and good (Fig. 7). As standardized, all three figures are drawn into advance and imported into primary figure on the appropriate position after cyclist's decision.

4. RESULTS

Results of cycling's automatized visualizing are presented graphically for three different training stages:

- after-season relaxation,

Figure 7: Describing feeling.

- mileage training and
- competition.

Result are constructed on a single cyclist, who had fulfiled his personal dataset based on his experiences and uploaded three different .TCX files to website. They have all been analysed and sent to our visualizer. It then generated three figures, which suppose to be shown in the athlete's calendar of activities.

4.1 After-season relaxation

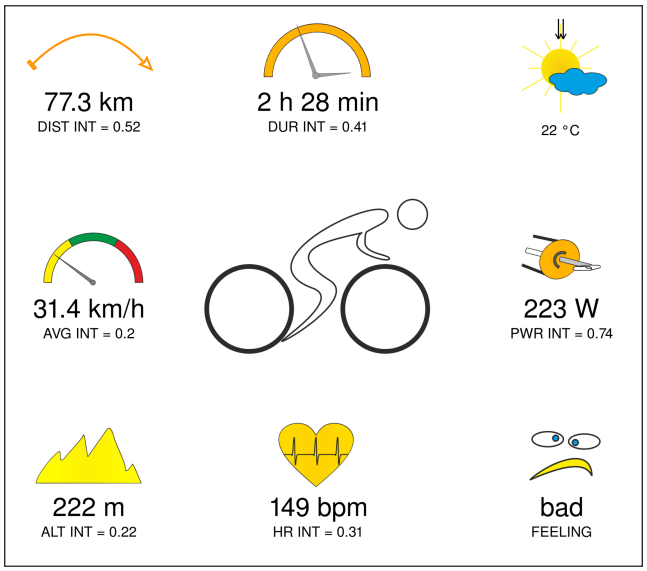


Figure 8: After-season relaxation.

Fig. 8 presents after-season relaxation training session. Cyclist did quite short training, what is seen from distance and duration. Its colors are orange, meaning that athlete only did half of his maximum. Cyclist did rode slowly, with low pace, what can be also seen from his pulse. His power was quite good, considering the flat ride (low altitude) and bad feeling. Cyclist probably did a short, regenerative training in order to raise his feeling and performance for next day's training, which will probably be much more difficult.

4.2 Mileage training

Mileage training is a long, distant training, seen from distance and duration elements on Fig. 9. Cyclist did train more than 100 km more than in first result. Cyclist did much altitude, due to red colored altitude indication. The weather on that way was cloudy, without wind. His feeling was medium and his heart rate like expected - in the

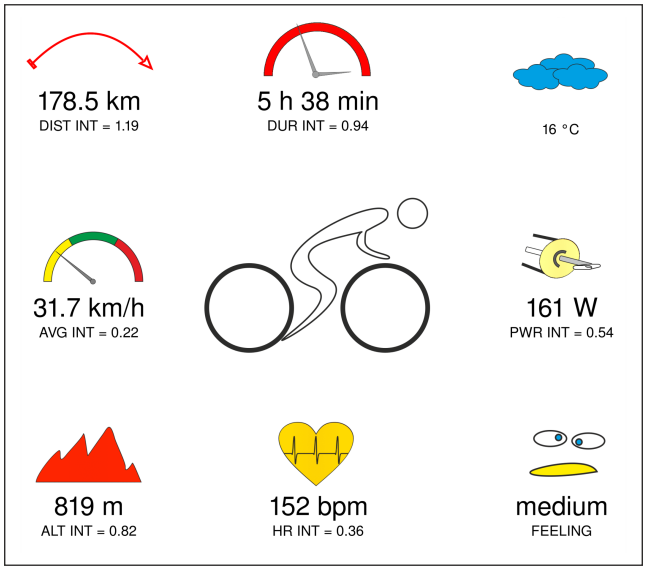


Figure 9: Mileage training.

strong yellow section to save his energy for the whole ride. The answer on the question, which asks why did the cyclist reach higher power in after-season relaxation than in mileage training is a bit complex. First, the power shown on the visualization figure is not average, but normalized (NP). NP is a correction of average power, due to rapid changes in training intensity and due to curvilinearly physiological responses [8]. In the first result, athlete probably did some intervals (escalating intensity) during the training to stretch his legs, which are rapid changes in intensity, and therefore improved his NP shown in the figure. Practically, rapid intervals on the training make NP higher. Oppose to first result, at this long ride, athlete did save with his energy and did no short intervals and no intensifying is observed from Fig. 9.

4.3 Competition

Competition visualization is as at first seen very relaxational (Fig. 10). Athlete did only little more than twenty kilometres, getting the distance curve barely noticed. His ride last for only half an hour, identifying yellow color from duration element. But otherwise, athlete had struggled at most by other three results - his average heart rate was 182 bpm, normalized power 294 watts and velocity 43.2 km/h. They are all in the red section, meaning that athlete surely competed at the time trial. He chose flat course, having only few altitude, therefore getting his velocity very high. His feeling was bad, at strong south-west wind and some clouds, meaning that cyclist could drive even faster.

5. CONCLUSION

In this paper, we presented a novel approach for visualizing most important cycling data. Accordingly, a visualizer in editing studio ImageMagick was implemented and controlled by Ruby programming language. We practically executed our approach and showed, that it is possible to automatically generate expected figures. Results of the performed work are, due to rapid visualizing (only five seconds per figure), excellent. The precision of process is valuable and resolution of figures is acceptable.

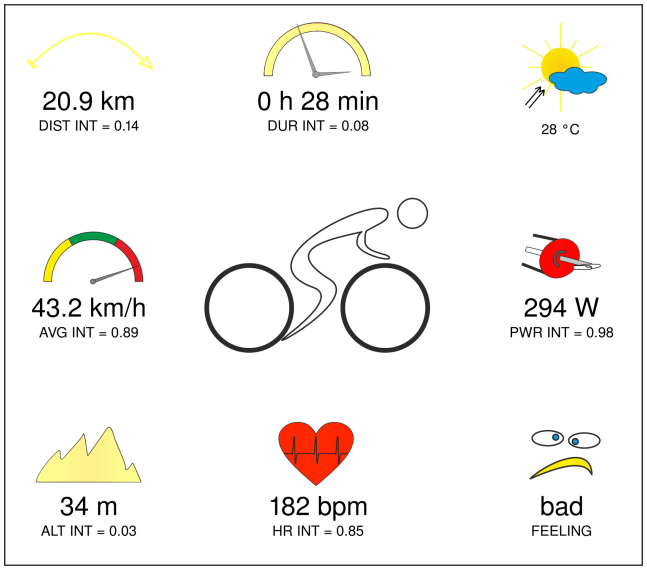


Figure 10: Competition.

However, some cues remain for the improving the paper:

- some graphical weather elements should be added, e.g. symbol for rain, snow and fog,
- standardized visualization scheme should be transformed into an adjusted one, if athlete does not own a heart rate monitor, or power meter during the training,
- the default feeling should be set to good by default and changed only after athlete's intervention,
- drawn symbols, which represent altitude, heart rate, weather symbols, power meter and feeling should be completely created in the ImageMagick studio (currently they are drawn in advance and imported into primary figure),
- automatic parse for weather should be implemented,
- sports, like running, swimming and roller-blading should be added into visualization with their elements (Fig. 11) and
- analyser should be implemented to read data from trackers.

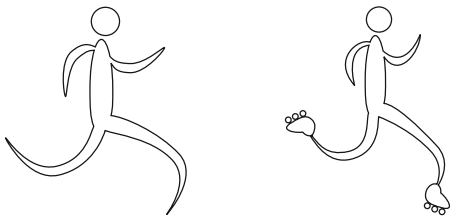


Figure 11: Running and roller-blading.

The evaluation of cycling data visualization was satisfactory implemented. It is worth to continue research in this way, since it may help to many cyclists.

6. REFERENCES

[1] Ogimet METAR. `http://www.ogimet.com`, 2005. [Accessed: 2016-08-31].

[2] Dan Nguyen. Image manipulation. `http://ruby.bastardsbook.com/chapters/image-manipulation/`, 2011. [Accessed: 2016-08-25].

[3] I. Fister, D. Fister, and S. Fong. Data mining in sporting activities created by sports trackers. In *Computational and Business Intelligence (ISCBI), 2013 International Symposium on*, pages 88–91. IEEE, 2013.

[4] I. Fister, S. Rauter, X.-S. Yang, and K. Ljubič. Planning the sports training sessions with the bat algorithm. *Neurocomputing*, 149:993–1002, 2015.

[5] Garmin. Sleep tracking. `http://www8.garmin.com/manuals/webhelp/forerunner230/EN-US/GUID-70D41BFB-2BB2-4933-BF95-47FF63140112.html`, 2014. [Accessed: 2016-08-25].

[6] G. Hrovat, I. Fister Jr, K. Yermak, G. Stiglic, and I. Fister. Interestingness measure for mining sequential patterns in sports. *Journal of Intelligent & Fuzzy Systems*, 29(5):1981–1994, 2015.

[7] M. Page and A. V. Moere. Towards classifying visualization in team sports. In *International Conference on Computer Graphics, Imaging and Visualisation (CGIV'06)*, pages 24–29. IEEE, 2006.

[8] Training Peaks. Normalized power, intensity factor and training stress score. `http://home.trainingpeaks.com/blog/article/normalized-power,-intensity-factor-training-stress`, 2005. [Accessed: 2016-08-31].

Izdelava klaviatur MIDI

Primož Bencak
Univerza v Mariboru
Fakulteta za strojništvo
Smetanova 17, Maribor
primoz.bencak@student.um.si

Dušan Fister
Univerza v Mariboru
Fakulteta za strojništvo
Smetanova 17, Maribor
dusan.fister@student.um.si

POVZETEK

Glasba obstaja, odkar obstaja človek. Vedno je bila človeku prijazna sopotnica ali pa sredstvo za sprostitev, umiritev in motivacijo. Skladno z glasbo in človekom se razvijajo, odraščajo in spreminjajo tudi glasbeni instrumenti, ki ustvarjanju glasbe dajejo poseben pečat. Zadnja leta na razvoj glasbene industrije vpliva predvsem elektronska industrija, zato se v tej študiji osredotočamo na izdelavo klaviatur, tj. glasbila s tipkami, po glasbenem standardu MIDI (angl. Musical Instrument Digital Interface). Predlagane klaviature MIDI omogočajo povezljivost z računalnikom, zato so zasnovane na osnovi mikrokrmilnika podjetja Texas Instruments in mikroračunalnika Raspberry Pi. Slednja na račun vedno večje integracije družno omogočata procesorsko moč osebnega računalnika in neposredno povezljivost s strojno opremo.

Ključne besede

Mikrokrmilnik, mikroračunalnik, MIDI standard, klaviature, sinteza zvoka

1. UVOD

Glasbo je mogoče ustvarjati na mnogo načinov. Do osemdesetih let prejšnjega stoletja so bili posamezni sklopi zvokov omejeni na določen glasbeni inštrument, v novejšem času pa se je pojavila ideja glasbene naprave s tipkami, ki lahko sintetizira (umetno predvaja) celotno paleto zvokov, celo takšnih, ki jih s tradicionalnimi inštrumenti ne moremo dobiti. Ideja o sintetizatorju, tj. napravi, ki s pomočjo vnosnega modula (tipkovnice) zbira podatke o tem, kdaj in katera tipka je bila pritisnjena, ki se na podlagi teh podatkov odzove z določenim zvokom [10], je spremenila potek glasbene industrije. Od tedaj naprej glasbeniki in producenti iščejo načine, kako sintetizatorju dodati nove zvoke, ali jih izboljšati.

Digitalni vmesnik glasbenih instrumentov, oz. krajše MIDI, je standard, ki je bil ustvarjen zaradi potrebe po medsebojni kompatibilnosti med digitalnimi sintetizatorji različnih proizvajalcev [9]. Formalno je bil sprejet leta 1983 in je vse do danes v glasbeni industriji ostal najbolj uporabljan standard. Omogoča medsebojno povezljivost različnih vmesnikov MIDI, ne da bi za to potrebovali dodatno strojno opremo. MIDI je zapis informacij o tem, katera nota oz. tipka je pritisnjena, na kak način, in s kakšno silo.

Ustvarjanje zvoka s pomočjo električne energije je vsekakor široko področje, ki sega daleč v zgodovino. V tem poglavju navajamo nekaj izvorov razvoja analognih in digitalnih sintetizatorjev. Griffis je 1984 [4] patentiral vezje, ki signale zvoka predvaja na stereo zvočnem analognem sintetizatorju. Elektronski sintetizator je leta 1988 patentiral Japonec Kaneoka [6], medtem ko je McDonald šest let kasneje predlagal uporabo zvočnega sintetizatorja v vozilih [7]. Razvoj digitalnega sintetizatorja sega že v leto 1990, ko je Gilmore patentiral, t.i. direktni digitalni sintetizator [3]. Leta 1999 je Cook predlagal, t.i. orodje za sintezo zvoka (angl. Synthesis toolkit) in razložil pomen standarda MIDI za ustvarjanje zvoka [2]. Princip analognega in digitalnega sintetizatorja je omogočil razvoj številnih izpeljank sintetizatorjev, od katerih je vredno podrobneje omeniti vzorčevalne, saj smo slednji princip uporabili tudi mi. Pregled vseh izpeljank je zapisan v [8].

V sklopu našega projekta želimo izdelati preprost sintetizator, tj. klaviature, ki posnemajo delovanje klavirja. Priporočljivo je, da so klaviature uporabne za vajo in snemanje zvoka, da ustrezajo standardu MIDI [1], [5] in da so čim enostavnejše za uporabo. Praktično to pomeni, da morajo delovati kot samostojna, baterijsko napajana naprava brez povezave z zunanjim svetom in morajo biti prenosne. V tem primeru jih namreč lahko uporabimo tudi na koncertu, kar je glavna motivacija našega dela.

Struktura članka v nadaljevanju je naslednja. Drugo poglavje govori o problemu izdelave klaviatur MIDI, tretje poglavje opisuje postopek izdelave klaviatur in četrto predstavlja rezultate. Članek zaključimo s predstavitvijo možnih izboljšav našega dela v prihodnje.

2. SINTETIZATORJI ZVOKA IN KLAVIATURE MIDI

Sintetizator zvoka uporabniku omogoča ustvarjanje in izvajanje najrazličnejših zvokov. Obstaja več vrst sintetizatorjev, od najosnovnejših analognih do nekoliko kompleksnejših digitalnih, npr. vzorčevalnih (angl. sampler), dodajnih (angl. additive) in odvzemnih (angl. subtract) sinte-

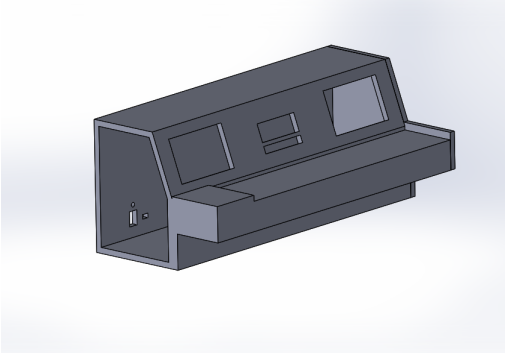
tizatorjev. V začetku šestdesetih let prejšnjega stoletja so se pojavili prvi zametki analognih sintetizatorjev zvoka. Ti so bazirali na osnovi analognih elektronskih komponent in so želen zvok generirali s sinusnim, pravokotnim, žagastim ali trikotnim generatorjem signala, kakor tudi pripadajočimi filtri ter generatorji šuma. Predstavljali so odskočno desko za razvoj naprednejših, digitalnih sintetizatorjev, ki so se pojavili z razvojem mikroprocesorjev in digitalnih vezij. V tem času se je pojavila ideja o digitalni sintezi zvoka in spoznanju, da lahko veliko število dragih analognih komponent nadomestimo z računalniškim algoritmom. Ta lahko opisuje barvo, višino, glasnost in trajanje zvoka digitalno. V glasbeni srenji se poleg navedenih pojavljajo tudi nekoliko zahtevnejši opisi zvoka in not, kot so:

- zavitost note (angl. pitch up, pitch down) - višina note se s časom spreminja,
- transponiranje note - nota je zvišana ali znižana za določeno višino (npr. pol oktave) in
- zadrževanje note (angl. sustain) - ohranjanje zvenenja note, četudi tipka ni več pritisnjena.

Da dosežemo zavitost note, je potrebno poleg klaviaturske tipke pritisniti tudi tipko, ki spreminja zavitost note, npr. navzgor. To pomeni, da mora uporabnik za določen čas, v katerem nota spreminja višino navzgor držati dve tipki. Enak princip velja tudi za transponiranje in zadrževanje note. Da ju vklopimo, je potrebno za določen čas držati dve tipki, kar spretno zapisuje standard MIDI. Oktava je razdalja med osmimi zaporednimi toni (npr. med prvim in osmim tonom), pri čemer ima vsak višji ton dvakrat višjo frekvenco od nižjega tona.

Pojav prvih digitalnih sintetizatorjev je zaradi manjšega števila elektronskih komponent omogočal cenejši pristop generiranja zvoka. A je poleg tega ta pristop omogočal tudi manjšo izbiro različnih tonov zvoka. Prav tako so se pojavljale neželene izgube pretvorbe iz analognega v digitalni format (in obratno). Dandanes analogne generatorje uspešno zamenjujejo digitalni signalni procesorji (DSP) ter kontrolne plošče s pripadajočimi prikazovalniki, kakovost zvoka pa presega zmožnosti tedanjih sintetizatorjev. Slednje uporabniku omogočajo spreminjanje nastavitev v realnem času in izbiro različnih zvokov, kakor tudi sporočajo informacijo o trenutnem stanju. Generiran signal je nato speljan do ojačevalnika, ki signal ojači in pošlje do zvočnika.

Vzorčevalni sintetizatorji temeljijo na principu vnaprej shranjenih posnetkov določenih zvokov. Ko program ali naprava zazna, da je bila pritisnjena tipka, jo nadomesti z zvokom, ki je shranjen v pomnilniku. Po tej klasifikaciji uvrščamo naše klaviature MIDI med vzorčevalnike (samplerje), saj naprava zvoka ne sintetizira, ampak le predvaja (slika 1). Zvoki so vnaprej posneti vzorci glasbil ali glasov in so shranjeni v pomnilniku mikroračunalnika Raspberry Pi 2 z naloženim operacijskim sistemom Raspbian. Kakovost reproduciranega zvoka zato neposredno določa kakovost posnetega vzorca, kar je njihova največja slabost. Prednost vzorčevalnih sintetizatorjev je v tem, da trošijo malo procesorske moči, saj je za predvajanje potrebno le prenesti podatke iz začasnega pomnilnika in jih povezati s pritisnjeno tipko.



Slika 1: Zasnovane klaviature MIDI.

2.1 Standard MIDI

Da naprava ustreza standardu MIDI, mora upoštevati nekaj sistemskih zahtev. Vsak instrument, ki je zmožen upravljanja s standardom MIDI, mora imeti oddajnik in sprejemnik. Vmesnik mora delovati na asinhronski (serijski) povezavi s hitrostjo 31.25 Kbit/s in prenaša tri bajte podatkov, tj. statusni bajt, podatkovni bajt 1 in podatkovni bajt 2 (tabela 1). Naj omenimo še, da je iz poslanih podatkov za spremljavo zvoka možno ustvarjati tudi osvetlitev in svetlobne efekte.

Zaradi njegovega enostavnega principa sintetiziranja zvoka in prenašanja not ga lahko enostavno implementiramo na mikrokrmilnikih. Potrebno je poznati le nabor ukazov, ki so podani kot binarni zapisi števil, npr. zapis za aktivacijo in deaktivacijo note v tabeli 1.

3. IZDELAVA KLAVIATUR MIDI

Izdelave klaviatur smo se lotili z načrtovanjem funkcij klaviatur, ki so:

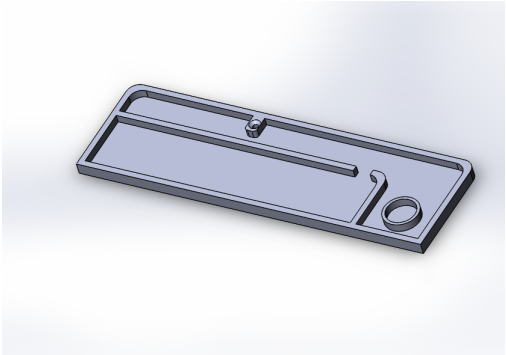
- predvajanje (reprodukcija) osnovnih zvokov,
- prenosljivost in avtonomija (baterijsko napajanje, vgrajen računalnik),
- zanesljivo delovanje,
- čim enostavnejša izvedba,
- nizka latenca (čas med pritiskom na tipko in predvajanjem zvoka) in
- predvajanje zvoka naj bo opravljena v ohišju klaviatur (vgrajena naj bosta tudi ojačevalec in zvočnik).

Prva alineja, tj. reprodukcija osnovnih zvokov predstavlja zmožnost predvajanja zvokov klavirja, kitare in bobnov. Prenosljivost ter avtonomija baterije (čas delovanja baterije, kateri znaša vsaj osem ur) omogočata, da lahko uporabnik sintetizator uporablja tudi na koncertu. Tretja in četrta alineja sodelujeta z roko v roki, zato želimo sintetizator izdelati s čim manj elementi, medtem ko se posledici pete alineje, tj. zakasnitve (angl. latency) ne moremo izogniti. Zadnja alineja podaja kompaktnost, oz. končno obliko sintetizatorja.

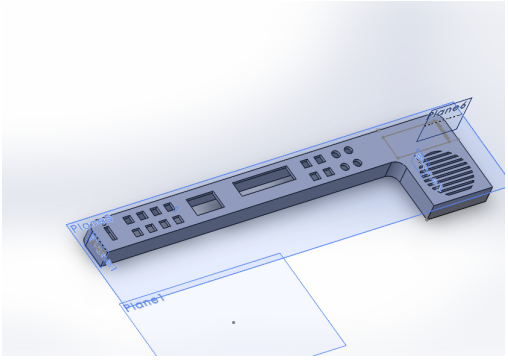
Postopek izdelave smo razdelili v več faz:

- načrtovanje 3D-modela ohišja in klaviaturskih tipk v programu SolidWorks,
- načrtovanje elektronskih vezij,
- izdelava modela in jedkanje elektronskih vezij,
- sestavljanje fizičnega modela, klaviaturskih tipk in elektronskih vezij v celoto,
- programiranje,
- testiranje in odpravljanje težav in
- vrednotenje dela.

Fazi načrtovanja 3D modela in tipk ter elektronskih vezij sta najpomembnejši izmed vseh. To pomeni, da smo jima namenili posebno pozornost. Skrbno smo določili vse dimenzije izdelka in material (slika 2 in slika 3). Klaviature smo izdelali iz lesa, saj je bila to cenovno najugodnejša rešitev. Cena izdelave ogrodja s pomočjo tiskalnika 3D, za katero smo se sprva sicer odločili, bi namreč večkrat presegla končne stroške lesa in spojnih elementov. Uporabili smo topolove vezane plošče debeline 8 mm, zato daje izdelek dodaten občutek trdnosti in estetske dovršenosti.



Slika 2: Zasnovano ohišje - spodnji del.



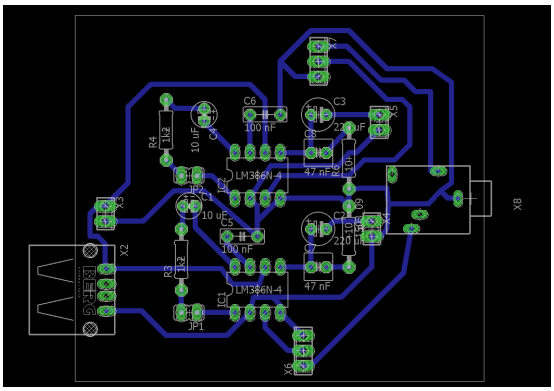
Slika 3: Zasnovano ohišje - zgornji del.

3.1 Izdelava vezij

Načrtovanje vezij smo izvedli s programoma za načrtovanje tiskanin, tj. Eagle in Target3001!. Izdelali smo tri vnosne module, s skupaj 36 tipkami, kar omogoča glasbeni razpon treh oktav. Širok razpon zvokov sicer ugodno vpliva na kakovost in uporabo klaviatur, vendar je potrebno omeniti problem omejenega števila vhodno-izhodnih priključkov na mikrokrmilniku MSP430F5529. V skladu s tem smo pred mikrokrmilnik namestili 16-kanalni analogni multiplekser HC4067. Slednji za šestnajst različnih vnosnih elementov (tipk ali potenciometrov) potrebuje samo štiri izhode in en priključek za analogno-digitalno (A/D) pretvorbo. Dodatni načrtovalski izziv je predstavljala nadzorna plošča, prek katere je moč nastavljati višino, glasnost ter zadrževanje not. Plošči smo prigradili tudi zaslon LCD. Zaradi različnih napajalnih napetosti LCD prikazovalnika (5 V enosmerne napetosti) in mikrokrmilnika (3.3 V enosmerne napetosti), smo uporabili invertersko vezje, tj. vezje ki generira negativno napetost. Invertersko vezje je priključeno na pulzno-širinski izhod mikrokrmilnika in na določen priključek prikazovalnika LCD, kjer ob pravilno nastavljeni širini pulza od 5 VDC odšteje ravno potrebnih 1.7 VDC. Tako smo dosegli dvosmerno komunikacijo, kar smo morali predvideti že v načrtovalski fazi.

Za zagotavljanje zanesljivosti smo kasneje zgradili še napajalni modul, kakor tudi stereo ojačevalno vezje, osnovano na vezju LM386. Ugotovili smo namreč, da je glasnost avdio signala iz mikroračunalnika Raspberry Pi nizka in da je zato gradnja ojačevalnega vezja neizogibna.

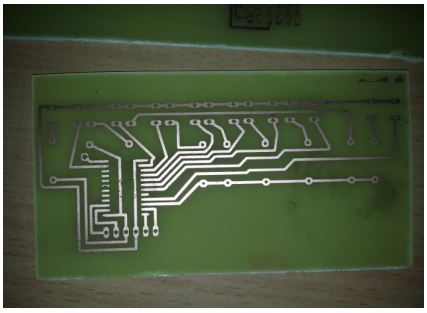
Načrtovano vezjo (slika 4) smo nato natisnili na ploščo Vitroplast. Vitroplast je plošča, izdelana iz polprevodniškega materiala, na katero je nanesena tanka plast bakra. Natisnjeno vezje in ploščo Vitroplast smo nato nekajkrat vstavili v vroč plastifikator. S tem smo dosegli, da se je načrtovano vezje zapeklo na tanko površino bakra. Zapečeno vezje smo nato z mešanico klorovodikove kisline ter vodikovega peroksida zjedkali. Ob jedkanju se neuporabljena plast bakra odstrani, zato na plošči ostane zgolj odtis električnega vezja. Vezje smo nato očistili z redčilom ter nazadnje izvrtali še predvidene luknje za spajanje elektronskih komponent. Slika 5 prikazuje jedkano in očiščeno vezje za tipkovnico.



Slika 4: Načrtovano vezje stereo ojačevalnika v programu Eagle.

Tabela 1: Kodiranje aktivacije in deaktivacije note.

Statusni bajt	Podatkovni bajt 1	Podatkovni bajt 2	Sporočilo
1000kkkk	0vvvvvvv	0ggggggg	Aktivacija note
1001kkkk	0vvvvvvv	0ggggggg	Deaktivacija note



Slika 5: Jedkano in očiščeno vezje ene izmed treh tipkovnic.

3.2 Sestavljanje klaviatur

Sestavljanje klaviatur smo začeli s sestavljanjem lesenega okvirja klaviatur. Ugotovili smo, da se tipkovnica težko prilagaja okvirju sintetizatorja, kar praktično nismo predvideli v načrtovalski fazi. Na levi strani čelne plošče smo namestili šest tipk, preko katerih lahko uporabnik spreminja nastavitve zvoka (zadrževanje note, višino, transponiranje,...). S pripadajočimi prikazovalniki LED (diodami) lahko uporabnik spremlja trenutno stanje nastavitev. Pod tipkami se nahajajo trije potenciometri, ki skrbijo za spreminjanje glasnosti, ojačanja vgrajenega ojačevalca in kontrasta LCD zaslona. Klaviaturam smo vgradili 1 W stereo ojačevalec ter dva zvočnika. Namestili smo tudi priklop za stereo izhod, ki ob preklopu klecnega stikala posreduje zvok iz vgrajenih zvočnikov v priklopljeno napravo. Za dodatno zaneljivost smo namestili še dva priklopa USB in sicer za baterijo in mikroračunalnik (ali kakšni drugi shranjevalni medij).

3.3 Programiranje

Za branje podatkov klaviatur smo uporabili mikrokrmilnik MSP430F5529 družine Texas Instruments. Program za mikrokrmilnik je v celoti napisan v programskem jeziku C v studiu IAR Embedded Workbench. Za ohranjanje preglednosti programa in omogočitev kasnejše nadgradnje smo program razdelili v številne knjižnice. Jedro programa tako predstavlja branje kanalov multiplekserja (tipk). Praktično to pomeni, da je glavna naloga mikrokrmilnika sprejemanje in obdelavo signalov.

Vsaka nota, ki jo predstavlja tipka, ima svojo lastno funkcijo v programu. Po pritisku slednje, A/D pretvornik mikrokrmilnika zazna spremembo napetosti in zaznane podatke pošlje mikroračunalniku. Pošiljanje podatkov poteka v obliki treh bajtov preko serijskega prenosa:

- v prvem bajtu je shranjen podatek o aktivaciji note (npr. število $144_{(10)}$ -> $010010001_{(2)}$ nosi ukaz o pritisku - aktivaciji note, medtem ko število $128_{(10)}$ -> $10000000_{(2)}$ o spustu - deaktivaciji),
- drugi bajt nosi informacijo o višini note, npr. število 0 predstavlja noto C0, število 128 pa noto G10 in
- tretji bajt nosi informacijo o glasnosti note pri čemer število 0 predstavlja minimalno glasnost in število 128 maksimalno glasnost.

Vse tri bajte podatkov lahko prikažemo tudi tabelarično. Tabela 1 opisuje kodiranje toka podatkov, ki se pojavlja med prenosom MIDI. Aktivacija (angl. Note on) note je predstavljena vnaprej. Opisana je s statusnim bajtom, katerega prvi štirje biti predstavljajo binarno vrednost 1000, zadnji štirje pa kanal MIDI (črka k). Namen kanala MIDI se skriva v istočasnem snemanju, oz. predvajanju več skladb MIDI, zato ima vsaka naprava rezerviran svoj kanal. Naprave, ki uporabljajo standard MIDI, namreč lahko medsebojno poljubno povežemo. Za deaktivacijo (angl. Note off) se binarna vrednost spremeni v 1001. Podatkovna bajta sta sestavljena iz note (črka n) in pritiska (črka p), oba variirata med vrednostmi 0-127, kar sovpada z vrednostjo 2^7.

Postopek aktivacije note prikazuje algoritem 1, medtem ko postopek deaktivacije note algoritem 2.

Algoritem 1 Serijski prenos aktivacije note

```
Serial.Init(); //naloži modul serijske komunikacije
Serial.Begin(9600); //začni s komunikacijo s hitrostjo 9600 bitov na sekundo
Serial.Write(144); //pošlji ukaz za aktivacijo note
Serial.Write(48); //nota naj ima višino C4
Serial.Write(50); //nota naj ima glasnost 50
Delay(500); //nota naj zveni 500 ms
```

Algoritem 2 Serijski prenos deaktivacije note

```
Serial.Write(128) //pošlji ukaz za deaktivacijo note
Serial.Write(48); //nota naj ima višino C4
Serial.Write(50); //nota naj ima glasnost 50
```

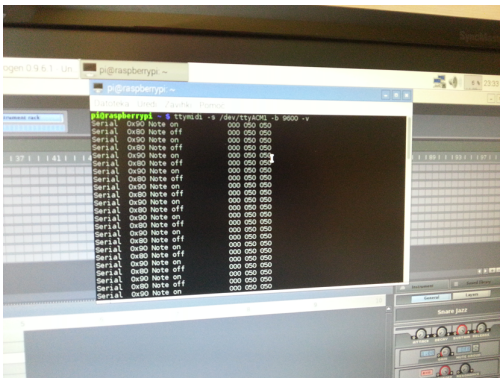
Algoritma 1 in 2 praktično prikazujeta serijski prenos podatkov iz mikrokrmilnika na mikroračunalnik. Sprva se inicializira povezava med obema, nato se pošlje ukaz za aktivacijo note, kateremu sledita višina in glasnost. Za izzvenjenje note mora mikrokrmilnik poslati ukaz za deaktivacijo note, sicer se nota še vedno predvaja.

Mikroračunalnik prejete podatke interpretira. Interpretacija poteka s programom ttyMIDI, ki je prednaložen na mikroračunalniku (slika 6). Program ttyMIDI najprej poskuša prepoznati pravilnost poslanega toka podatkov v skladu z Alg. 1, šele nato v lupini po vrsticah izpiše poslane dogodke (slika 7). Poleg tega, program ustvarja navidezna

vrata MIDI, ki so potrebna za kasnejšo sintetizacijo zvoka v programu Hydrogen. Vrata nadomeščajo fizični 5-pinski priključek MIDI, ki ga mikroračunalnik Raspberry Pi nima vgrajenega. Čeprav na tržišču obstajajo specializirani pretvorniki MIDI na USB, ki bi ga lahko uporabili v našem projektu, vendar bi za povezavo z mikrokrmilnikom potrebovali še kabel MIDI.

Za uspešno predvajanje zvoka na mikroračunalniku je potrebno konfigurirati Raspberry glasbeno (angl. audio) nadzorno ploščo QjackCtl. Prva konfiguracija je nekoliko zapletena, je pa princip enak za vse sintetizatorske programe.

Program Hydrogen je splošno namenjen za predvajanje zvokov bobnov, četudi omogoča spreminjanje barve zvoke, dodajanje efekta odmeva, ipd. Program je zato možno uporabiti tudi za predvajanje nekaterih drugih glasbil. Velja omeniti, da program Hydrogen ni edini program za sintetizacijo zvoka na operacijskem sistemu Raspbian, ampak obstaja še npr. Swami. Princip delovanja omenjenega temelji na t.i. zvočnih pisavah (angl. Soundfont), katere izberemo ob zagonu programa. Soundfont je format zapisa vzorcev glasbil (za vsako noto je posnet zvok glasbila), ki jih vzorčevalni sintetizator poveže s poslanimi notami MIDI. Mikroračunalniku smo za prijaznejšo uporabniško izkušnjo dodali tudi uporovni zaslon na dotik, ki omogoča upravljanje s priloženim pisalom (slika 7).



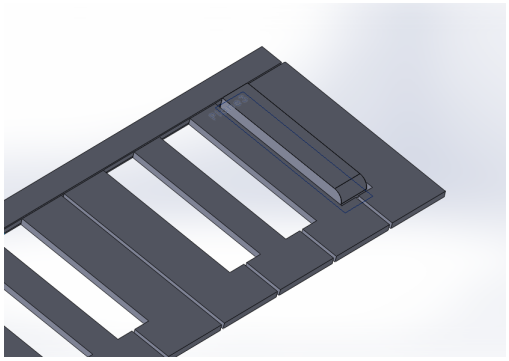
Slika 6: Zaznavanje poslanih MIDI not v programu ttyMIDI na mikroračunalniku Rasbperry Pi.

Slika 7: Mikroračunalnik Raspberry Pi 2 z zaslonom na dotik.

4. TESTIRANJE KLAVIATUR

Klaviature merijo 45 × 21 × 18 cm, tehtajo okrog 1300 g (brez baterije) in so napram velikim sintetizatorjem lahko prenosljive (slika 8). Težo bi z izbiro primernejšega materiala lahko še dodatno zmanjšali. Klaviature trenutno sicer še niso popolne, saj jim zaradi neujemanja zaenkrat še nismo namestili pokrovov tipk (slika 9). Za tipke smo predvideli tisk 3D.

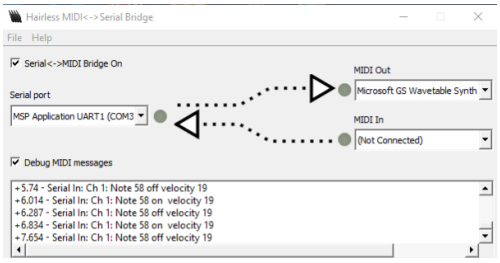
Slika 8: Končen, vendar ne končan izdelek.



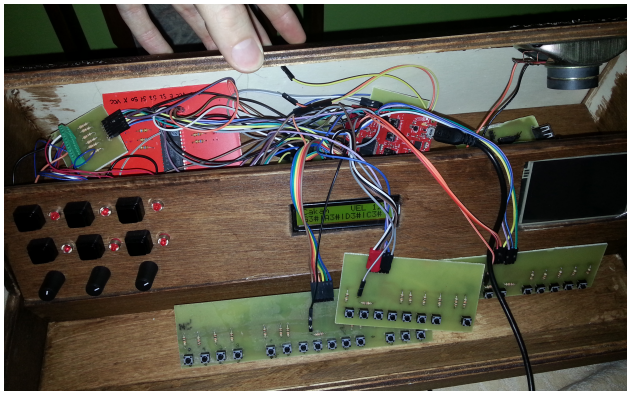
Slika 9: Načrtovani pokrovi tipk, ki pa še niso bili natisnjeni.

Kljub nepopolnemu izdelku smo klaviature lahko stestirali na operacijskem sistemu Raspbian in sistemu Windows (slika 10).

Ugotovili smo, da je s klaviaturami moč zaigrati kakršnokoli skladbo. Kakovost tipkovnice je solidna, delujejo vse tipke. Predvajan zvok je dovolj glasen, zato je ojačevalec primerno načrtovan, kakor tudi kakovost zvoka, ki je nekoliko odvisna tudi od oblike klaviatur. Tipki za transponiranje in zavijanje note zadovoljivo upravičujeta svojo vlogo, saj lahko sintetizator predvaja tudi vmesne note.



Slika 10: Zaznavanje poslanih not MIDI v programu Hairless MIDI Serial Bridge na osebnem računalniku.



Slika 11: Nepregledno število žic.

5. SKLEP

Ugotovili smo težavo s prekomerno zakasnitvijo med pritiskom tipke in dejanskim predvajanjem zvoka. Sumimo, da do tega prihaja zaradi slabe optimizacije programa mikrokrmilnika. Problema se popolnoma že v osnovi ne da odpraviti, lahko se ga pa minimizira. Z nadaljnjo optimizacijo lahko dosežemo zakasnitev pod 30 ms, kar je veliko manj kot trenutna, ki znaša 100-150 ms.

Prav tako bi bilo že v osnovi potrebno izbrati zaslon LCD, ki deluje na napajalni napetosti mikrokrmilnika (3.3 VDC), s čimer bi dodatno prihranili prostor in nekaj priključkov na mikrokrmilniku. Odzivni čas zaslona je v primerjavi s tistimi iz višjega cenovnega razreda visok in nam je v začetni fazi povzročal nemalo težav z odzivnostjo klaviatur. Problem smo odpravili z uvedbo prekinitev znotraj programa na mikrokrmilniku. Pojavljale so se tudi težave z zaslonom na dotik, saj je bilo težko najti primerne gonilnike ter zaslon usposobiti.

Da bi bile klaviature resnično prenosljive in uporabniku prijazne, bi bilo potrebno napisati grafični vmesnik, kateri bi se pojavil takoj ob zagonu. Ta bi moral interaktivno omogočati izbiro glasbila in njegovih parametrov. Najprimernejši programski jezik za uporabniški program bi po našem mnenju bila Java, v kateri je relativno enostavno programirati in je kompatibilna z mikroračunalnikom Raspberry Pi.

Za širšo paleto sintetiziranih zvokov predlagamo uporabo Soundfont-ov, katerih opus obsega zvoke klavirja, bobnov, kitare, trobente in nekaterih drugih. Prav tako želimo v nadaljevanju tudi za glasbeno spremljavo usposobiti povezljivost toka MIDI podatkov s svetlobnimi efekti osvetljave.

Kot končno rešitev na množico kablov in priključkov (slika 11) bi predlagali gradnjo univerzalne matične plošče, kamor bi pritrdili vse zahtevane module in tako optimirali izrabo prostora. Prav tako bi prihranili na končnih stroških, saj bi s tem odpadli stroški številnih priključkov.

Zahvala

Za pomoč pri izvedbi projekta, se iskreno zahvaljujem profesorju Alešu Hacetu s Fakultete za elektrotehniko, računalništvo in informatiko v Mariboru in asistentu Roku Pučku, ki sta mi pomagala pri programiranju mikrokrmilnika MSP430. Hvala tudi za priložnost gradnje klaviatur kot projektnega dela.

6. REFERENCE

[1] MIDI Association. `https://www.midi.org/`. [Dostopano: 2016-09-2].

[2] P. R. Cook and G. Scavone. The synthesis toolkit (stk). In *Proceedings of the International Computer Music Conference*, pages 164–166, 1999.

[3] R. P. Gilmore. Direct digital synthesizer driven phase lock loop frequency synthesizer, Oct. 23 1990. US Patent 4,965,533.

[4] P. D. Griffis. Switching arrangement for a stereophonic sound synthesizer, Oct. 23 1984. US Patent 4,479,235.

[5] J. Hass. Midi. `http://www.indiana.edu/~emusic/etext/MIDI/chapter3_MIDI4.shtml`, 2013. [Dostopano: 2016-08-31].

[6] Y. Kaneoka. Electronic sound synthesizer, Nov. 8 1988. US Patent 4,783,812.

[7] A. M. McDonald, D. C. Quinn, and D. C. Perry. Sound synthesizer in a vehicle, Dec. 6 1994. US Patent 5,371,802.

[8] M. Russ. *Sound synthesis and sampling*. Taylor & Francis, 2004.

[9] Wikipedia. MIDI. `https://en.wikipedia.org/wiki/MIDI`, 2016. [Dostopano: 2016-08-31].

[10] Wikipedia. Sintetizator. `https://sl.wikipedia.org/wiki/Sintetizator`, 2016. [Dostopano: 2016-08-22].

Greedy Heuristics for the Generalized Independent Cascade Model

László Tóth
University of Szeged
13 Dugonics Squere
Szeged, Hungary
tlaszlo@inf.u-szeged.hu

ABSTRACT

In this paper we formulate optimization problems for the Generalzed Independent Cascade Model, which is introduced by Bota et al. The purpose of these tasks is to define the size k which cause the greatest infection in a network. We show that a greedy heuristic ensure the theoretically guaranteed precision for the problems. Finally we investigate the our methods with practical tests.

Keywords

Influence Maximization, Generalized Independent Cascade Model, Greedy Heuristics

Supervisor

Dr. Miklós Krész
Department of Applied Informatics,
University of Szeged, Hungary
Contact: kresz@jgypk.szte.hu

1. INTRODUCTION

We usually think about medical science if we hear about the infection. It also appeared in the sociology in the early years of scientific modeling, then some sociologist start to use Linear Threshold Model (in [5]) and try to model different social effect. Therefore, economist discovered that the modified version of threshold model (Independent Cascade Model, in [4]) is suitable to help research of business process. This model is the most important break-through the theory of computation point of view. The infection spreads in social networks the following way: we choose an initial infected vertex set, and only those vertices spread the infection after this that became infected in the previous iteration. Every edge start from an infected vertex has a single chance for infecting using its own infection probability. Kempe et al. [6] defined the influence-maximization problem, where we look the highest. This problem is NP-complete, but Kempe et al. proved in [6] that applying a greedy algorithm gives a guaranteed precision that results in good quality solutions in practice. In [2] Bóta et al. extended the model and introduced the Generalized Independent Cascade Model. The initial state of this model does not consist of *infected* and *uninfected* vertices, but introduces an *a priori* probability of infection for each vertex. Each vertex receives an *a posteriori* infection value by the end of the infection process. Because the infection process above is #P-complete, [2] introduces a number of approximation algorithms. In turn, the influence-maximization problem has not been defined for this more general case.

In this paper we define two infection-maximization problems connected to the generalized model above. They are the positive and the negative problems. At the *positive task* every elements of a social network have a probability values, so-called 'input' values. These values are considered if a element is part of the inintial infection set, the other values is zero during the positive infection process. At the end of this the amount of the probability values of infected vertices mean the a posteriori value of the current process. If we model a campain of a product we can take an affinity analysis. The optimization problem for this task is to find the expected value of the maximal number of the infected elements. At the introduced *negative task* we assume that a firm would like to increase the clients of 'churn incliation' with help of a offered products. In this case every single elements (clients) have another values, so-called 'uplift' values as the measure of the churn willingness. The method calculates with uplift values of the elements if the they are part of the initial set, otherwise with the The related optimization problem is to find the minimal churn incliation.

We also present solution methods based on greedy heuristics, and compare their results with each other. We prove that a guaranteed approximation precision can be achieved for these greedy algorithms. To guarantee efficiency from a practical point of view, we decrease the search space for the greedy methods using different approaches (e.g. selection based on vertex degree). In our test cases we compared this heuristics with help of generated graphs.

2. INFECTION MODELS

The aim of this chapter is to introduce the main definition of our topic. The basic definitions and concepts related to graph theory are based on [10]. A simple graph is a $G = (V, E)$ structure, where V is a finite vertex or vertex set, E is the edge set. There are probabilities on the edges and the infection is spread helping these values according to a given

rule. There are networks that are built of simple graphs which contain directed edges with values. These networks are used in infection models, which have several types. All of them have similar input structure which is built by a finite iterative process. During this process there are vertex sets that contain three type of vertices: infected, infectious, not infected yet (susceptible). We can consider two types of process: progressive and non-progressive. The base of the progressive infection models is that their vertices cannot be *recovered* if they had been infected. In turn, during the non-progressive process there is recovery. The models of paper [6] use progressive process. The infection models can be distinguished by the type of edge-values. The models can be stochastic infection models if edge-values are probabilities (fractions between 0 and 1). These probabilities give the so-called 'effect of network'. After building a model we get an infected vertex set A and the value of infection as the expected value of infected vertices is $\sigma(A)$.

2.1 Independent Cascade Model

In this section we show a specialized infection model, the Independent Cascade Model (IC). It is introduced in paper [4]. Starting with an initially active set of vertices $A_0 \subset V(G)$. Let $A_i \subset V(G)$ denote the set of vertices newly activated in iteration i. In iteration $i+1$, every vertex $u \in A_i$ has one chance to activate each of its inactive neighbors $v \in V \setminus \cup_{0 \leq j \leq i} S_j$ according to $w_{u,v}$. These are usually probability values. If the attempt is successful, then v becomes active in iteration $i+1$. In this case the vertices are infected independently. The vertices can infect a susceptible one at once in the current iteration. [6]

The open question of the field is to compute the expected number of infected vertex is #P-complete. There are relevant researches for this question in papers [3] and [7]. Kempe and his colleagues gave approximate values with arbitrary accuracy for the number of infected elements with simulation for this problem. Unfortunately, this simulation is very costly. They defined the influence-maximization problem for this model as well. They looked for the vertex set for which the expected value of the infected vertices is the highest. The optimal solution for influence maximization can be efficiently approximated within a factor of $(1 - 1/e - \epsilon)$, where e is the base of the natural algorithm and ϵ is any positive real number. This is a performance guarantee slightly better than 63%. They proved that applying a greedy algorithm gives the guaranteed precision that results in good quality solutions in practice.

2.2 Generalized Independent Cascade Model

Bóta et al. extended the IC and introduced the Generalized (Independent) Cascade Model (GIC)[1] [2]. The initial state of this model does not consist of infected and uninfected vertices, but introduces an a priori probability (input infection weight w) of infection for each vertex. At the end of the infection process, each vertex receives an a posterior infection value (output infection weight (w^*)).

Formally, there is a given function $w_v : V \to [0, 1]$ on the vertices, which is the a priori measure of elements of network G.

[1]Bóta et al. used the Generalized Cascade technical term, but this name is already reserved by Kempe et al. [6]

At the beginning the vertices become active independently of each other with their assigned probabilities. Thereafter, the infection process of the IC model applies with a randomly selected active set. In this process, each vertex gets a value by function $w^* : V \to [0, 1]$ on the vertices, which are the a posteriori influence. The influence measure $\sigma(\cdot)$ which is the sum of output infection value can be given:

$$\sigma(w) = \sum_{v \in V} w^*(v)$$

There are several simulation that calculating this measure. We consider the Complete Simulation (CS) according to paper [2] and we prove that theory of Kempe et al. is right for the examined GIC. A required number of simulations can be defined to give certain measure of approximation (ϵ) to $\sigma(A)$ with a given probability (δ). The CS is a method that gets a network G and a sample size k. The result of algorithm is the relative frequency of infection. Since this method is based on frequency counting, its precision and time complexity depends greatly on the size k.

Proposition 1. *If the GIC starting with A is simulated independently at least*

$$\ln(2/\delta)/(2w_{av}^2\epsilon^2)$$

times, then the average number of activated vertices over these simulation is a $(1\pm\epsilon)$-approximation to $\sigma(A)$, with probability at least $(1-\delta)$, if $\delta, \epsilon \in (0, 1)$, and w_{av} is the average of the input weights.

We remark that the Proposition 1. is not only true for GIC, but for the infection process in general.

2.3 Infection Heuristics

Although after the model generalization, Bóta et al. developed simulations and heuristics which efficiency and accuracy were not acceptable. As a consequence, other heuristics are worth to research. We looked for a unique heuristic which can be accelerate existing methods. In our study we used the *analytic formula* by Srivastrava et al. [9], which is usually applied in binary infection models. This formula has been used for IC model before. We used it at first for GIC.

Let $A_{u,t}$ denote a probability that vertex u is infected at time t. Furthermore let $B_{u,t}$ denote another probability that vertex u is infected until time t. According to above $A_{u,0} = w(u)$, so at time 0 the apriori infection will be given. In addition to $B_{u,t} = \sum_{j=0}^{t} A_{u,j}$, therefore $A_{u,t} = B_{u,t} - B_{u,t-1}$.

Let $N_i(u)$ denote the endpoints set of incoming edges of vertex u, then we can calculate $B_{u,t}$ simply:

$$B_{u,t} := 1 - (1 - B_{u,t}) \cdot \prod_{v \in N_i(u)} (1 - p(v, u) \cdot A_{v,t-1})$$

We can approximate the a posteriori infection with $B_{u,t}$ because the infection process is progressive. The accuracy can be enhanced by increasing t. The basic idea of this heuristic is to consider independently the effect of edge probabilities.

3. INFLUENCE-MAXIMIZATION

At the original model a function $\sigma(\cdot)$ is given at the IC model and we are looking for the set, for which the function is maximal. For the general case a function w of probabilities is given a priori and it is not obvious how we can maximize it. We defined two problems to analyze the function. They are positive and negative influence maximization problem connection with spreading of positive and negative effect.

3.1 Influence-maximization in the GIC

Because the influence-maximization problem has not been defined for more general cases, we defined two infection-maximization problems. Both problems have modified input probabilities. For the positive case we defined probability values, which are greater than zero on vertices of initial vertex set. It is zero on the other vertices. An example for the positive case is product affinity, which is measured by the reaction of the clients for a product at the service sector. An example for the negative infection maximization model is churn prediction. We measure the relationship between the clients and the service provider. We would like to maximize the minimal churn. In this case there are the so called 'uplift' probability values, which are given at initial vertices of vertex set. These values are measured for the clients at the service providers. The vertices that are not in the initial vertex set can be measured by the original probability values. These values give the size of the client churn at companies. The formally definitions of tasks are the next:

Definition 1. *(Positive influence maximization model) Having a network G, where a set $A \subset V$. Let w function of weights, and F a GIC model. Furthermore let $w^A_{p,v}$ the modified input probability, where*

- $\forall v \in A,\ w^A_{p,v} = w_v$
- $\forall v \notin A,\ w^A_{p,v} = 0.$

Optimization task: Search set A with k elements, where $\sigma^p(A) = \sigma(w^A_p)$ is maximal.

Thus elements of A get values according to w, another elements $(V \setminus A)$ get zero.

Definition 2. *(Negative influence maximization model) An IC model which of input infection have a $w^A_{n,v}$ modified input probability, where*

- $\forall v \in A, w^A_{n,v} = w^{up}_v$, *uplift probability*
- $\forall v \notin A, w^A_{n,v} = w_v$.

Optimization task: Search set A with k elements, where $\sigma_n(A) = \sigma(w) - \sigma(w^A_n)$ is maximal.

Thus elements of A get values according to an uplift function, another elements $(V \setminus A)$ get values according to w.

At the campain of the products k clients are targeted for both cases. At the current infection process step all susceptible elements will be activated according to their edge and vertex probabilities.

3.2 Reducing the solution space

To accelarate our methods we have to reduce the solution space. The elements of the initial set of vertices are from the set of which the elements have positive input probability at the positive case. At the negative case there are only those elements in the initial set which are smaller than the input value. This reduction is useful because the excluded elements are not important at calculating the estimated probability of the network. In this case the influence-maximization problem is to find reduced set V an initial set A with given k elements, where these elements have maximal infection. Formally let $v \in V$:

- *Positive case*: if $w_v > 0$, then $v \cup V^*$
- *Negative case*: if $w^{up}_v < w_v$, then $v \cup V^*$

We prove that we can get optimal value after the reduction.

Lemma 1. *Let V^* denote elements of V, for which the input infection w is positive at network G and case of GIC model. Thus, infection function $\sigma^G_p(w)$ gets its optimal value on the set $A \subseteq V^*$.*

Lemma 2. *Let V^* denote elements of V, for which value w^{up} less than input infection w at network G and case of GIC model. Thus, infection function $\sigma^G_n(w_n)$ gets its optimal value on the set $A \subseteq V^*$.*

4. SOLUTION METHODS

Since Kempe et al. in [7] realized that approximation of the optimum of the complex task can be given with arbitrary precision with the help of a greedy algorithm, it seems natural to use it at the general case. We illustrate what type of acceleration can be used for the calculation.

4.1 Greedy framework

In this section we introduce a theorem which can be used for the specified two tasks and we show the proof. To present our results, we need to introduce some concepts based on [7]. We said that function f is *submodular* if it satisfies a natural 'diminishing returns' property: the marginal gain from adding the same element to a superset of S. Formally, a submodular function satisfies

$$f(S \cup \{v\}) - f(S) \geq f(T \cup \{v\}) - f(T),$$

for all elements v and all pairs of sets $S \subseteq T$. And a function f *monotone* if adding an element to a set cannot cause f to decrease:

$$f(S \cup \{v\}) \geq f(S).$$

The greedy method starts from an empty set. It repeatedly adds the vertex x to set A maximizing the *marginal gain* $g(A \cup \{x\}) - g(A)$ if having a submodular function g. [3]

In [8] Nemhauser et al. studied the monotone, submodular functions:

Theorem 1. *Let $\sigma(\cdot)$ denote a non-negative, monotone, submodular function. Then the greedy algorithm (for k iterations) adds an element with the largest marginal increase in $\sigma(\cdot)$ produces a k-element set A such that*

$$\sigma(A) \geq (1 - 1/e) \cdot max_{|B|=k}\sigma(B)$$

Algorithm 1 Greedy Method

1: Start with $A = \emptyset$.
2: **while** $|A| \leq k$ **do**
3: For each vertex x, use repeated sampling to approximate $g(A \cup \{x\})$
4: Add the vertex with largest estimate for $g(A \cup \{x\})$ to A.
5: **end while**
6: Output the set A of vertices.

The function σ satisfies the conditions above if we consider the IC model. Therefore Nemhauser et al. gave a suitable approximation. Our main result is that we showed that it is suitable for the GIC. So it is also true for the expected value $\sigma(w_{p,v})$ and $\sigma(w_{n,v})$ in the case of the positive and negative influence maximization problems.

Let σ^* denote σ_p and σ_n to facilitate the description of methods. As seen before σ^* are monotone, submodular and non-negative, we can use the greedy strategy for the general model (Algorithm 2)

Algorithm 2 Greedy Method for GIC

Initialization:

$A := \emptyset$

Iteration:

1: **while** $|A| \leq k$
2: $A = A \cup \arg\max_{x \in V^* \setminus A} \sigma^*(A \cup \{x\})$

Output: A

If σ^* is non-negative, monotone and submodular then the greedy strategy above ensures the guaranteed accuracy according to [8]. We proved the following theorems:

Theorem 2. *Let G be a network and let w denote input influence function. Pick a set V^* contains elements for which w gives positive value. Then $\sigma_p^G(w)$ is non-negative, monoton and submodular on the set V^*.*

Theorem 3. *Let G be a network and let $w_n = (w, w^{up})$ denote input influence function. Let pick a set V^* contains elements for which $w^{up} < w(v)$ gives positive value. Then $\sigma_n^G(w)$ is non-negative, monoton and submodular on the set V^*.*

The following proprosition is consequence of the theorems above and [8].

Corollary 1. *The Greedy Heuristic gives at least $(1-1/e-\epsilon)$-aprroximation for the positive and negative infection maximization problems if the model is GID.*

4.2 Reduction methods

We decrease the search space for the greedy methods using different approaches based on practical considerations. Based on the above, if the algorithm does not look for the solution on the whole search space, we lose the guaranteed precision. Our expectation is that using the appropriate strategy the result will be satisfying. The speed of the algorithm increase because of the greedy property after the reduction. Formally, the methods look like as following:

In the given iteration, if X is the current solution, we consider the $f^*(v)$ value for all elemets of $X \in V^*$. From these we consider the greatest r elements, set $V^{\tilde{}}$. We use the greedy strategy only on set $V^{\tilde{}}$ and we consider the greatest marginal gain. We can choose from two different reduction method:

- *Degree:* Function f^* gets values as following:
  Positive maximalization problem: For each vertex, adding the probabilities of outgoing edges. The sum is multiplied by the input probability of the given vertex. The method selects vertices which are the greatest according to these values.
  Negative maximalization problem: It is similar to positive one, but the product of values are subtracted from uplift value of the given vertex.
  The process sets up a sequence for both cases and considers the greatest r elements.
- *Modified Degree:* Function f^* gets values as following:
  The process is similar to *Degree*. The difference is that the currently investegated vertex will be not in the next phase. Because of it, this is a dynamic calculation.

After the above is executed the greedy method (Algorithm 2) with $V^{\tilde{}}$ instead of V^*.

5. TEST RESULTS

In this section we present the program, which was implemented for test. After that we show the details of the test cases. At the end of the chapter we present the result for analysis of GIC.

5.1 Test program

We created a scalable framework to analyze our theoretical solutions. As part of this framework we have a simulator, which generates edge- and vertex-probabilities, what we usually need to measure in real networks. We also use σ-approximations, a simulation and a heuristic generator. The uplift value were set up according to the input value. We can generate the follows:

- edge probability
- a priori infection probabilities of vertices
- uplift probabilities of vertices according to a priori infection probabilities

We can set the following:

- type of task (positive, negative)
- heuristic mode (degree, modified degree)
- iteration, evaluation method (complete simulation, analytic formula heuristic [with parameter t])
- a priori infected set size
- number of infection and evaluation iteration

5.2 Test Cases

In many test cases we investigated the main behavior of algorithms. We worked with about hundred graphs, which came from a word-association case study. In this graphs the vertices are words, and the directed connections are association one of words to another. We also used graph generator (Forest Fire Model) to obtain graphs. [1] The vertex number in these graphs is from hundred to thousand. The rate of those vertices where the uplift probability is greater than input probability is 25%. For other vertices the rate is 50%. In the first case, the increase of probabilities can be up to 20%, at the other case, the decrease of probabilities can be up to 60%. The other parameters can be seen on the next table.

Table 1: Parameterization

variable	value
$p(u,v)$	$0,1$
$\|V\|$	$100,..,1000$
w_v	$[0,1] with uniform distribution$
k	$\|V\| \cdot 0,05$

5.3 Results

After the infection model was built we used simulations on our networks to evaluate them. We used it the whole solution space of the given network. Then, we applied our *analytic formula* (AF) heuristic with parameter $t \leq 5$ for the same networks, but we reduced the solution space helping with the introduced reduction method. We investigate the average of results. If we use this heuristic, we need to be careful about the parameters, because the accuracy will be at the expense of efficiency. We need to find the balance.

Table 2: Results of positive problem testing

method	rate
whole solution space with Simulation	100%
degree reduction with AF	**95,51%**
modified degree reduction with AF	96,00%

Table 3: Results of negative problem testing

method	rate
whole solution space with Simulation	100%
degree reduction with AF	**89,52%**
modified degree reduction with AF	99,14%

When we used the reduction, the run time was five times faster on average than the case of whole solution space. The tables show results obtained from the mean value of different networks. At the negative problem testing the values are smaller than at the positive case because of the theoretical background. At this point the difference is greater then at positive, but the behavior of tasks are similar. The network effect works in a different way from network which negative-parameterized than a positive one and it can be detected in.

6. CONCLUTIONS AND FUTURE WORKS

During our work we defined several problems for the influence-maximization in generalized models. We worked on suitable greedy framework, which gave guarantee precision for our problems. We reduced the search space with different method to reach the required accuracy, and to accelerate our process. In the end we compared the results. In the future we would like to build an efficient simulate annealing framework, then we will test our method on real graphs with real churn value from databases of companies.

Although during the work a lot of theoretical compositions and methods have been introduced, but they have a relevant role to play in everyday life. Nowadays the information spreading and viral marketing are becoming more and more common terms. Economic agents realized that finding the relevant customers in connection with the given problem means financial potential. Solving such questions precisely and quickly can give unexpected positive results.

7. ACKNOWLEDGEMENT

I would like to express my gratitude to Dr. Miklós Krész, my supervisor for his enthusiastic encouragement and useful remarks of this research work.

This research was supported by National Research, Development and Innovation Office - NKFIH Fund No. SNN-117879.

8. REFERENCES

[1] P. Bak, K. Chen, and C. Tang. A forest-fire model and some thoughts on turbulence. *Physics letters A*, 147(5):297–300, 1990.

[2] A. Bóta, M. Krész, and A. Pluhár. Approximations of the generalized cascade model. *Acta Cybernetica*, 21(1):37–51, 2013.

[3] W. Chen, C. Wang, and Y. Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *KDD*, 2010.

[4] P. Domingos and M. Richardson. Mining the network value of customers. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 57–66, New York, NY, USA, 2001. ACM.

[5] M. Granovetter. Threshold models of collective behavior. *American Journal of Sociology*, 83(6):1420–1443, 1978.

[6] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 137–146, New York, NY, USA, 2003. ACM.

[7] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. *Theory of Computing*, 11(4):105–147, 2015.

[8] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical Programming*, 14(1):265–294, 1978.

[9] A. Srivastava, C. Chelmis, and V. K. Prasanna. The unified model of social influence and its application in influence maximization. *Social Network Analysis and Mining*, 5(1):1–15, 2015.

[10] D. B. West. *Introduction to Graph Theory*. Prentice Hall, 2005.

Hybrid Cuckoo Search for constraint engineering design optimization problems

Uroš Mlakar
University of Maribor
Smetanova 17
Maribor, Slovenia
uros.mlakar@um.si

ABSTRACT

This paper investigates, how the hybrid self-adaptive Cuckoo Search algorithm (HSA-CS) behaves, when confronted with constraint engineering design optimization problems. These problems are commonly found in literature, namely: welded beam, pressure vessel design, speed reducer, and spring design. The obtained results are compared to those found in literature, where the HSA-CS achieved better or comparable results. Based on results, we can conclude, that the HSA-CS is suitable for use in real-life engineering applications.

Keywords

design optimization, hybridization, cuckoo search, self-adaptation

1. INTRODUCTION

There is a increasing rate in the research community for developing new constrained optimization algorithms. Therefore suitable problems must be used, to show the effectiveness, efficiency and convergence of these new algorithms. Such problems are usually mathematical problems like the CEC competition problems, but also engineering design optimization problems are adopted in the specialized literature. Many researchers have studied these problems, by applying a wide range of different optimization methods such as Quadratic Programming [5], Simulated Annealing [12], Genetic Algorithms [9], and Swarm Intelligence [2, 3, 6, 7, 1]. The listed algorithms are among the most used in the literature. The design optimization problems usually have a non-linear objective function and constraints, while the design variables are often a combination of discrete and continuous. The hardest part of finding the optimal solution for these problems is directly related to the constraints, which are imposed on the problem.

Since SI based algorithms are getting a lot of attention in the past couple of years, our aim was to test the behaviour of a novel hybrid self-adaptive Cuckoo Search [8] (HSA-CS) on the design optimization problems. The rest of the paper is organized as follows. In Section 2 some of related work, mostly that of SI-based algorithms is presented. Section 3 is dedicated to Cuckoo Search (CS) and the used self-adaptive hybrid Cuckoo Search algorithm, where an emphasis is on describing the main differences from the original CS. Section 4 deals with describing the design optimization problems, and then in Section 5 the obtained results are presented. In Section 6 the results obtained by the HSA-CS is compared to those in the literature, and the paper is concluded in Section 7.

2. RELATED WORK

Since the HSA-CS belongs to the SI-based algorithms, it would only be reasonable to review the literature from this point of view. Akay and Karaboga [1] presented an artificial bee colony (ABC) algorithm, with a very simple constraint handling method. This method is biased to choose feasible solutions rather than those, which are infeasible. Gandomi et al. [7] use a bat algorithm for solving constraint optimization problems. Their results indicate that their method obtained better results, compared to those in literature. Another ABC algorithm was proposed by Brajevic and Tuba [3]. The upgraded ABC algorithm enhances fine-tuning characteristics of the modification rate parameter and employs modified scout bee phase of the ABC algorithm. Baykasoglu and Ozsoydan [2] presented an adaptive firefly, enhanced with chaos mechanisms. The adaptivity is focused on on the search mechanism and adaptive parameter settings. They report that some best results found in literature, were improved with their method. Bulatović [4] applied the improved cuckoo search (ICS) for solving constrained engineering problems, which produces better results than the original cuckoo search (CS). Their improvements lie in the dynamic changing of the parameters of probability and step size. Yang et al. [11] utilized a multi-objective CS (MOCS) for the beam design problem and disc brake problems. They conclude that the proposed MOCS is efficient on problems with complex constraints.

3. CUCKOO SEARCH

Cuckoo search is a stochastic population-based optimization algorithm proposed by Yang and Deb in 2009 [10]. It belongs in the SI-based algorithm family, and it is inspired by the natural behaviour of some cuckoo species in nature. To trap the behavior of cuckoos in nature and adapt it to be suitable for using as a computer program the authors [10] idealized three rules:

- Each cuckoo lays one egg, and dumps it in a randomly chosen nest,
- Nests with high-quality egg, will be carried over to the next generations,
- Any egg laid by a cuckoo, may be discovered by the host bird with a probability of $p_a \in (0, 1)$. When an egg is discovered, the host bird may get rid of it or simply abandon the nest and build a new one.

Each solution in the population of the cuckoo search algorithm corresponding to a cuckoo nest, represents the position of the egg in the search space. This position can be mathematically defined:

$$\mathbf{x}_i = \{x_{i,j}\},\ for\ i = 1, \ldots, Np\ and\ j = 1, \ldots, D, \quad (1)$$

where Np represents the population size, and D the dimension of the problem to be solved.

Generating new solutions in the CS is done by executing a random walk, with the use of the Levy flight distribution:

$$\mathbf{x}_i = \mathbf{x}_i + \alpha L(s, \lambda). \quad (2)$$

The term $L(s, \lambda)$ determines the characteristic scale, and $\alpha > 0$ denotes the scaling factor of the step size s.

3.1 Hybrid self-adaptive Cuckoo Search

According to [8] the CS was modified by adding the following mechanisms: balancing of the exploration strategies within the CS, self-adaptation of the parameters, and population reduction. The used exploration employed by the HSA-CS are:

- random long distance exploration,
- stochastic short-distance exploration, and
- stochastic moderate-distance exploration.

The listed strategies have an impact on how the trial solution will be generated. The random long distance exploration is implemented as the abandon operator. The second strategy improves the current solution by using a local random walk, with the help of Levy flights (Eq. 2). The last strategy is borrowed from the DE algorithm. Additionally the last strategy adds a crossover operation to the CS algorithm. These execution of these strategies is controlled by a single parameter.

As was stated all parameters are fully self-adaptive, except the starting population size, which must be experimentally defined. Additionally the strategy balancing probability, the abandon rate, and the elitist parameter (controls whether a random of best solution is taken as the basis trial vector calculation) are determined by the user. Lastly the population reduction is implemented by using a simple linear reduction.

It was proven by the authors of the HSA-CS, that the biggest impact on the results has the inclusion of multiple strategies, than followed by self-adaptation. Population reduction did not have a big impact on the results. For more information about HSA-CS readers are referred to [8].

4. CONSTRAINED DESIGN OPTIMIZATION PROBLEMS

The following design optimization problems have been used in this study: welding beam, pressure vessel design, spring design, and speed reducer design. The used problems are thoroughly presented and formally defined in the remainder of this section.

4.1 Welding beam

The goal of this problem is to design a welded beam subject to minimum cost, subject to some constraints. The problem consists of four design variables, with the objective is to find the minimum fabrication cost, with constraints of shear stress τ, bending stress σ, buckling load P_c, and end deflection on the beam δ. The mathematical model can be formulated as follows:

$$f(\mathbf{x}) = 1.10471 x_1^2 x_2 + 0.04811 * x_3 x_4 (14 + x_2), \quad (3)$$

subject to:

$$g_0 : \tau - 13600 \leq 0,\ g_1 : \sigma - 30000 \leq 0,\ g_2 : x_1 - x_4 \leq 0,$$
$$g_3 : 0.10471 x_1^2 + (0.04811 x_3 x_4 (14 + x_2)) - 5 \leq 0,\ g_4 : 0.125 - x_1 \leq 0,$$
$$g_5 : \delta - 0.25 \leq 0,\ g_6 : 6000 - P_c \leq 0, \quad (4)$$

where

$$\tau = \sqrt{\tau_1^2 + 2\tau_1\tau_2\frac{x_2}{2R} + \tau_2^2},\ \tau_1 = \frac{6000}{\sqrt{2}x_1x_2},\ \tau_2 = \frac{MR}{J},$$
$$M = 6000(14 + \frac{x_2}{2}),\ R = \sqrt{\frac{x_2^2}{4} + (\frac{x_1 + x_3}{2})^2},$$
$$J = 2(\sqrt{2}x_1x_2(\frac{x_2^2}{12}) + (\frac{x_1 + x_3}{2})^2),\ \sigma = \frac{504 * 10e^3}{x_4x_3^2},$$
$$\delta = \frac{65856 * 10e^3}{3 * 10e^6 x_4 x_3^3},$$
$$P_c = \frac{(12.039 * 10e^6\sqrt{(x_3^2x_4^6)/36})}{196}(1 - \frac{x_3\sqrt{\frac{3*10e^6}{48*10e^6}}}{28.0}) \quad (5)$$

The design variables are bounded as: $0.1 \leq x_2, x_3 \leq 10$, and $0.1 \leq x_1, x_4 \leq 2$.

4.2 Pressure vessel design

The idea of this problem is designing a compressed air storage design, with a working pressure of 1000 psi and and minimum volume of 750 ft^3. The problem is described using four variables, which represent shell thickness, spherical head thickness, radius and length of the shell. The objective of the problem is minimizing the manfuacturing cost of the pressure vessel, and can be formulated as:

$$f(\mathbf{x}) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3, \quad (6)$$

subject to

$$g_0 : -x_1 + 0.0193x_3 \leq 0,\ g_1 : -x_2 + 0.00954x_3 \leq 0,$$
$$g_2 : -\pi x_3^2 x_4 - \frac{4}{3}\pi x_3^3 + 1296000 \leq 0,\ g_3 : x_4 - 240 \leq 0. \quad (7)$$

The bounds of the design variables are: $0.0625 \leq x_1, x_2 \leq 99 * 0.0625$, and $10 \leq x_3, x_4 \leq 200$.

4.3 Spring design

The spring design optimization problem deals with an optimal design of a tension spring. The problem consists of three variables, which are the number of spring coils, winding diameter, and wire diameter. The objective is to minimize the weight of the spring, subject to minimum deflection, surge frequency, shear stress, and limits on the outside diameter. Mathematically it can be formulated as:

$$f(\mathbf{x}) = (x_3 + 2) * x_1^2 * x_2, \tag{8}$$

subject to the following constraints:

$$g_0 : 1 - \frac{x_2^3 x_3}{71785 x_1^4} \leq 0,\ g_1 : \frac{4x_2^2 - x_2 x_3}{12566(x_2 x_3^3 - x_3^4)} + \frac{1}{5108 x_3^2} - 1 \leq 0,$$
$$g_2 : 1 - \frac{140.45 x_1}{x_2^2 x_3} \leq 0, g_3 : (x_1 + x_2) - 1.5 \leq 0 \tag{9}$$

The search space of design variables are limited as: $0.05 \leq x_1 \leq 2$, $0.25 \leq x_2 \leq 1.3$, and $2 \leq x_3 \leq 15$.

4.4 Speed reducer

This problem deals with a minimum weight design of a speed reducer, subject to bending stress of the gear teeth, surface stress, stresses in the shafts, and transverse deflections of the shafts. This problem is formulated with seven design variables, using the following mathematical definition:

$$f(\mathbf{x}) = 0.7854 x_1 x_2^2 (3.3333 x_3^2 + 14.9334 x_3 - 43.0934) -$$
$$1.508 x_1 (x_6^2 + x_7^2) + 7.477 (x_6^3 + x_7^3) + 0.7854 (x_4 x_6^2 + x_5 x_7^2), \tag{10}$$

subject to:

$$g_0 : \frac{27.0}{x_1 x_2^2 x_3} - 1.0 \leq 0,\ g_1 : \frac{397.5}{x_1 x_2^2 x_3^2} - 1.0 \leq 0,$$
$$g_2 : \frac{1.93 x_4^3}{x_2 x_3 x_6^4} - 1.0 \leq 0,\ g_3 : \frac{1.93 x_5^3}{x_2 x_3 x_7^4} - 1.0 \leq 0,$$
$$g_4 : \frac{\sqrt{(\frac{745 x_4}{x_2 x_3})^2 + 16.9 * 10e^6)}}{(110 x_6^3)} - 1.0 \leq 0,$$
$$g_5 : \frac{\sqrt{(\frac{745 x_5}{x_2 x_3})^2 + 157.5 * 10e^6)}}{(85 x_7^3)} - 1.0 \leq 0,$$
$$g_6 : \frac{x_2 x_3}{40} - 1 \leq 0,\ g_7 : \frac{5 x_2}{x_1} - 1 \leq 0,\ g_8 : \frac{x_1}{12 x_2} - 1 \leq 0,$$
$$g_9 : \frac{1.5 x_6 + 1.9}{x_4} - 1 \leq 0,\ g_{10} : \frac{1.1 x_7 + 1.9}{x_5} - 1.0 \leq 0. \tag{11}$$

The search of the design variables is defined as:
$(2.6, 0.7, 17.0, 7.3, 7.8, 2.9, 5.0)^T \leq \mathbf{x} \leq (3.6, 0.8, 28.0, 8.3, 8.3, 3.9, 5.5)^T$

5. RESULTS

The HCS-SA was applied to solve the design optimization problems, which were described in the previous section. To provide for a fair comparison with the literature, the number of function evaluations was set to 50000, as advised in [2], where the authors determined, that such a number is an average value for function evaluations found in literature. The parameters of HSA-CS were set according to the authors in [8], while the population size was varied as: $Np = 30$ for welded beam, spring design, and speed reducer, while for pressure vessel $Np = 50$. Each experiment was replicated 50 times, thus the results reported here are the average of those runs.

Table 1 holds the results of the experiments. For each problem the minimum (min), maximum (max), mean, median (md), and standard deviation (std) values are reported.

The results in Table 1 indicate that the HSA-CS was able to find the same solution for the welded beam and speed reducer problems in all 50 runs of the algorithm. On the contrary, the HSA-CS had trouble in converging towards a single solution.

6. DISCUSSION

In this section we analyze the results from our experiments and compare them to those found in the literature. For this purpose a Table 2 is provided, where results from literature are gathered. It can be seen, that the HSA-CS achieved competitive results if not better results on all test optimization problems. For the welded beam problem our method and the method in [2] converged to a single solution, whereas other methods were not as successful. It is also hard to say, which method performed the best, since the findings in other papers are reported only to 6 digits. On the pressure vessel problem, the HSA-CS achieved similar results as for the welded beam problem. Based on the mean value the only competitive method was again the one proposed in [2]. HSA-CS acheived the best results for the speed reducer. Again, like for the welded beam, the results were unanimous, converging to a single solution, which was also the smallest based on mean value. Good results were also obtained on the spring design problem, were the HSA-CS had the smallest std value over the 50 runs, while obtaining good results based on the mean value. We can conclude the HSA-CS would be suitable for use in real-life constraint optimization problems.

7. CONCLUSION

This paper investigated the recently proposed HSA-CS algorithm, on four well known engineering design optimization problems with constraints. The problems at hand were: welded beam, pressure vessel design, speed reducer design, and spring design. The obtained results were compared to the some state-of-the-art methods, were the HSA-CS performed very well, thus we can conclude it would be suitable for use in real-life engineering applications.

8. REFERENCES

[1] Bahriye Akay and Dervis Karaboga. Artificial bee colony algorithm for large-scale problems and engineering design optimization. *Journal of Intelligent Manufacturing*, 23(4):1001–1014, 2012.

[2] Adil Baykasoğlu and Fehmi Burcin Ozsoydan. Adaptive firefly algorithm with chaos for mechanical design optimization problems. *Applied Soft Computing*, 36:152–164, 2015.

Table 1: Results obtained by the HSA-CS on the four design optimization problems.

Problem	min	max	mean	md	std
Welded beam	1.724852309	1.724852309	1.724852309	1.724852309	0
Pressure vessel	6059.714	6410.087	6095.32	6059.714	88.270
Speed reducer	2996.218	2996.218	2996.218	2996.218	0
Spring design	0.01266523	0.0129087	0.01269661	0.01267089	5.402618 *10e^{-5}

Table 2: Comparison with state-of-the-art.

Welded beam				
Method	min	max	mean	std
Gandomi et al [7]	1.7312	2.3455793	1.8786560	0.2677989
Akay et al.[1]	1.724852	–	1.741913	0.031
Brajevic et al. [3]	1.724852	–	1.724853	0.0000017
Baykasoglu et al. [2]	1.724852	1.724852	1.724852	0
This study	1.724852309	1.724852309	1.724852309	0
Pressure vessel				
Method	min	max	mean	std
Gandomi et al [6]	6059.714	6495.347	6447.736	502.693
Akay et al.[1]	6059.714736	–	6245.308144	205.00
Brajevic et al. [3]	6059.714335	–	6192.116211	204
Baykasoglu et al. [2]	6059.71427196	6090.52614259	6064.33605261	11.28785324
This study	6059.714	6410.087	6095.32	88.270
Speed reducer				
Method	min	max	mean	std
Gandomi et al [6]	3000.9810	3009	3007.1997	4.9634
Akay et al.[1]	2997.058412	–	2997.058412	–
Brajevic et al. [3]	2994.471066	–	2994.471072	0.00000598
Baykasoglu et al. [2]	2996.372698	2996.669016	2996.514874	0.09
This study	2996.218	2996.218	2996.218	0
Spring design				
Method	min	max	mean	std
Gandomi et al [7]	0.01266522	0.0168954	0.01350052	0.001420272
Akay et al.[1]	0.012665	–	0.012709	0.012813
Brajevic et al. [3]	0012665	–	0.012683	0.00000331
Baykasoglu et al. [2]	0.0126653049	0.0000128058	0.0126770446	0.0127116883
This study	0.01266523	0.0129087	0.01269661	5.402618*10e^{-5}

[3] Ivona Brajevic and Milan Tuba. An upgraded artificial bee colony (abc) algorithm for constrained optimization problems. *Journal of Intelligent Manufacturing*, 24(4):729–740, 2013.

[4] Radovan R Bulatović, Goran Bošković, Mile M Savković, and Milomir M Gašić. Improved cuckoo search (ics) algorithm for constrained optimization problems. *Latin American Journal of Solids and Structures*, 11(8):1349–1362, 2014.

[5] JZ Cha and RW Mayne. Optimization with discrete variables via recursive quadratic programming: Part 2—algorithm and results. *Journal of Mechanisms, Transmissions, and Automation in Design*, 111(1):130–136, 1989.

[6] Amir Hossein Gandomi, Xin-She Yang, and Amir Hossein Alavi. Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. *Engineering with computers*, 29(1):17–35, 2013.

[7] Amir Hossein Gandomi, Xin-She Yang, Amir Hossein Alavi, and Siamak Talatahari. Bat algorithm for constrained optimization tasks. *Neural Computing and Applications*, 22(6):1239–1255, 2013.

[8] Uroš Mlakar, Iztok Fister Jr., and Iztok Fister. Hybrid self-adaptive cuckoo search for global optimization. *Swarm and Evolutionary Computation*, 29:47 – 72, 2016.

[9] Shyue-Jian Wu and Pei-Tse Chow. Genetic algorithms for nonlinear mixed discrete-integer optimization problems via meta-genetic parameter optimization. *Engineering Optimization+ A35*, 24(2):137–159, 1995.

[10] Xin-She Yang and Suash Deb. Cuckoo search via lévy flights. In *Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on*, pages 210–214. IEEE, 2009.

[11] Xin-She Yang and Suash Deb. Multiobjective cuckoo search for design optimization. *Computers & Operations Research*, 40(6):1616 – 1624, 2013. Emergent Nature Inspired Algorithms for Multi-Objective Optimization.

[12] Chun Zhang and Hsu-Pin Wang. Mixed-discrete nonlinear optimization with simulated annealing. *Engineering Optimization*, 21(4):277–291, 1993.

Weights Optimization in Statistical Machine Translation using Modified Genetic Algorithm

Jani Dugonik
University of Maribor, Faculty of Electrical Engineering and Computer Science
jani.dugonik@um.si

ABSTRACT

Translations in a statistical machine translations are generated on the basis of statistical models. These models are weighted, and different models' weights gives different translations. The problem of finding a good translation can be regarded as an optimization problem. One way to find good translations is to find optimal models' weights. In this paper we present the usage of the genetic algorithm with roulette wheel selection to find the optimal models' weights. Experiments were performed using well-known corpora and the most used translation metric in the SMT field. We compared the modified genetic algorithm with the basic genetic algorithm and the state-of-the-art method. The results show improvement in the translation quality and are comparable to the related state-of-the-art method.

Keywords

Genetic Algorithm, Statistical Machine Translation, Weight Optimization, Roulette Wheel Selection

1. INTRODUCTION

Machine translation (MT) can ease the work of a translator. The most studied and used MT method is the statistical machine translation (SMT) [2]. SMT is based on statistical methods which were originally used for translating single words (word-based translation). Now they have progressed to the level of translating sequences of words called phrases (phrase-based translation). Currently the most successful SMT systems are based on phrase-based translation [20]. Translations in SMT are generated on the basis of statistical models. Each model is weighted, and different models' weights provide various translations, which can be evaluated by translation error metrics. The problem of finding a good translation can be regarded as an optimization problem. The optimization can be done with models' weights. There are various methods for solving this optimization problem and in this paper we used a genetic algorithm (GA) [18]. It is based on a natural selection process that mimics biological evolution. The GA algorithm repeatedly modifies a population of individual solutions by relying on bio-inspired operators such as mutation, crossover and selection.

The main goal of this paper was to build two SMT systems for the language pair English-Slovenian, and to improve translation quality with finding optimal weights using a modified genetic algorithm. The used translation error metric was bilingual evaluation understudy (BLEU) [19].

The remainder of the paper is organized as follows. Section 2 presents related work. In the Section 3 we will describe some background of the problem and in Section 4 we will describe the basic and modified GA algorithms. Experiments are presented in Section 5. We conclude this paper with Section 6 where we give our conclusion.

2. RELATED WORK

The ability to optimize models' weights according to a translation error metric has become a standard assumption in SMT, due to the wide-spread adoption of Minimum Error Rate Training (MERT) [16, 3]. Authors in [16] analyzed various training criteria which directly optimize translation quality. These training criteria make use of an automatic evaluation metrics. They describe a new algorithm for efficient training an unsmoothed error count. The results in the paper show that significantly better results can often be obtained if the final evaluation criterion is taken directly into account as part of the training procedure.

The problems with MERT can be addressed through the use of surrogate loss functions. The Margin Infused Relaxed Algorithm (MIRA) [23, 10, 9, 8, 13] employs a structured hinge loss and is an instance of *online* learning. In order to improve generalization, the average of all weights seen during learning is used on unseen data. D. Chiang in [10] took advantage of MIRA's online nature to modify each update to better suit SMT. The cost is defined using a pseudo-corpus BLEU which tracks the n-gram statistics of the model-best derivations from the last few updates. This modified cost matches corpus BLEU better than Add-1 smoothing but also makes cost time-dependent.

As far as we know, there is only one published method for tuning SMT based on EAs. In [11] authors used a basic differential evolution algorithm to see how evolutionary algorithms behave in the field of statistical machine translation. The results were comparable to the state-of-the-art

methods.

3. BACKGROUND

The first ideas of SMT were introduced in 1949 [24]. SMT was re-introduced in the late 1980s and early 1990s by researchers at IBM's Thomas J. Watson Research Center [6, 5, 7] and has contributed to the significant resurgence in interest in MT. Nowadays it is by far the most widely studied MT method.

The idea behind SMT comes from information theory. A text is translated according to the probability distribution $p(e|f)$ where a string e in the target language is the translation of a string f in the source language.

One approach to model the probability distribution $p(e|f)$ is to apply Bayes Theorem:

$$p(e|f) = p(f|e) \cdot p(e) \tag{1}$$

where the translation model $p(f|e)$ is the probability that the source string f is the translation of the target string e, and the language model $p(e)$ is the probability of seeing that target string e. This decomposition is attractive as it splits the problem into the two subproblems. Finding the best translation e^* is done by searching among all possible translations E for the translation with the highest probability using the following equation:

$$e^* = \arg\max_{e \in E} p(e|f) = \arg\max_{e \in E} p(f|e) \cdot p(e) \tag{2}$$

Performing this search efficiently is the work of a MT decoder that uses the foreign string, heuristics and other methods to limit the search space and at the same time keeping acceptable quality. This trade-off between quality and time usage can also be found in speech recognition. A text is typically translated sentence by sentence, but even this is not enough. In SMT models are based on n-grams, where a n-gram is a sequence of n items from a given sequence of text. The statistical translation models were initially word based, but significant advances were made with the introduction of phrase based models. Language models are typically approximated by smoothed n-gram models, and similar approaches have been applied to the translation models, but there is additional complexity due to different sentence lengths and word orders in the languages. Therefore there are more advanced models that deal with these additional complexity. Phrase and word penalty models ensures that the translations/phrases are not too long or too short. Lexical reordering model conditions reordering on the actual phrases. By limiting the reordering, we can speed up the decoder as well as increase the translation quality. The translation quality is considered to be the correspondence between a machine and professional human (reference) translation. One of the first metrics to achieve a high correlation with human judgments is the BLEU metric, and it is the more popular metric in SMT. The BLEU metric always output a real-valued number between 0 and 1. This value indicates how similar are the machine and reference translations. Values closer to 1 represent the more similar texts, however, no machine translations will attain a score of 1.

4. GENETIC ALGORITHM

Encoding of chromosomes is one of the problems, when you are starting to solve problem with GA. Encoding very depends on the problem. Our problem is to find optimal models' weights. In our algorithm, vector of weights is defined individual $\vec{x} = x_1, x_2, ..., x_D$ where D is the dimension of the problem. The dimension D is the number of models' weights.

The initial population is consisted of Np individuals and is generated randomly between min and max values. As seen in Algorithm 1, the simplest form of GA involves three types of operators: crossover, mutation, and selection.

The crossover operator uses a crossover probability (Cr) to cross over randomly selected individuals $\vec{s_1}$ and $\vec{s_2}$ from the population to form a new individual ($\vec{x}$) using the following equation:

$$x_j = \begin{cases} s1_j & rand() < Cr, \\ s2_j & else. \end{cases} \tag{3}$$

where j is a value between 1 and D and $rand()$ returns a real value between 0 and 1. If no crossover was performed, the offspring x is an exact copy of one of the parents.

The mutation operator uses a mutation probability (F) to mutate new offspring at each position using the following equation:

$$x_j = \begin{cases} x_j + rand_1() & rand_2() < F, \\ x_j & else. \end{cases} \tag{4}$$

where j is a value between 1 and D and $rand_r()$, $r = \{1, 2\}$, returns a real value between 0 and 1. After the crossover and mutation the weights can fall out of the interval [min, max].

The selection operator selects solutions from the newly created population and the current population. Solutions with better fitness survives into the next generation g. The end condition is a maximum number of generations(G).

4.1 Roulette Wheel Selection

Because the basic GA selects from all solutions (individuals) in the population, we decided to use the roulette wheel (RW) selection [1] to speed up the process. The basic part of the selection process is to stochastically select from one generation to create the basis of the next generation. The requirement is that the fittest individuals have a greater chance of survival than weaker ones. This replicates nature in that fitter individuals will tend to have a better probability of survival and will go forward to form the mating pool for the next generation. Weaker individuals are not without a chance. In nature such individuals may have genetic coding that may prove useful to future generations.

Algorithm 1 Genetic Algorithm

```
Initialization
Evaluate the initial population
for g = 1 to G do
    for i = 1 to Np do
        Crossover
        Mutation
        Place newly generated individual in a new population
    end for
    Selection
end for
```

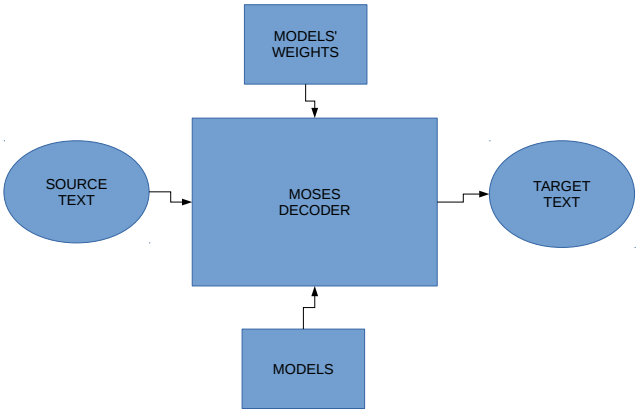


Figure 1: SMT system.

Instead of iterating over all solutions, we chose 50 % of the population **P** which will then be recombined and mutated, as seen in Algorithm 2. First we calculate *sum* of all individuals' fitness values. Then we calculatet $value = sum * rand()$, where *rand()* returns a random value between 0 and 1. After that, we iterate over all the population **P**, and calculate *value*. When the new *value* is lower or equal than zero, we return the current individual.

Algorithm 2 Roulette Wheel Selection

```
Calculate value
for i = 1 to Np do
    ind = P_i
    value = value − f(ind)
    if value ≤ 0 then
        return ind
    end if
end for
```

5. EXPERIMENTS

In this section we will describe the corpus preparation, how we built the two SMT systems, and the optimization settings. For each method (baseline, MERT, GA, GA_{RW}) we got a file containing weights. The two SMT systems used these weights to translate the test set. The translated sets were then evaluated using the BLEU metric, as shown in Table 1.

5.1 Corpus Preparation

In our experiments we used English and Slovenian JRC-ACQUIS Multilingual Parallel Corpora [21]. In this corpora the sentences are aligned which means the first sentence from the English text is the translation of the first sentence from the Slovenian text and so on. We split each corpus in three parts: train, tuning, and test sets. The train set was used to train our SMT systems and consisted of 6,500 sentences or 184,072 words in English set and 158,768 words in Slovenian set. The tuning set was used to optimize the two SMT systems and consisted of 500 sentences or 13,526 words in English set and 12,128 words in Slovenian set. The test set was used to evaluate our SMT systems and consisted of 3,000 sentences or 86,145 words in English set and 76,879 words in Slovenian set.

5.2 Building the SMT systems

To successfully build the SMT system, seen in Figure 1, we created models from the training set, described in the previous subsection, and Moses toolkit [15]. Moses toolkit is an open-source toolkit for SMT which contains the SMT decoder and a wide variety of tools for training, tuning and applying the system to many translation tasks. Using Moses toolkit we created language and translation models. The language model was a 5-gram model with improve Kneser-Ney smoothing using IRST language modeling (IRSTLM) toolkit [12]. The translation model was built using grow-diag-final-and alignment from GIZA++ [17]. Our SMT systems were extended with four advanced models: distortion, lexalized reordering, word, and phrase penalty models. This gave us six models.

5.3 Optimization Settings

The following settings were used for the optimizations of both SMT systems: $D = 14$, $min = -1$, $max = 1$, $Np = 15$, $G = 70$, $F = 0.5$ and $Cr = 0.7$. We can see the tuning process in the Figure 2. Input to tuning methods are the tuning set, models and models' weights, and the output are the new models' weights.

5.4 Results

Our experiments were based on experiments in [3]. Authors compared the old and new implementations of MERT and if they are similarly effective. They also measured the computation time. In our experiments we compared the basic GA with GA with Roulette Wheel Selection (GA_{RW}). Each of the optimization method (MERT, GA, GA_{RW}) was ran one time, as in [3], and compared to each other. All three methods were tuned on the same system using the same tuning corpus, and the BLEU scores are shown in Table 1. As we can see, the BLEU scores for GA_{RW} are better than the basic GA, and comparable with MERT. From the Table 2 we can see the tuning time for each method where GA_{RW} is 50 % faster as the basic GA, and comparable with MERT.

5.5 Discussion

Since the optimization is a part of the training, both of them are done *offline* which means the tuning time does not affect the actual translating. The actual translating is done *online* and the time to translate text depends on the number of words. But usually one sentence is translated in 1 second. The GA_{RW} method is computationally faster because

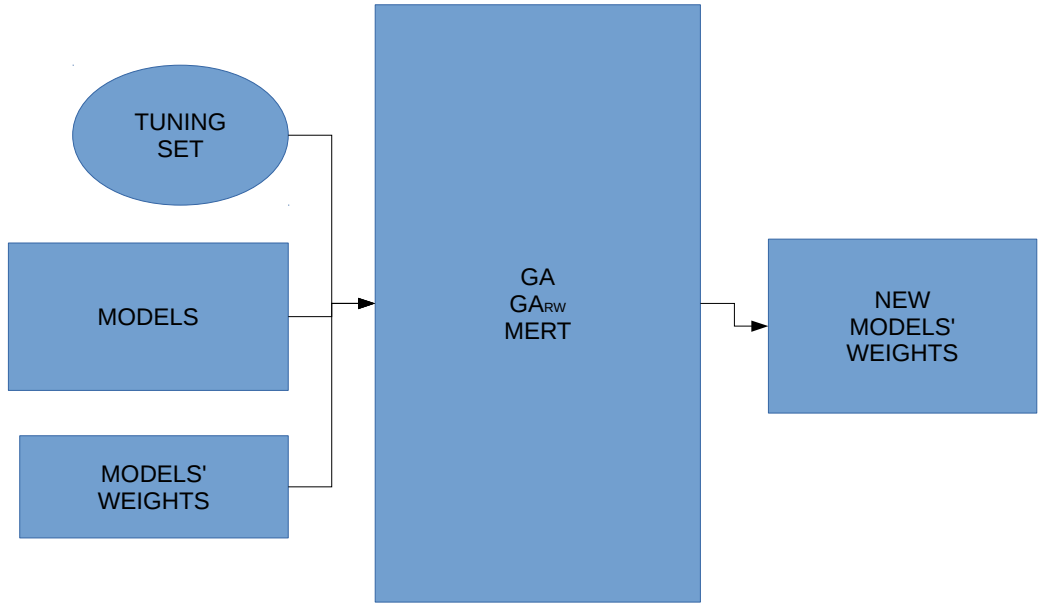


Figure 2: Tuning process.

Table 1: BLEU scores on test corpora.

System	BLEU (%) ↑	
	Slovenian → English	English → Slovenian
MERT	33.18	23.34
GA	32.15	23.39
GA$_{RW}$	**33.19**	**23.49**

Table 2: Tuning time on tuning set.

System	Time (minutes)
MERT	97
GA	182
GA$_{RW}$	91

it makes 50 % less evaluations than the basic GA. The current difference is 1.5 hours but with much more larger sets the difference could be in 10 hours or even days. The BLEU score represents the similarity between the machine translated text and human translated text. At this moment it is impossible to achieve a BLEU score of 100 % because that would mean that the two texts are identical. Even human translators can not produce the same translations. Thats why every % counts and the difference $\ll 1$ % is actually a good improvement in the translation quality.

6. CONCLUSIONS

We successfully built two SMT systems using JRC-Acquis corpora for Slovenian-English language pair. We implemented the basic GA and added the Roulette Wheel Selection to speed up the process. We showed that using the Roulette Wheel Selection the tuning time was shortened while still maintaining the comparable translation quality. For further research we can use more advanced EAs, such as jDE [4], L-SHADE [22], nature-inspired algorithms [14] to improve the translation quality and shorten the tuning time.

7. REFERENCES

[1] O. Al Jadaan, L. Rajamani, and C. Rao. Improved selection operator for ga. *Journal of Theoretical & Applied Information Technology*, 4(4), 2008.

[2] T. F. Albat. US Patent 0185235, Systems and Methods for Automatically Estimating a Translation Time, 2012.

[3] N. Bertoldi, B. Haddow, and J.-B. Fouet. Improved Minimum Error Rate Training in Moses. *ACL*, pages 160–167, 2009.

[4] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE Trans. Evolutionary Computation*, 10(6):646–657, 2006.

[5] P. Brown, J. Cocke, S. D. Pietra, V. D. Pietra, F. Jelinek, J. D. Lafferty, R. L. Mercer, and P. Roosing. A statistical approach to machine translation. *Computational Linguistics*, pages 79–85,

1990.

[6] P. Brown, J. Cocke, S. D. Pietra, V. D. Pietra, F. Jelinek, R. L. Mercer, and P. Roosing. A statistical approach to language translation. *Association for Computational Linguistics*, pages 71–76, 1988.

[7] P. Brown, S. D. Pietra, V. D. Pietra, and R. Mercer. The mathematics of statistical machine translation: parameter estimation. *Computational Linguistics*, pages 263–311, 1993.

[8] C. Cherry and G. Foster. Batch Tuning Strategies for Statistical Machine Translation. *NAACL*, 2012.

[9] D. Chiang, K. Knight, and W. Wang. 11,001 new features for statistical machine translation. *HLT-NAACL*, pages 218–226, 2009.

[10] D. Chiang, Y. Marton, and P. Resnik. Online large-margin training of syntactic and structural translation features. *EMNLP*, pages 224–233, 2008.

[11] J. Dugonik, B. Bošković, M. S. Maučec, and J. Brest. The usage of differential evolution in a statistical machine translation. In *2014 IEEE Symposium on Differential Evolution, SDE 2014, Orlando, FL, USA, December 9-12, 2014*, pages 89–96, 2014.

[12] M. Federico, N. Bertoldi, and M. Cettolo. IRSTLM: an open source toolkit for handling large scale language models. In *INTERSPEECH 2008, 9th Annual Conference of the International Speech Communication Association*, pages 1618–1621, 2008.

[13] E. Hasler, B. Haddow, and P. Koehn. Margin Infused Relaxed Algorithm for Moses. *The Prague Bulletin of Mathematical Linguistics*, pages 69–78, 2011.

[14] I. F. Jr., X.-S. Yang, I. Fister, J. Brest, and D. Fister. A brief review of nature-inspired algorithms for optimization. *CoRR*, abs/1307.4186, 2013.

[15] P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. J. Dyer, O. Bojar, A. Constantin, and E. Herbst. Moses: Open source toolkit for statistical machine translation. *ACL Demo and Poster Session*, 2007.

[16] F. J. Och. Minimum Error Rate Training for Statistical Machine Translation. *ACL*, pages 160–167, 2003.

[17] F. J. Och and H. Ney. Improved Statistical Alignment Models. In *ACL*, pages 440–447, 2000.

[18] A. Otman and A. Jaafar. Article:a comparative study of adaptive crossover operators for genetic algorithms to resolve the traveling salesman problem. *International Journal of Computer Applications*, 31(11):49–57, October 2011. Full text available.

[19] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. BLEU: a method for automatic evaluation of machine translation. *ACL*, pages 311–318, 2002.

[20] L. Specia. Fundamental and New Approaches to Statistical Machine Translation, 2010.

[21] R. Steinberger, B. Pouliquen, A. Widiger, C. Ignat, T. Erjavec, D. Tufis, and D. Varga. The JRC-Acquis: A Multilingual Aligned Parallel Corpus with 20+ Languages. *LREC*, 2006.

[22] R. Tanabe and A. S. Fukunaga. Improving the search performance of SHADE using linear population size reduction. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2014, Beijing, China, July 6-11, 2014*, pages 1658–1665, 2014.

[23] T. Watanabe, J. Suzuki, H. Tsukada, and H. Isozaki. Online large-margin training for statistical machine translation. *EMNLP-CoNLL*, pages 764–773, 2007.

[24] W. Weaver. Translation. *Machine Translation of Languages*, 1955.

StuCoSReC

University of Primorska Press
www.hippocampus.si
ISBN *978-961-6984-37-9*
Not for resale