Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Ján Antala

Minimalizácia štýlových predpisov v jazyku CSS

Evolučné algoritmy

máj 2013
https://github.com/janantala/genetic-css-minify

Obsah

Zoznam obrázkov

1 Zadanie

Pomocou GA sa pokúste minimalizovať veľkosť štýlových predpisov v jazyku CSS, pričom vnútorná sémantika štýlového predpisu sa nezmení.

Experimentujte s parametrami ako elitárstvo, pravdepodobnosť mutácie a kríženia, či veľkosť populácie. Dosiahnuté výsledky experimentov zanalyzujte a zdokumentujte.

2 Úvod

Minimalizovanie prenášaných dát cez internet je nesmierne dôležité. Ich nevyhnutnosť vzrastá najmä s nástupom mobilných zariadení a čoraz väčšom využívaní drahých mobilných dát. Priemerná veľkosť webovej stránky v roku 2012 bola 1.25 MB a pri súčasnom trende bude jej veľkosť 2 MB v roku 2014 [4]. Jednou z možností minimalizovania prenosov je minifikácia zasielaného CSS súboru.

Kaskádové štýly („Cascading Style Sheets" - CSS) je jednoduchý mechanizmus na vizuálne formátovanie webových dokumentov [3]. Bol navrhnutý štandardizačnou organizáciou W3C. Zatiaľ boli vydané dve verzie špecifikácie: CSS1 a CSS2, v súčasnosti sa pracuje na verzii CSS3. Hlavným cieľom používanie CSS je oddelenie vzhľadu dokumentu od jeho štruktúry a obsahu.

Kaskádový štýl tvorí súbor pravidiel, ktorý sa skladá zo selektorov (napr. p, h2...) a deklarácii (napr. color, padding, margin, font...):

```
p {
    color: black;
    padding: 5px;
    margin-bottom: 5px;
    font-size: 14px;
}

h2 {
    color: black;
    padding: 5px;
    margin-bottom: 10px;
    font-size: 22px;
}
```

Cieľom minifikácie je zmenšiť veľkosť predpisu nielen odstránením bielych znakov, ale aj spojením duplicitných deklarácií pod jeden selektor. Výsledok minifikácie bude tvoriť nasledujúci stav:

```
p, h2 {
    color: black;
    padding: 5px;
}

p {
    margin-bottom: 5px;
    font-size: 14px;
}

h2 {
    margin-bottom: 10px;
    font-size: 22px;
}
```

V tomto jednoduchom príklade sme minifikáciou ušetrili veľkosť 27 znakov. Pri komplexnejších kaskádových štýloch sa ušetrená veľkosť bude zväčšovať. Spájanie selektorov však nemusí byť vždy úplne efektívne. Príkladom je použitie príliš dlhých názvov, ktoré bude dlhšie ako samotné deklarácie, a následným vytvorením nového spojeného selektora.

Problém je známy ako „Minimum Biclique Covering" (minimálny bipartitný graf), ktorého riešenie je NP zložité [1, 2]. Na minifikáciu CSS súborov je použitý genetický algoritmus. Spôsob realizácie algoritmu je popísaný v nasledujúcej kapitole.

3 Algoritmus

Algoritmus je naprogramovaný v jazyku JavaScript pre systém node.js. Na spracovanie CSS súburu do podoby objektov je použitá npm knižnica css[1].

3.1 Fitness

Fitness funkcia aktuálneho kaskádového štýlu sa vypočíta ako rozdiel veľkosti (počtu znakov) pôvodného CSS súboru a veľkosti aktualného štýlu. Čím viac sa nám podarilo kaskádový štýl minifikovať, tým bude mať väčšiu fitness.

```
fitness = pôvodná veľkosť CSS - aktuálna veľkosť CSS
```

3.2 Selekcia

Na selekciu jedincov z populácie je možné vybrať z nasledujúcich algoritmov: turnaj alebo ruleta. Výber je možné realizovať v úvodných nastaveniach programu.

3.2.1 Turnaj

Selekcia turnajom je implementovaná pre veľkosť turnaja dva. Prebieha tak, že sa z populácie náhodne zvolia dvaja jedinci a z nich sa na základe fitness vyberie ten lepší.

3.2.2 Ruleta

Selekcia ruletou prebieha tak, že každému jedincovi prislúcha taká pomerná časť z celkového súčtu fitness všetkých jedincov, akú má fitness on sám. Čím má jedinec väčšiu fitness oproti ostatným v populácii, tým je jeho šanca na výber väčšia.

3.3 Kríženie

V algoritme je kríženie celkom komplexná záležitosť. Keďže nový jedinec musí obsahovať všetky pravidlá, tak je potrebné ho opraviť. Kríženie medzi dvoma jedincami prebieha tak, že sa zoberie polovica dĺžky prvého jedinca a pridá sa k nemu druhá polovica z druhého jedinca. Aby nechýbali niektoré pravidlá, tak sa k novému jedincovi pridá ešte druhá polovica z prvého jedinca a následne sa zlúčia všetky duplicitné selektory. Podobne to prebieha aj s druhým jedincom.

Samotnú pravdepodobnosť kríženia jedincov je možné nastaviť v nastaveniach algoritmu, pokiaľ kríženie neprebehne, tak sa vrátia pôvodní jedinci.

3.4 Mutácia

V programe sú použité dva spôsoby mutácie: mutácia rozdelením selektorov a mutácia spájaním. Zakaždým sa náhodné zvolí typ mutácie a pravidlo kaskádového štýlu, pre ktoré sa mutácia uplatňuje. Pravdepodobnosti mutácie je možné nastaviť.

[1] https://npmjs.org/package/css

3.4.1 Mutácia delenia

Mutácia rozdelením je aplikovaná na zložený selektor. V prípade náhodneho výberu jednoduchého selektoru sa nič nevykoná.

```
p, h2 {
    color: black;
    padding: 5px;
}
```

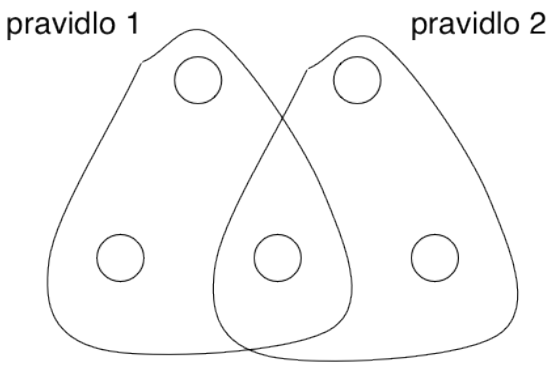
Po jej aplikovaní na kaskádový štýl sa vytvoria dva nové selektory s rovnakými deklaráciami. V prípade existencie rovnakého selektora sa ich deklaracie spoja a nevznikajú tak duplicity. Výsledok mutácie bude nasledujúci:

```
p {
    color: black;
    padding: 5px;
}

h2 {
    color: black;
    padding: 5px;
}
```

3.4.2 Mutácia spájania

V prípade mutácie spájaním sa náhodne zvolia dva selektory z kaskádového štýlu a následne sa zisťuje, ktoré deklarácie sú rovnaké a je ich tak možné spojiť pod jeden zložený selektor. Mutácia spájaním je opakom mutácie rozdelením.



Obr. 1: Možnosť spojenia dvoch pravidiel

3.5 Elitizmus

Program umožňuje nastaviť množstvo najlepších jedincov z populácie, ktoré sa automaticky presunie do nasledujúcej generácie.

3.6 Koncový stav

Keďže sa jedná o špecifický druh úlohy tak koncový stav dopredu nepoznáme. Ukončenie algoritmu je tak možné dvoma spôsobmi: vyčerpaním maximálneho počtu generácii alebo prekročením limitu generácii, kedy sa fitness najlepšieho jedinca z populácie nezlepší. Oba parametre je možné nastaviť.

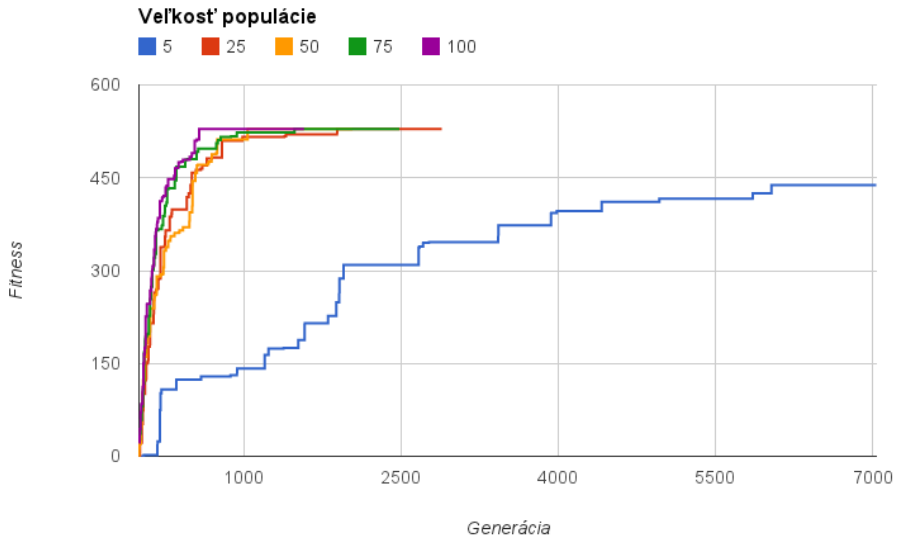
4 Experimenty

Experimenty boli vykonávané na reálnych CSS súboroch získaných na webovej stránke denníka SME. Menili sa rôzne parametre ako je veľkosť populácie, spôsob selekcie, elitizmus alebo pravdepodobnosť kríženia či mutácie a sledoval sa ich vplyv na vývoj fitness najlepšieho jedinca v populácii.

4.1 Veľkosť populácie

Experimenent veľkosti populácie prebiehal s nasledujúcimi parametrami, pričom sa skúmal vzťah medzi veľkosťou populácie a vývoja fitness.

```
var options = {
    populationLength: [5, 25, 50, 75, 100],
    maxGenerations: 10000,
    roundOut: 1000,
    mutateLine: 0.2,
    crossover: 0.2,
    elites: 2,
    selection: 'tournament'
};
```



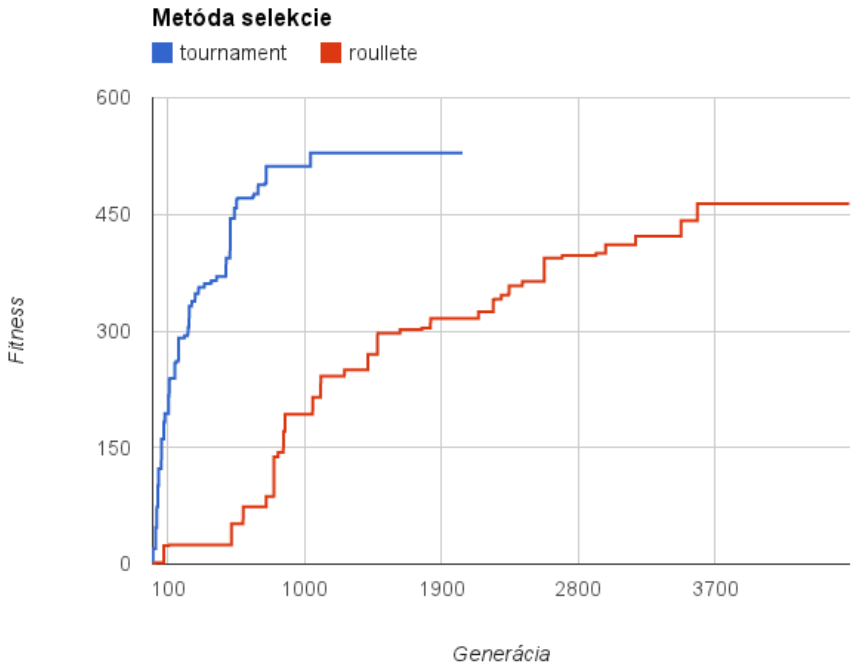
Obr. 2: Vývoj fitness v závislosti od veľkosti populácie

Ako najlepšia veľkosť populácie vzhľadom na rýchlosť zlepšovania fitness a dĺžku vykonávania programu vyznela populácia veľkosti 50. Pri malej populácii bola rýchlosť zvyšovania fitness pomalá a pri veľkej populácii sa jej rýchlosť už nezvyšovala, naopak zvyšovala sa dĺžka vykonávania programu.

4.2 Selekcia

Experimenent selekcie prebiehal s nasledujúcimi parametrami, pričom sa skúmal vzťah medzi druhom selekcie a vývoja fitness.

```
var options = {
    populationLength: 50,
    maxGenerations: 10000,
    roundOut: 1000,
    mutateLine: 0.2,
    crossover: 0.2,
    elites: 2,
    selection: ['tournament', 'roullete']
};
```



Obr. 3: Vývoj fitness v závislosti od metódy selekcie

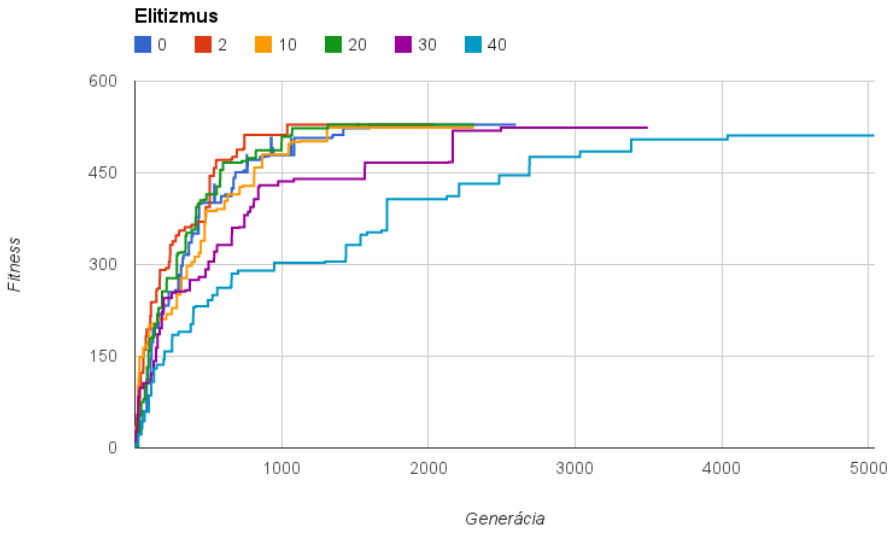
Lepší spôsob selekcie je selekcia turnajom. Selekcia pomocou rulety je časovo výrazne pomalšia a rýchlosť zlepšovania fitness najlepšieho jedinca v populácii taktiež dosahuje horšie výsledky ako selekcia turnajom.

4.3 Elitizmus

Experimenent elitizmu prebiehal s nasledujúcimi parametrami, pričom sa skúmal vzťah medzi počtom elít presunutých do nasledujúcej generácie a vývoja fitness.

```
var options = {
    populationLength: 50,
    maxGenerations: 10000,
    roundOut: 1000,
    mutateLine: 0.2,
    crossover: 0.2,
    elites: [0, 2, 10, 20, 30, 40]
    selection: tournament
```

```
};
```



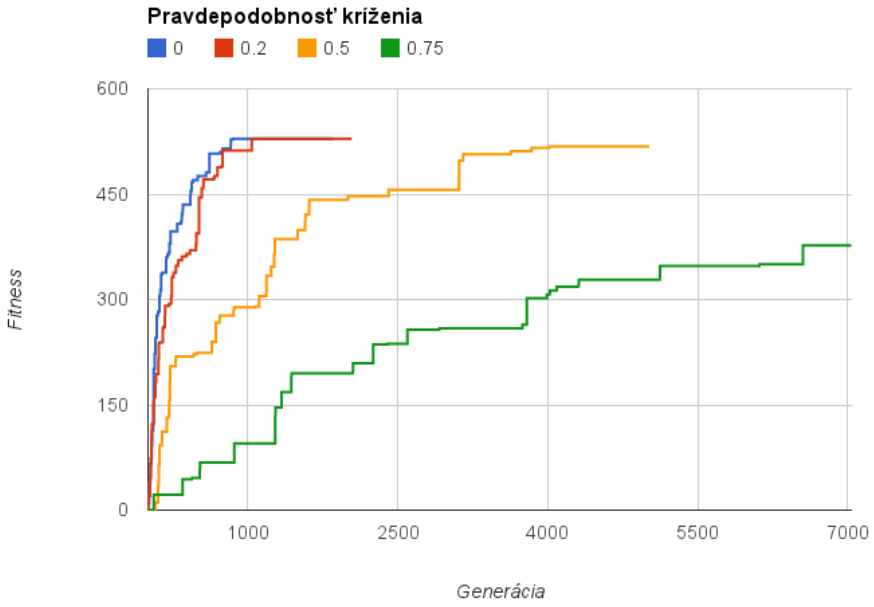
Obr. 4: Vývoj fitness v závislosti od veľkosti elitizmu

Z experimentu vyplýva, že v prípade malého presunutého počtu elít (v experimente sú to dvaja najlepší jedinci) do nasledujúcej generácie sa vývoj fitness zlepšuje najlepšie. Pokiaľ je elitizmus nulový, tak najlepšia fitness môže aj klesnúť. V prípade väčšieho elitizmu sa zlepšovanie fitness spomaluje.

4.4 Pravdepodobnosť kríženia

Experimenent kríženia prebiehal s nasledujúcimi parametrami, pričom sa skúmal vzťah medzi pravdepodobnosťou kríženia a vývoja fitness.

```
var options = {
    populationLength: 50,
    maxGenerations: 10000,
    roundOut: 1000,
    mutateLine: 0.2,
    crossover: [0, 0.2, 0.5, 0.75, 1]
    elites: 2
    selection: tournament
};
```



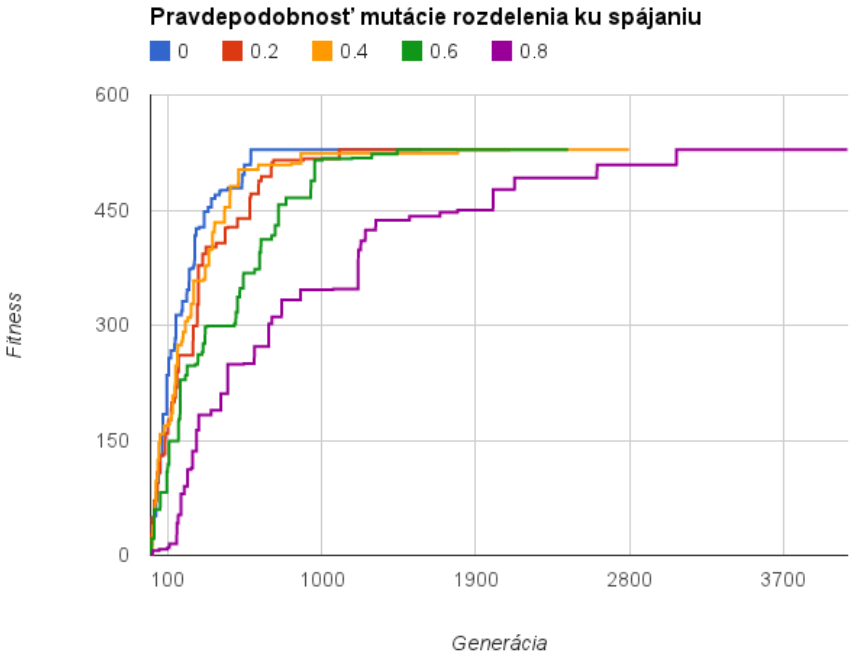
Obr. 5: Vývoj fitness v závislosti od pravdepodobnosti kríženia jedincov

Experimentom sa overil predpoklad, že pokiaľ je v algoritme zahrnuté kríženie, tak rýchlosť zvyšovania fitness sa zmenšuje. Je to spôsobené najmä tým, že po krížení jedincov často vznikne nový s väčšou veľkosťou ako bol pôvodný, pretože je potrebné zachovať všetky pravidlá selektorov a nového jedinca opravovať. Navyše so zvyšovaním pravdepodobnosti kríženia sa zvyšuje a dĺžka vykonávania programu.

4.5 Pravdepodobnosť mutácie

Experimenent mutácie prebiehal s nasledujúcimi parametrami, pričom sa skúmal vzťah medzi pravdepodobnosťou mutácie a vývoja fitness. Mutoval každý zvolený jedinec a menil sa spôsob mutácie medzi rozdelovaním a spájaním selektorov pomocou parametra mutateLine. Ten označuje interval medzi rozdelovaním a spájaním, teda od 0 po mutateLine je pravdepodobnosť rozdelenia, od mutateLine po 1 je pravdepodobnosť spojenia.

```
var options = {
    populationLength: 50,
    maxGenerations: 10000,
    roundOut: 1000,
    mutateLine: [0, 0.2, 0.4, 0.6, 0.8, 1]
    crossover: 0
    elites: 2
    selection: tournament
};
```



Obr. 6: Vývoj fitness v závislosti od pravdepodobnosti mutácie jedincov

Výsledkom experimentu je, že fitness najlepšie stúpa pri menšej pravdepodobnosti delenia a väčšej pravdepodobnosti spájania, aj keď presná hodnota dosť závisí od pôvodného CSS súboru.

5 Obmedzenia programu

Algoritmus v súčasnosti nepodporuje CSS3 selektory, ktoré vytvárajú vlastný priestor pôsobnosti, ale pracuje len z globálnymi selektormi. Medzi nepodporované patria napríklad @media alebo @keyframes a tak sú z pôvodného súboru ignorované.

Príklad nepodporovaných CSS pravidiel je nasledujúci, do algoritmu bude zahrnutý len prvý selektor „span”.

```
span { color: white; }

@media only screen
and (min-device-width : 768px)
and (max-device-width : 1024px)
and (orientation : portrait) {
    span { color: red; }
}
```

6 Možné vylepšenia

Okram dopracovania kompletnej podpory CSS3 selektorov existujú aj ďalšie vylepšenia programu. Medzi najdôležitejšie patrí zjednotenie duplicitných zápisov deklarácii zapísaných rôznymi spôsobmi. Príkladom je použitie farby:

```
p, div, span {
    color: white;
    color: #FFFFFF;
    color: rgb(255,255,255);
}
```

Nastavenie farby by bolo možné aj jednoduchším zápisom:

```
p, div, span { color: white; }
```

Farba však nie je jediným problémom. Ďalšími sú deklarácie ako „margin", „padding" či „border", ktoré umožňujú vytvárať štýly pre každú stranu elementu a je ich možné zjednotiť do jednej. Problém reprezentuje nasledujúci príklad:

```
div {
    margin-top: 10px;
    margin-bottom: 10px;
    margin-left: 10px;
    margin-right: 10px;
}
```

Takýto zápis je možné zjednušiť pomocou jedného riadku:

```
div { margin: 10px; }
```

Dlhodobejším cieľom je zapracovanie projektu ako vlastnej úlohy do nástroja Grunt[2], ktorý je v súčasnosti veľmi populárnym pri vývoji webového front-endu. Úloha by automaticky minifikovala vytvorený CSS súbor.

[2] `http://gruntjs.com/`

7 Záver

Minifikácia CSS súborov bola úspešne testovaná na reálne nasadených projektoch. Použitím minifikátora sa podarilo zmenšiť ich celkovú veľkosť asi o 25%. Z experimov sa zistili nasledujúce vlastnosti:

- najvhodnejšia veľkosť populácie je 50 jedincov
- selekcia pomocou metódy turnaj
- do nasledujúcej generácie je vhodné preniesť malý počet najlepších jedincov
- kríženie nepoužívať, prípadne použiť len s malou pravdepodobnosťou
- použiť mutáciu spájania selektorov s väčšou pravdepodobnosťou ako mutáciu rozdelovania

V súčasnosti sa však čoraz viac začínajú používať CSS3 selektory, ktoré nie sú v súčasnej verzii podporované. Preto je potrebné ich do plnej verzie zapracovať.

Literatúra

[1] ORLIN, J. Contentment in graph theory: covering graphs with cliques, 1977.

[2] SCHREIBER, R. – MILOSAVLJEVIć, N. – ENE, A. Solving the Minimum Biclique Cover Problem in Practice. 2007. Dostupné z: `http://web.engr.illinois.edu/~ene1/talks/mbc-hp.pdf`.

[3] W3C. Cascading Style Sheets. Dostupné z: `http://www.w3.org/Style/CSS/`.

[4] WROBLEWSKI, L. Data Monday: Mobile, Responsive Design, & Web Page Sizes. February 4, 2013. Dostupné z: `http://www.lukew.com/ff/entry.asp?1681`.