AUTOMATED SLOGAN PRODUCTION USING A GENETIC ALGORITHM

Polana Tomašič
XLAB, Pot za Brdom 100, 1000 Ljubljana, Slovenia
and
Jožef Stefan International Postgraduate School, Ljubljana, Slovenia
polona.tomasic@xlab.si

Gregor Papa
Computer Systems Department
Jožef Stefan Institute, Ljubljana, Slovenia
and
Jožef Stefan International Postgraduate School, Ljubljana, Slovenia
gregor.papa@ijs.si

Martin Žnidaršič
Department of Knowledge Technologies
Jožef Stefan Institute, Ljubljana, Slovenia
and
Jožef Stefan International Postgraduate School, Ljubljana, Slovenia
martin.znidarsic@ijs.si

Abstract Invention of slogans is an intelligent and highly creative task. As such, it is a challenging problem for computational methods. In this paper we present our solution based on the use of linguistic resources and evolutionary computing.

Keywords: Computational creativity, Genetic algorithms, Slogan generation.

1. Introduction

Generation of slogans for companies and products is one of the less explored problems in the field of Computational Creativity. To our knowledge, there is only one scientific study dedicated particularly to slogan (and other creative sentences) generation, namely the BrainSup framework [13]. This approach requires the user to provide keywords, domain, emotions and similar properties of the slogans. This shrinks the huge search space of slogans and improves the quality of results. We, however, have aimed at a completely autonomous approach that is not influenced by the user in any way, apart from providing a short textual description of the target entity.

In this paper, we present our slogan generation procedure, which is based on a genetic algorithm (GA) [1]. Genetic algorithms ensure good coverage of the search space and are relatively often used in Computational Creativity. For instance, they have been successfully used for generating recipes [11], poetry [7] and trivial dialog phrases [10]. However, genetic algorithm has not previously been used for slogan generation. Our method is the first to use it for that purpose. It follows the BrainSup framework in the initial population generation phase, and it uses a collection of heuristic slogan functions in the evaluation phase.

The results of the experiments indicate some deficiencies of our method. The generated slogans nonetheless present a good starting point for brainstorming.

2. Resources

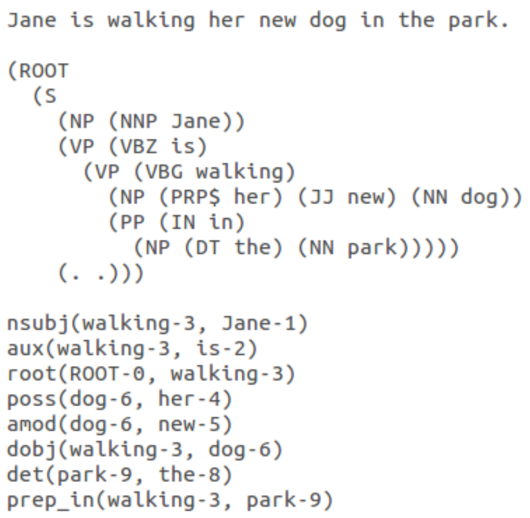
Our slogan generation method requires some linguistic and semantic resources for the generation of initial population:

- **Database of the existing slogans**
  The database of existing slogans serves as a basis for the initial population generation and for comparison with generated slogans. It contains famous slogans obtained from the Internet.

- **Database of the frequent grammatical relations**
  For the acquisition of the frequent grammatical relations between words in sentences we used the Stanford Dependencies Parser [8]. Stanford dependencies are triplets containing two words, called *governor* and *dependent*, and the name of the relation between them. The parser also provides part-of-speech (POS) tags and phrase structure trees. An example of its output is in Figure 1. To get representatives of frequent grammatical relations between words, we parsed 52,829 random Wikipedia pages, sentence by sentence, and obtained 4,861,717 different dependencies.

- **Database of the slogan skeletons**
  A slogan skeleton contains information about each position in the sentence - its POS tag and all its dependencies relations with other

words in the sentence. It does not contain any content words, only stop words. An example of a skeleton from [13] is in Figure 2. Skeletons were obtained by parsing existing slogans with the Stanford Dependencies Parser.



```
Jane is walking her new dog in the park.

(ROOT
  (S
    (NP (NNP Jane))
    (VP (VBZ is)
      (VP (VBG walking)
        (NP (PRP$ her) (JJ new) (NN dog))
        (PP (IN in)
          (NP (DT the) (NN park)))))
    (. .)))

nsubj(walking-3, Jane-1)
aux(walking-3, is-2)
root(ROOT-0, walking-3)
poss(dog-6, her-4)
amod(dog-6, new-5)
dobj(walking-3, dog-6)
det(park-9, the-8)
prep_in(walking-3, park-9)
```

Figure 1. Stanford dependencies parser's output for the sentence "Jane is walking her new dog in the park."

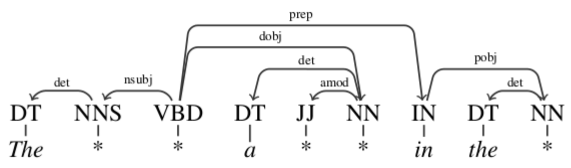


Figure 2. Example of a skeleton from [13].

3. Slogan Generation

In this section we describe our slogan generation approach in terms of its inputs, outputs and algorithmic steps. The whole procedure is shown in the Algorithm 1.

3.1 Extraction of the Keywords and the Main Entity

Target keywords are extracted from the input text using the Nodebox English Linguistics library [12]. The aim is to generate positive slogans.

Algorithm 1 SloganGenerator

Input: A textual description of a company or a product T, Size of the population S_{P}, Maximal number of iterations Max_iter, Crossover probability $p_{\mathrm{crossover}}$, Mutation probability p_{mutation}, Set of evaluation weights W.

Output: A set of generated slogans S.

```
1: $Keywords, Entity \Leftarrow$ GetKeywordsAndEntity($T$)
2: $P \Leftarrow$ CreateInitialPopulation($S_{\mathrm{IP}}, Keywords, Entity$)
3: Evaluate($P$)
4: while $Max\_iter > 0$ do
5:     ChooseParentsForReproduction($P$)
6:     Crossover($P, p_{\mathrm{crossover}}$)
7:     Mutation($P, p_{\mathrm{mutation}}$)
8:     DeleteSimilarSlogans($P$)
9:     while Size($P$) $< S_{\mathrm{P}}$ do
10:        AddARandomSeed($P$)
11:    end while
12:    Evaluate($P$)
13:    $Max\_iter \Leftarrow Max\_iter - 1$
14: end while
15: $S \Leftarrow P$
```

That is why all the sentences with the negative polarity in the input text are being removed. A sentence has a negative polarity if it contains words that are associated with negative emotions. After the removal, the most frequent words are selected as keywords. The main entity is usually the name of the company and is obtained by selecting the most frequent entity in the whole text using *nltk* library [2].

Example of the keywords and the entity, extracted from the Coca-Cola Wikipedia page:
keywords = ['win', 'celebrate', 'enjoy', 'follow', 'available', 'raspberry', 'snowy', 'cherry', 'famous', 'wonderful', 'familiar', 'sugar', 'sparkle', 'passion', 'beloved', 'fountain', 'bubble', 'enjoyment', 'drink', 'fluid', 'diet', 'candy', 'tour', 'beverage', 'contribution', 'dream', 'vision', ...]
entity = Coke

3.2 Generation of the Initial Population of Slogans

The procedure of generating the initial population of slogans is based on the BrainSup framework [13], with some modifications. It follows the steps in Algorithm 2. Skeletons are obtained from the database of slogan skeletons, and fillers are words from the database of all grammatical relations between words in sentences and must satisfy all predefined dependencies and POS tags. If there are any keywords in a set of all possible filler words, the algorithm assigns them higher priority for the selection phase. The main difference between our algorithm and the BrainSup method is in selection of filler words. We select them at random, while the BrainSup framework uses a beam search in the space of all possible lexicalizations of a skeleton to promote the words with the highest likelihood of satisfying the user specifications.

Algorithm 2 CreateInitialPopulation

Input: Size of the population S_{P}, a set of target keywords K, and the target entity E.

Output: A set of initial slogans S.

```
1: $S \Leftarrow \emptyset$
2: while $S_{\mathrm{IP}} > 0$ do
3:     $SloganSkeleton \Leftarrow$ SelectRandomSloganSkeleton()
4:     while not AllEmptySlotsFilled($SloganSkeleton$) do
5:         $EmptySlot \Leftarrow$ SelectEmptySlotInSkeleton($SloganSkeleton$)
6:         $Fillers \Leftarrow$ FindPossibleFillerWords($EmptySlot$)
7:         $FillerWord \Leftarrow$ SelectRandomFillerWord($Fillers$)
8:         FillEmptySlot($SloganSkeleton, FillerWord$)
9:     end while
10:    AddFilledSkeleton($S, SloganSkeleton$)
11:    $S_{\mathrm{P}} \Leftarrow S_{\mathrm{P}} - 1$
12: end while
```

3.3 Evaluation of Slogans

To order the slogans by their quality, an aggregated evaluation function was constructed. It is composed of 9 different sub-functions, each assessing a particular feature of a slogan with scores in the interval [0,1]. Parameter of the aggregation function is a list of 9 weights that sum to 1. They define the proportions of sub-functions in the overall score. In this subsection we give a short description for every one of them.

Bigram Function. In order to work with 2-grams, we obtained the data set of 1,000,000 most frequent 2-grams and 5000 most frequent words in Corpus of Contemporary American English (COCA) [3]. We assume that slogans containing many frequent 2-grams, are more likely to be semantically coherent. That is why they get higher bigram evaluation score.

Length Function. This function assigns score 1 to slogans with less than 8 words, and score 0 to longer ones.

Diversity Function. The diversity function evaluates a slogan by counting the number of repeated words. The highest score goes to a slogan with no repeated words. If a slogan contains identical consecutive words, it receives score 0.

Entity Function. It returns 1, if slogan contains the main entity, and 0, if it doesn't.

Keywords Function. If one up to half of the words in a slogan belong to the set of keywords, the keywords function returns 1. If a slogan doesn't contain any keyword, the score is 0. If more than half of the words in the slogan are keywords, the score is 0.75.

Word Frequency Function. This function prefers slogans with many frequent words, because slogans with many infrequent words are considered bad. The score is obtained by dividing the number of frequent words by the number of all words in the slogan. Word is considered to be frequent, if it is among 5000 most frequent words in COCA.

Polarity and Subjectivity Functions. To calculate the polarity and subjectivity scores based on the adjectives in the slogan, we used the *sentiment* function from *pattern* package for Python [4].

Semantic Relatedness Function. This function computes the relatedness between all pairs of content words in the slogan. Stop words are not taken into account. Each pair of words gets a score based on the path distance between corresponding synsets (sets of synonyms) in WordNet [9]. The final score is the sum of all pairs' scores divided by the number of all pairs.

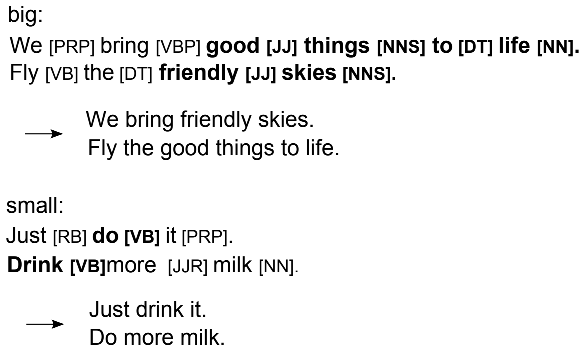
3.4 Production of a New Generation of Slogans

A list of all generated slogans is ordered descending with regard to the evaluation score.

We use a 10% elitism [5]. The other 90% of parent slogans are selected using a roulette wheel [6].

A new generation is built by pairing parents and performing the crossover function followed by the mutation function, which occur with probabilities $p_{crossover}$ and $p_{mutation}$ respectively. Offspring are then evaluated and compared to the parents, in order to remove very similar ones. If the number of the remaining slogans is smaller than the size of the population, some additional random slogans are generated using the method for initial slogans production. After that, slogans proceed into the next generation. These steps are repeated until the predefined maximal number of iterations is achieved.

Crossover. There are two types of crossover function, the *big* and the *small* one. Both inspect POS tags of the words in both parents, and build a set of possible crossover locations. Each element in the set is a pair of numbers. The first one provides a position of crossover in the first parent and the second one in the second parent. The corresponding words must have the same POS tag. Let the chosen random pair from the set be (p, r). Using the *big* crossover, the part of the first parent, from the p^{th} position forward, is switched with the part of the second parent, from the r^{th} position forward. For the *small* crossover only the p^{th} word in the first parent and the r^{th} word in the second parent are switched. Examples for the *big* and the *small* crossover are in Figure 3.


big:
We [PRP] bring [VBP] **good [JJ] things [NNS] to [DT] life [NN].**
Fly [VB] the [DT] **friendly [JJ] skies [NNS]**.

$\longrightarrow$ We bring friendly skies.
Fly the good things to life.

small:
Just [RB] **do [VB]** it [PRP].
Drink [VB]more [JJR] milk [NN].

$\longrightarrow$ Just drink it.
Do more milk.

Figure 3. Examples for the *big* and the *small* crossover.

Mutation. Two types of mutation are possible. Possible *big* mutations are: deletion of a random word; addition of an adjective in front of a noun word; addition of an adverb in front of a verb word; replacement of a random word with new random word with the same POS tag. *Small* mutations are replacements of a word with its synonym, antonym, meronym, holonym, hypernym or hyponym. A meronym is a word that denotes a constituent part or a member of something. The opposite of a meronym is a holonym - the name of the whole of which the meronym is a part. A hypernym is a general word that names a broad category that includes other words, and a hyponym is a subdivision of more general word.

Functions for obtaining such replacements are embedded into the Nodebox English Linguistics library and are based on the WordNet lexical database [9].

Deletion of similar slogans. Every generated slogan is compared to all its siblings and to all the evaluated slogans from the previous generation. If a child is identical to any other slogan, it gets removed. If more than half of child's words are in another slogan, the two slogans are considered similar. Their evaluation scores are being compared and the one with the higher score remains while the other one is removed. The child is also removed, if it contains only one word or if it is longer than 10 words. Deletion of similar slogans prevents the generated slogans to converge to the initial ones.

4. Experiments

We made a preliminary assessment of the generator with experiments as described in this section.

4.1 Experimental Setting

In presented experiments and results we use a case of the U. S. soft drinks manufacturer Coca-Cola. The input text was obtained from Wikipedia [15].

First, we tried to find the optimal weights for the evaluation function. We tested different combinations of weights on a set of manually evaluated slogans. The comparison of the computed and the manually assigned scores showed that the highest matching was achieved with the following weights: [bigram: 0.22, length: 0.03, diversity: 0.15, entity: 0.08, keywords: 0.12, frequent words: 0.1, polarity: 0.15, subjectivity: 0.05, semantic relatedness: 0.1].

In our experiments we used probabilities for crossover and mutation $p\_crossover = 0.8$, $p\_mutation = 0.7$. The probability for mutation was set very high, because it affects only one word in a slogan. Consequently the mutated slogan is still very similar to the original one. Thus the high mutation probability does not prevent population to converge to an optimum solution. For the algorithm to decide which type of crossover to perform, we set probabilities for the *big*, the *small* and *both* crossovers to 0.4, 0.2 and 0.4 respectively. The mutation type is chosen similarly. Probabilities of the *big* and the *small* mutation were set to 0.8 and 0.2. These control parameters were set according to the results of testing on a given input text, as their combination empirically leads to convergence.

Due to the high computational complexity of our method, the maximal number of iterations was set to 150. We performed 3 experiments and for each of them we executed 20 runs of the algorithm using the same input parameter values. The only difference between these three tests was in the size of the population - 25, 50 and 75.

4.2 Results and Discussion

All 20 runs of the algorithm on the same input data had similar statistical results. Statistics of average slogans' scores for each of the experiments are gathered in Table 1. Slogans' scores increased with each iteration. The results in Figure 4 show that the slogans' scores increased very fast in relation to the number of evaluations when the size of the population was set to 25. They increased a bit slower when the size of the population was set to 50 and 75.

Table 1. Comparison of average slogans' scores for sizes of population: 25, 50 and 75. (F = final slogans, IP = initial population).

	Minimum	*Maximum*	*Average*	*Median*	*Standard Deviation*
IP (25)	0.000	0.720	0.335	0.442	0.271
IP (50)	0.000	0.721	0.318	0.377	0.270
IP (75)	0.000	0.736	0.311	0.412	0.270
F (25)	0.542	0.874	0.736	0.754	0.089
F (50)	0.524	0.901	0.768	0.775	0.082
F (75)	0.497	0.920	0.778	0.791	0.086

The numbers in graph show that our method ensures higher slogan scores with each new iteration of genetic algorithm, for a given experimental cases. Examples of slogans for one specific run of the algorithm are listed in the following two lists. The first list contains 10 best rated initial slogans and the second one contains 10 best rated final slogans for the case when the size of the population was set to 25. Evaluation scores are in the brackets.

Initial population:

1 The lucky player to sign the language in (0.714)

2 it should enjoy lead without learning line (0.706)

3 collection remains available more. fit more (0.647)

4 growing child have (0.600)

5 not a speed in a generator (0.595)

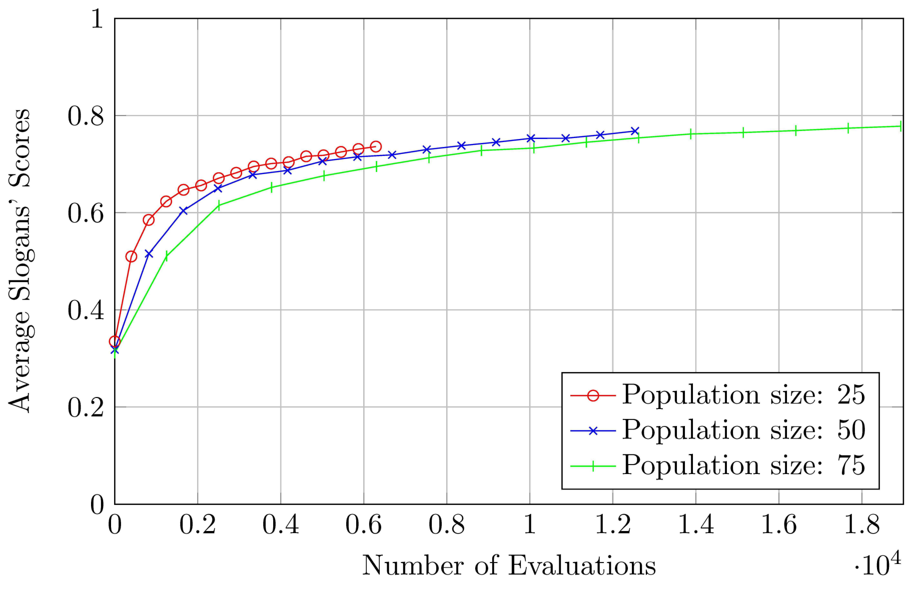


Figure 4. Average slogans' scores in relation to the number of evaluations.

6 Where the dream lives the environment (0.594)

7 Coke on the legal test (0.592)

8 called skip (0.560)

9 add to Coke Capazoo (0.559)

10 also stories that provide alternatives might read due as our community (0.527)

Final slogans:

1 love to take The Coke size (0.906)

2 rampage what we can take more (0.876)

3 love the man binds the planetary Coke (0.870)

4 devour what we will take later (0.859)

5 you can put The original Coke (0.850)

6 lease to take some original nose candy (0.848)

7 contract to feast one's eyes the na keep (0.843)

8 it ca taste some Coke in August (0.841)

9 hoy despite every available larger be farther (0.834)

10 you Can love the simple Coke (0.828)

The analysis of initial populations and final slogans in all runs of experiments shows that the majority of slogans are semantically incoherent and have grammatical errors.

Our system currently lacks an evaluation function for detection or correction of these mistakes.

Some seemingly good slogans can be found already in the initial populations. The evaluation function seems not yet aligned well with human evaluation, as such slogans often do not make it to the final round.

5. Conclusions

The proposed slogan generation method works and could be potentially useful for brainstorming. The genetic algorithm ensures that new generations of slogan candidates have higher evaluation scores. The critical part of the method is the evaluation function, which is inherently hard to formalize and needs further improvement. The definitions of evaluation sub-functions are currently too simplified. We believe that the refinement of semantic and sentiment evaluation functions would increase the quality of slogans, not only their scores.

The current algorithm is suitable only for production of slogans in English, because there is a wide range of lexical and semantic resources for it. There is a possibility of generating slogans in a language with different characteristics, for instance Slovenian. However, the lack of resources and different language properties would require a lot of work in order to adapt our algorithm for a non-English language.

There are also many other ideas for the future work that would improve the quality of slogans. One is checking for grammatical errors and correcting them if possible. New weights for the evaluation could be computed periodically with semi-supervised learning on manually assessed slogans. Also, control parameters for GA could be adaptively calculated during the optimization process [14].

Acknowledgment

This research was partly funded by the European Union, European Social Found, in the framework of the Operational Programme for Human Resources Development, by the Slovene Research Agency and supported through EC funding for the project ConCreTe (grant number 611733) and project WHIM (grant number 611560) that acknowledge the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission.

References

[1] T. Bäck. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms.* Oxford University Press, 1996.

[2] S. Bird, E. Klein, and E. Loper. *Natural language processing with Python.* O'Reilly Media, 2009. `http://www.nltk.org/`

[3] M. Davies. N-grams data from the Corpus of Contemporary American English (COCA). Downloaded from `http://www.ngrams.info` on April 15, 2014.

[4] T. De Smedt and W. Daelemans. Pattern for Python. *J. Mach. Learn. Res.*, 13:2063–2067, 2012.

[5] D. Dumitrescu, B. Lazzerini, L. C. Jain, and A. Dumitrescu. *Evolutionary Computation.* CRC Press, 2000.

[6] J. H. Holland. *Adaption in Natural and Artificial Systems.* MIT Press, 1992.

[7] R. Manurung, G. Ritchie, and H. Thompson. Using genetic algorithms to create meaningful poetic text. *J. Exp. Theor. Artif. In.*, 24:43–64, 2012.

[8] M. Marneffe, B. MacCartney, and C. Manning. Generating typed dependency parses from phrase structure parses. In *Proc. 5th International Conference on Language Resources and Evaluation (LREC)*, pages 449–454, 2006.

[9] G. A. Miller. WordNet: A Lexical Database for English. *Comm. ACM*, 38:39–41, 1995.

[10] C. S. Montero and K. Araki. Is it correct?: towards web-based evaluation of automatic natural language phrase generation. In *Proc. Joint Conference of the International Committee on Computational Linguistics and the Association for Computational Linguistics (COLING/ACL)*, pages 5–8, 2006.

[11] R. G. Morris and S. H. Burton. Soup over bean of pure joy: Culinary ruminations of an artifcial chef. In *Proc. 1st IEEE International Conference on Communication in China (ICCC)*, pages 119–125, 2012.

[12] Nodebox. `http://nodebox.net/code/index.php/Linguistics`

[13] G. Özbal, D. Pighin, and C. Strapparava. BRAINSUP: Brainstorming Support for Creative Sentence Generation. In *Proc. 51st Annual Meeting of the Association for Computational Linguistics*, pages 1446–1455, 2013.

[14] G. Papa. Parameter-less algorithm for evolutionary-based optimization. *Comput. Optim. Appl.*, 56:209–229, 2013.

[15] Wikipedia. `http://en.wikipedia.org/wiki/Coca-Cola` on April 29, 2014.