Using a Genetic Algorithm to Produce Slogans

Polona Tomašič
XLAB d. o. o., Pot za Brdom 100, SI-1000 Ljubljana, Slovenia and
Jožef Stefan International Postgraduate School, Jamova cesta 39, SI-1000 Ljubljana, Slovenia
E-mail: polona.tomasic@xlab.si

Gregor Papa
Computer Systems Department, Jožef Stefan Institute, Jamova cesta 39, SI-1000 Ljubljana, Slovenia and
Jožef Stefan International Postgraduate School, Jamova cesta 39, SI-1000 Ljubljana, Slovenia
E-mail: gregor.papa@ijs.si

Martin Žnidaršič
Department of Knowledge Technologies, Jožef Stefan Institute, Jamova cesta 39, SI-1000 Ljubljana, Slovenia and
Jožef Stefan International Postgraduate School, Jamova cesta 39, SI-1000 Ljubljana, Slovenia
E-mail: martin.znidarsic@ijs.si

Keywords: genetic algorithm, slogan generation, computational creativity, linguistic resources

Received: December 1, 2014

Creative tasks, such as creation of slogans for companies, products or similar entities, can be viewed from the combinatorial perspective – as a search through the space of possible combinations. To solve such a combinatorial optimization problem, we can use evolutionary algorithms. In this paper, we present our solution for generation of slogans based on a genetic algorithm and linguistic resources. We also compare it to the unguided slogan generator.

Povzetek: Na kreativne naloge, kot je snovanje sloganov za podjetja in produkte, lahko gledamo s kombinatoričnega vidika – kot na iskanje v prostoru možnih kombinacij. Za reševanje tovrstnih kombinatoričnih optimizacijskih problemov lahko uporabljamo evolucijske algoritme. V tem članku predstavljamo rešitev za generiranje sloganov na podlagi genetskega algoritma in jezikovnih virov. Predstavljeno rešitev primerjamo tudi z generatorjem sloganov brez vodenja.

1 Introduction

Automated generation of slogans is a problem from the field of Computational Creativity [5]. There are very few studies dedicated to slogan generation. In fact, the only one we came across is the BRAINSUP framework [19], which is based on beam search through a carefully defined space of possible slogans. This space gets reduced by applying user specified constraints on keywords, domain, emotions, and other properties of slogans.

High quality slogans are often a result of group brainstorming. Several individuals present their ideas and the proposed slogans are then mixed into new slogans, and some new ideas emerge. This brainstorming process is similar to the evolution, from which we got the idea of using evolutionary algorithms for slogan generation. The initial slogans from brainstorming represent an initial population, mixing the best proposed slogans represents recombination, and new included ideas represent mutations. Evolutionary algorithms have already been applied to different natural language processing problems [2].

In this paper, we present our slogan generation procedure which is not influenced by the user in any way, apart from being provided with a short textual description of the target entity. The method is based on a genetic algorithm (GA) [3]. Genetic algorithms are the most traditional evolutionary algorithms and they ensure a good coverage of the search space. They have been successfully used for generating recipes [17], poetry [13] and trivial dialog phrases [16]. However, genetic algorithms have not been previously used for slogan generation. Our method follows the BRAINSUP framework in the initial population generation phase, and it uses a collection of heuristic slogan functions in the evaluation phase.

We tested our slogan generator and compared it to the random slogan generator. The statistical results are in favor of our method. However, even though the generated slogans can present a good starting point for brainstorming, their quality is not yet at the desired level.

The rest of the paper is organized as follows. In Section 2 we present the linguistic and semantic resources used in our solution. Section 3 provides a detailed description of the entire slogan generation process. It includes description of the evaluation functions and it clarifies the differ-

ence between the slogan generator and the unguided slogan generator. The performed experiments and the discussion of the results are presented in Section 4. The conclusions are drawn in Section 5.

2 Resources

Linguistic and semantic resources are a prerequisite for any kind of text generation. We use them at several steps of our method – for generation of initial population, mutation, and evaluation. Some are available as extended libraries for programming languages, others are available for download from the Internet, and some databases were created by ourselves. The origin of the data and the process is briefly described in the following paragraphs.

1. **Database of famous slogans:** it serves as a basis for the initial population generation and for comparison with generated slogans. It contains 5,249 famous slogans obtained from the Internet.

2. **Database of frequent grammatical relations between words in sentences:** for its acquisition we used the Stanford Dependencies Parser [14]. Stanford dependencies are triplets containing two words and the name of the relation between them. The parser also provides part-of-speech (POS) tags and phrase structure trees. 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.

3. **Database of slogan skeletons:** slogan skeletons were obtained by parsing famous slogans with the Stanford Dependencies Parser. A slogan skeleton contains information about each position in the sentence – its POS tag and all its dependence relations with other words in the sentence. It does not contain any content words, only stop words. An example of a skeleton is shown in Figure 1.

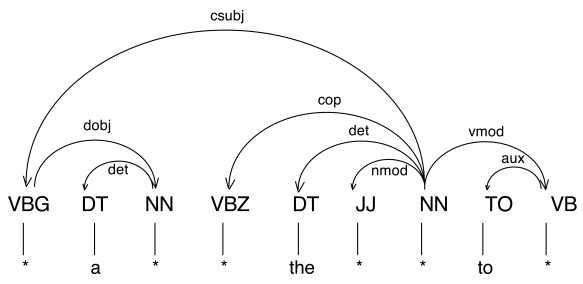

Figure 1: Example of a skeleton.

3 Slogan Generation

An input of our slogan generator is a short textual description about the target entity. It is the only required input from a user. It is used to obtain the name of the target entity and a set of keywords. An output is a list of generated slogans. The whole procedure is shown in Algorithm 1.

3.1 Extraction of the Keywords and the Main Entity

The most frequent non-negative words from the input text are selected as keywords. Negative words are detected using the Nodebox English Linguistics library [18]. The main entity is usually the name of the company and is obtained by selecting the most frequent entity in the whole text using the *nltk* library [4].

3.2 Generation of the Initial Population of Slogans

The procedure of generating the initial population of slogans is based on the BRAINSUP framework [19], with some modifications. It follows the steps in Algorithm 2. Skeletons are obtained from the database of slogan skeletons. Fillers are the words from the database of all grammatical relations between words in sentences that 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 the selection of filler words. We don't consider any user specified constraints, while the BRAINSUP framework uses 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. Thus using our method we can produce many different slogans from the same slogan skeleton, whereas BRAINSUP produces only one for given user specifications.

3.3 Evaluation of Slogans

An aggregated evaluation function is used to evaluate the slogans. 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.

3.3.1 Bigram Function

In order to work with 2-grams, we obtained the dataset of 1,000,000 most frequent 2-grams and 5,000 most frequent words in Corpus of Contemporary American English (COCA) [6]. We assume that slogans containing many frequent 2-grams, are more likely to be semantically coherent.

Algorithm 1: SloganGenerator

Input: A textual description of a company or a product T, Size of the population S_{P}, Maximum number of iterations $MaxIterations$, Crossover probability $p_{\mathrm{crossover}}$, Mutation probability p_{mutation}, Set of evaluation weights W.
Output: A set of generated slogans *S*.

$Keywords, Entity \Leftarrow$ GetKeywordsAndEntity(T)
$P \Leftarrow$ CreateInitialPopulation($S_{\mathrm{P}}, Keywords, Entity$)
Evaluate(P)
$Iteration \Leftarrow 0$
while $Iteration < MaxIterations$ **do**
  $Parents \Leftarrow$ ChooseParentsForReproduction(P)
  $Children \Leftarrow$ Crossover($Parents, p_{\mathrm{crossover}}$)
  $Children \Leftarrow$ Mutation($Children, p_{\mathrm{mutation}}$)
  $NewGeneration \Leftarrow$ DeleteSimilarSlogans($P, Children$)
  while Size($NewGeneration$) $< S_{\mathrm{P}}$ **do**
    AddRandomlyGeneratedSlogan($NewGeneration$)
  end while
  Evaluate($NewGeneration$)
  $P \Leftarrow S_{\mathrm{P}}$BestSlogans($NewGeneration$)
  $Iteration \Leftarrow Iteration + 1$
end while
$S \Leftarrow P$

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*.

$S \Leftarrow \emptyset$
while $S_{\mathrm{P}} > 0$ **do**
  $SloganSkeleton \Leftarrow$ SelectRandomSloganSkeleton()
  while not AllEmptySlotsFilled($SloganSkeleton$) **do**
    $EmptySlot \Leftarrow$ SelectEmptySlotInSkeleton($SloganSkeleton$)
    $Fillers \Leftarrow$ FindPossibleFillerWords($EmptySlot$)
    $FillerWord \Leftarrow$ SelectRandomFillerWord($Fillers$)
    FillEmptySlot($SloganSkeleton, FillerWord$)
  end while
  AddFilledSkeleton($S, SloganSkeleton$)
  $S_{\mathrm{P}} \Leftarrow S_{\mathrm{P}} - 1$
end while

3.3.2 Length Function

The length function is very strict, it assigns score 1 to slogans with less than eight words, and score 0 to longer ones. The threshold between 0 and 1 was set according to the results of the experiments, which showed that a large majority of the generated slogans that contained more than seven words were grammatically incorrect and semantically incoherent. Also, more than 90% of the famous slogans are less than eight words long. This function acts as an absolute constraint and that is why no values between 0 and 1 are allowed.

3.3.3 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.

3.3.4 Entity Function

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

3.3.5 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.

3.3.6 Word Frequency Function

This function prefers slogans with many frequent words. A word is considered to be frequent, if it is among 5,000 most frequent words in COCA. The frequency score is obtained by dividing the number of frequent words by the number of all words in the slogan.

3.3.7 Polarity and Subjectivity Functions

Polarity of a slogan indicates whether slogan contains positive or negative words. For instance, the adjective “happy" is a positive word. In a similar way subjectivity of a slogan indicates whether slogan contains words that express the attitude of the author. For instance, the adjectives “good" and “bad" both represent the opinion of the author and are therefore subjective. The polarity and subjectivity scores are calculated based on the adjectives in the slogan, using the *sentiment* function from *pattern* package for Python [7].

3.3.8 Semantic Relatedness Function

This function computes the relatedness between all pairs of content words in a 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 [15]. 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 10% elitism [8]. The other 90% of parent slogans are selected using a roulette wheel [11].

A new generation is built by pairing parents and performing the crossover function followed by the mutation function, which occur with probabilities $p_{\text{crossover}}$ 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 creation of initial population. After that, slogans proceed into the next generation. These steps are repeated until the predefined number of iterations is achieved.

3.4.1 Crossover

We use two types of crossover functions, 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 illustrated in Figure 2.

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

→ We bring friendly skies.
Fly the good things to life.

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

→ Just drink it.
Do more milk.

Figure 2: Examples for the *big* and the *small* crossover.[1]

3.4.2 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.

3.4.3 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 higher score remains in the population 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. This has been checked by testing our method without the deletion of similar slogans phase.

[1] Slogans used in the examples were or still are official slogans of the following companies: General Electric, United Airlines, Nike, and BC Dairy Association.

3.5 Correction of Grammatical Errors

Crossover and mutation functions may cause grammatical errors in generated slogans. For instance, incorrect usage of determiners (e.g., “a apple" instead of “an apple"), sequence of incompatible words (e.g., “a the"), and others. Spelling mistakes were much less frequent.

In order to remove both types of errors in the final slogans, we tested different spell- and grammar checkers. One example of a spell-checker is Hunspell [12]. Its downside is that it works on one word at a time and does not take the word's context into account. As the majority of errors in slogans originated from grammar, we tested several grammar checkers. They, on the other hand, work on the sentence level rather than on the word level. Most of these grammar checkers are available as online services, and don't support API calls. One that does is python-ginger [10] – a Python package for fixing grammar mistakes. It comes with an unofficial Ginger [9] API. This tool corrects different types of grammatical mistakes. It is also used for contextual spelling correction. We used python-ginger only on final slogans, the ones that are displayed to the user, because the corrected slogan may not have the same structure anymore. Possible added words, or replacing a word with another one with different POS tag would cause errors while executing crossover, mutation and evaluation functions.

3.6 Unguided Slogan Generator

For the purpose of evaluation of our slogan generation method, we also implemented an unguided slogan generator (USG). This generator produces random slogans, such as the ones in the initial population. The only difference between our method and the unguided slogan generation method is in the production of a new generation. USG has no crossover and the mutation steps. Instead it produces a new generation using a method for creation of initial population. Thus children are independent of the previous generation. The algorithmic steps are shown in Algorithm 3.

4 Experiments

We tested the slogan generation method on different input texts and for different values of algorithm parameters. We analyzed the results of every iteration of the genetic algorithm to see how the slogans' scores changed and made further assessment of the generator by comparing its results with the results of the unguided slogan generator.

4.1 Experimental Setting

4.1.1 Input Text

In the presented experiments, we use a case of the Croatian provider of marine solutions, Sentinel. Sentinel is a control module that provides more security to boat owners, and is comprised of a network of sensors and a central information hub. It ensures the vessel is monitored at all times. The input text was obtained from the Sentinel's web-page [21].

4.1.2 Algorithm Parameters

Different combinations of weights of the evaluation function were tested on a set of manually evaluated slogans. We added one constraint – the weight of the keywords function had to be at least 0.2 in order to include keywords in the slogans. Without this constraint the computed weight for the keywords was almost zero. The comparison of the computed and the manually assigned scores showed that the highest matching was achieved with the following weights: [bigram: 0.25, length: 0.01, diversity: 0.01, entity: 0.1, keywords: 0.2, frequent words: 0.25, polarity: 0.01, subjectivity: 0.02, semantic relatedness: 0.15].

Probabilities for crossover and mutation were set to $p_{\text{crossover}} = 0.8$ and $p_{\text{mutation}} = 0.7$. 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 from converging to the 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 algorithm parameters were set according to the results of testing on a given input text, as their combination empirically leads to convergence.

We performed three experiments and for each of them we executed 20 runs of the algorithm using the same input parameter values. The difference between these three tests was in the size of the population (S_{P}) and the number of iterations (N_{It}). Those were chosen according to the desired number of all evaluations ($\approx 6,800$ NoE), and the NoE was set according to the desired execution time for one run of the algorithm – approximately 2 hours.

1. S_{P}: 25, N_{It}: 360
2. S_{P}: 50, N_{It}: 180
3. S_{P}: 75, N_{It}: 120

4.1.3 Comparison with the Unguided Slogan Generator

For comparison, we performed three experiments with the unguided slogan generator. For each of them we executed 20 runs of the algorithm using the same input parameter values as in the experiments with slogan generator. The initial populations were also identical. The number of iterations were again chosen so as to match the number of all evaluations ($\approx 6,800$ NoE) in the experiments with slogan generator:

1. S_{P}: 25, N_{It}: 300
2. S_{P}: 50, N_{It}: 150
3. S_{P}: 75, N_{It}: 100

Algorithm 3: UnguidedSloganGenerator

Input: A textual description of a company or a product T, Size of the population S_{P}, Maximum number of iterations $MaxIterations$, Set of evaluation weights W.
Output: A set of generated slogans S.

$Keywords, Entity \Leftarrow$ GetKeywordsAndEntity(T)
$P \Leftarrow$ CreateInitialPopulation($S_{\mathrm{P}}, Keywords, Entity$)
Evaluate(P)
$Iteration \Leftarrow 0$
while $Iteration < MaxIterations$ **do**
  $Children \Leftarrow$ CreateInitialPopulation($S_{\mathrm{P}}, Keywords, Entity$)
  $NewGeneration \Leftarrow$ DeleteSimilarSlogans($P, Children$)
  while Size($NewGeneration$) $< S_{\mathrm{P}}$ **do**
    AddRandomlyGeneratedSlogan($NewGeneration$)
  end while
  Evaluate($NewGeneration$)
  $P \Leftarrow S_{\mathrm{P}}$BestSlogans($NewGeneration$)
  $Iteration \Leftarrow Iteration + 1$
end while
$S \Leftarrow P$

In USG, children in new generations are frequently identical to parents, and therefore need no evaluation (we already have the scores of the parents). We wanted to compare the two generators based on the number of evaluations, not the number of iterations. For our slogan generator to reach the same number of evaluations as the unguided slogan generator, it needs to perform more iterations of genetic algorithm. That is why the numbers of iterations in SG and USG differ.

4.2 Results and Discussion

Comparing the statistical results of the initial and final populations of slogans, there were no major differences between the 20 runs of the algorithm on the same input data for all 6 experiments. The number of evaluations for each run is approximately $6,800$.

Statistics of average initial slogans' scores are in Table 1. The numbers are the same for both generators. Average final slogans' scores are in Table 2. The average minimum score is much higher using the unguided slogan generator (USG). This is because in our slogan generator (SG) many slogans get deleted in the deletion phase of the algorithm. Consequently some new random slogans are automatically included in a new generation, and they can have very low evaluation scores. However, SG has higher maximum slogan scores. This suggests that the usage of crossover and mutation functions actually increases the slogan scores. The average score of the 10 best slogans is higher using the SG.

Numbers in both tables show that average slogans' scores increased a lot from the initial population to the final one. Figures 3 and 4 show the relation between average slogan scores and the number of performed evaluations in a genetic algorithm using SG and USG. Using the USG causes the scores to increase immensely already in the first few iterations of the genetic algorithm. After that, they do not increase much anymore. In SG slogans' scores increase a little bit slower, but at some point they exceed the USG scores.

From the two graphs in Figures 3 and 4 one might conclude that the unguided slogan generator is at least as good as our developed slogan generation method. However, the numbers are calculated on slogans from a whole generation. In practice we don't expect the user to go through all 75 final slogans, but only a few. Thus only the best few slogans from the final list are important. Table 3 shows the average scores for the 10 best final slogans. In this case the slogan generator outperforms the unguided slogan generator.

In the following two lists, there are examples of slogans for one specific run of the algorithm. 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 50. Evaluation scores are in the brackets. The final slogans list contains the corrected versions of slogans using the Ginger API.

Initial Population:

1. Former offices for all its members houses. (0.692)
2. The lowest water to play Sentinel build. (0.664)
3. Land routes to better places. (0.663)
4. The Sentinel performance is topic. (0.662)
5. On day to perform. (0.642)
6. The side take in region. (0.639)
7. Even now right as not. (0.638)
8. A precise application consists with a pair. (0.632)
9. Draft the choice of allowing. (0.629)
10. The initiative in pursuing systems and weapons. (0.623)

Table 1: Comparison of average initial slogans' scores for population sizes 25, 50 and 75.

Size of the population	*Minimum*	*Maximum*	*Average*	*Median*	*Standard deviation*
25	0.000	0.713	0.287	0.359	0.257
50	0.000	0.740	0.289	0.302	0.257
75	0.000	0.730	0.274	0.295	0.251

Table 2: Comparison of average final slogans' scores using our slogan generator (SG) and the unguided slogan generator (USG) for population sizes 25, 50 and 75

Size of the population	*Minimum*	*Maximum*	*Average*	*Median*	*Standard deviation*
25 (SG)	0.578	0.906	0.801	0.823	0.088
50 (SG)	0.511	0.927	0.793	0.807	0.090
75 (SG)	0.488	0.939	0.773	0.791	0.094
25 (USG)	0.763	0.840	0.795	0.796	0.021
50 (USG)	0.723	0.837	0.767	0.761	0.032
75 (USG)	0.707	0.840	0.750	0.743	0.036

Table 3: Comparison of average scores of 10 best final slogans, using our slogan generator (SG) and the unguided slogan generator (USG) for population sizes 25, 50 and 75

Size of the population	*Minimum*	*Maximum*	*Average*	*Median*	*Standard deviation*
25 (SG)	0.833	0.906	0.869	0.870	0.023
50 (SG)	0.877	0.927	0.895	0.892	0.019
75 (SG)	0.871	0.939	0.902	0.902	0.023
25 (USG)	0.799	0.840	0.816	0.813	0.013
50 (USG)	0.804	0.837	0.819	0.813	0.011
75 (SG)	0.801	0.840	0.818	0.814	0.013

Final Slogans:

1. Enjoy the part like water for sentinel. (0.958)
2. Enjoy a take of routine on sentinel. → Enjoy a track of routine on sentinel. (0.958)
3. Make all safety in safe for sentinel. → Make all safety in safe for a sentinel. (0.958)
4. Demand and enjoy the use in sentinel. → Demand and enjoy the ease in sentinelthe sentinel. (0.958)
5. Write a base for demand on sentinel. (0.948)
6. Demand the of potential as sentinel. (0.945)
7. Enjoy a sentinel performance show. (0.922)
8. Themes for head on sentinel. (0.913)
9. Contents with application on sentinel. → Contents with application of sentinel. (0.913)
10. Make the sentinel performance plays. (0.897)

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. However, slogans produced with the unguided slogan generator seemed more structured and semantically coherent. This is understandable, since the crossover and mutation functions in our slogan generator affect the sentence structure a lot. The percentage of corrected final slogans is also in favor of the unguided slogan generator: 24.6% of final slogans produced with USG got corrected with the Ginger API, while the percentage of corrected final slogans for SG is 33.9%. But we need to take into account the fact that Ginger API does not work without mistakes. Some of the corrections are strange or unnecessary (e.g., see example 4 in the final slogans list).

5 Conclusion

The proposed slogan generation method works and could be potentially useful for brainstorming. It produces slogans solely from the textual description of the target entity. No other user specifications are needed. The genetic algorithm ensures higher slogan scores with each new iteration. Our method outperforms the unguided slogan generator whose best 10 final slogans have significantly lower average scores. The unguided slogan generator also needs more than six times more time to produce and evaluate the same number of slogans as our slogan generator.

The evaluation function is inherently hard to formalize and seems not yet fully aligned with human evaluation. The definitions of evaluation sub-functions need further improvement in order to increase the quality of slogans, not only their scores.

The current algorithm is suitable only for production of slogans in English. The lack of resources and different language properties would require a lot of work in order to adapt our algorithm to another language.

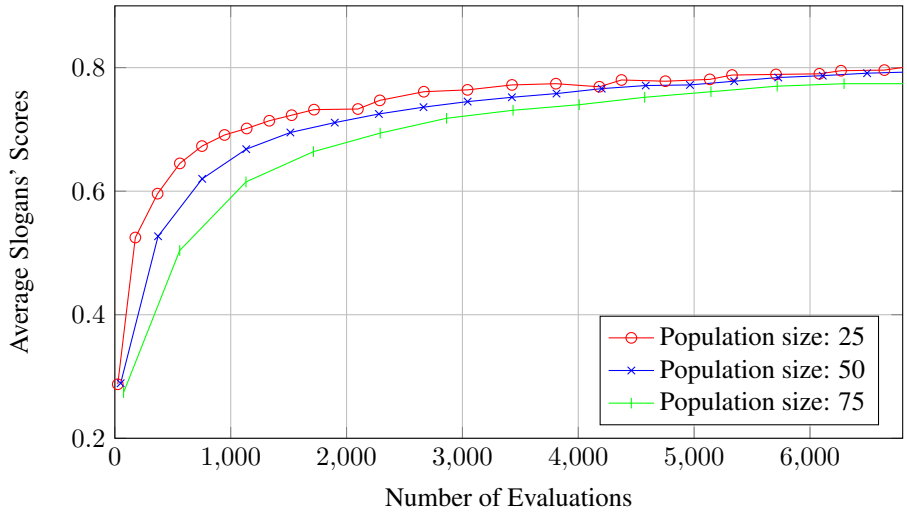

Figure 3: Slogan generator: average scores of slogans in a relation to the number of evaluations.

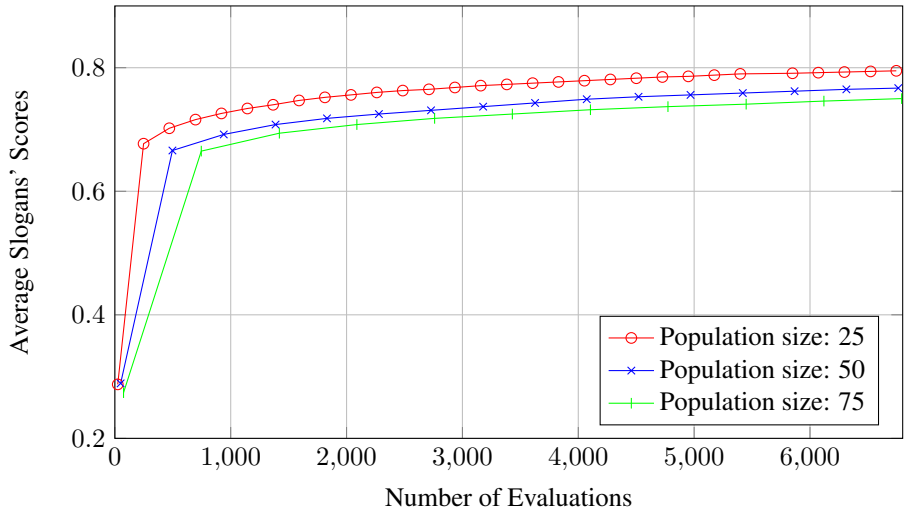

Figure 4: Unguided slogan generator: average scores of slogans in a relation to the number of evaluations.

Following are some ideas for the future work that would improve the quality of slogans. One is detecting and correcting grammatical errors already during the generation phase. New weights for the evaluation could be computed periodically with semi-supervised learning on manually assessed slogans. The parallelization of GA [1] might provide gains in performance. Also, the GA parameters could be adaptively calculated during the optimization process [20].

Acknowledgement

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] E. Alba, J. M. Troya (1999) A survey of parallel distributed genetic algorithms, *Complexity*, vol. 4, pp. 31–52.

[2] L. Araujo (2009) How evolutionary algorithms are applied to statistical natural language processing, *Artificial Intelligence Review*, vol. 28, pp. 275–303.

[3] T. Bäck (1996) *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*, Oxford University Press.

[4] S. Bird, E. Klein, E. Loper (2009) *Natural language processing with Python*, O'Reilly Media.

[5] S. Colton, R. Mantaras, O. Stock (2009) Computational Creativity: Coming of age, *AI Magazine*, vol. 30, no. 3, pp. 11–14.

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

[7] T. De Smedt, W. Daelemans (2012) Pattern for Python, *Journal of Machine Learning Research*, vol. 13, pp. 2063–2067.

[8] D. Dumitrescu, B. Lazzerini, L. C. Jain, A. Dumitrescu (2000) *Evolutionary Computation*, CRC Press.

[9] Ginger, `www.gingersoftware.com/grammarcheck`, accessed on October 17, 2014.

[10] Ginger API, `github.com/zoncoen/python-ginger`, accessed on October 17, 2014.

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

[12] Hunspell, `hunspell.sourceforge.net`, accessed on October 20, 2014.

[13] R. Manurung, G. Ritchie, H. Thompson (2012) Using genetic algorithms to create meaningful poetic text, *Journal of Experimental & Theoretical Artifcial Intelligence*, vol. 24, pp. 43–64.

[14] M. Marneffe, B. MacCartney, C. Manning (2006) Generating typed dependency parses from phrase structure parses, *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC)*, pp. 449–454.

[15] G. A. Miller (1995) WordNet: A Lexical Database for English, *Communications of the ACM*, vol. 38, pp. 39–41.

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

[17] R. G. Morris, S. H. Burton (2012) Soup over bean of pure joy: Culinary ruminations of an artifcial chef, *Proceedings of the International Conference on Computational Creativity (ICCC)*, pp. 119–125.

[18] NodeBox, `nodebox.net/code/index.php/Linguistics`, accessed on October 17, 2014.

[19] G. Özbal, D. Pighin, C. Strapparava (2013) BRAINSUP: Brainstorming Support for Creative Sentence Generation, *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pp. 1446–1455.

[20] G. Papa (2013) Parameter-less algorithm for evolutionary-based optimization, *Computational Optimization and Applications*, vol. 56, pp. 209–229.

[21] Sentinel, `sentinel.hr`, accessed on October 10, 2014.