

TJHSST Computer Systems Lab Senior  
Research Project  
Word Play Generation  
2009-2010

Vivaek Shivakumar

June 16, 2010

**Abstract**

Computational humor is a subfield of artificial intelligence focusing on computer recognition and generation of humorous language. This paper investigates methods for generating various types of word play (e.g., puns, palindromes, acronyms) using a lexicon from the Natural Language Toolkit and the semantic web WordNet as well as phonetic information, which is the basis for manypuns. Although no formal model or theory for humor exists, pun-generation has been implemented using simple, constrained models and this project attempts to recreate such implementations with possible expansions to more types of word play.

**Keywords:** computational humor, pun, word game

## 1 Introduction

One of the main goals of artificial intelligence is natural language generation, meaning that computers should eventually be able to generate meaningful text that would require human intelligence and cognition to produce. One aspect of human language that is of interest to AI is humor. Not only is humor essential to the goals of computer creativity, but humor generation

has applications for language development and teaching, and is essential for the future of human-computer interaction [1].

The purpose of this project is to investigate and implement methods for generating humor, specifically various types of short puns and word play. Puns include spoonerisms, humorous acronyms, and a variety of other forms. However, there is no strict definition for a pun [2]. Furthermore, puns and humor in general have not been studied to the point of developing a formal model of humor that could be used in AI applicaitons [3]. Nevertheless, work has been done in the area of pun generation since 1994 [4] culminating in advanced user-interface programs such as STANDUP [5].

## 2 Background

Humor has been studied extensively in a social or otherwise non-AI context. Although much literature on humor and human cognition with respect to humor has indicated patterns and similarities yielding some informal accounts of humor such as the incongruity theory, no formal models or theories yet exist to facilitate the application of humor to AI [3]. Nevertheless, the possibility of computer generation or recognition of humor is by no means impossible. Several computer programs have been created to generate small puns, including JAPE, which was based on certain classes of puns modeled by *schema* and *templates* and utilizing the large lexicon WordNet. Recently, a successful effort called STANDUP was put through to improve JAPE and create a full interactive module for generating puns geared towards “children with complex communication needs”. [5]

Other applications of computational humor that have been implemented include a humorous acronym-builder, a “What do you get when you cross” generator, and several joke-recognition applications. [5][6]

### 2.1 Word Play

Various types of word play exist, such as acronyms and backronyms, palindromes, anagrams, spoonerisms, and puns. Not much research exists on the generation of sophisticated and new instances of such word games, other than puns. However, one project (Stock and Strapparava) [7] created HA-HAcronym, a program to reanalyze existing acronyms by substituting words

to result in humorous interpretations, e.g. FBI = Fantastic Bureau of Intimidation. Some examples of other types of word games:

- Palindrome: A man, a plan, a canal- Panama!
- Anagram: “Eleven plus two” = “Twelve plus one”

## 2.2 Puns

Most popular types of puns that could potentially be implemented for generation, e.g. question-answer riddle puns, have some element of combining seemingly unrelated or random elements in a way that plays not necessarily on the semantics of the words themselves but on the phonetics. Some techniques employed by pronunciation-based puns include rhyming, homonyms, spoonerisms (trading initial sounds in sets of words) and syllable/word substitution based on phonetic similarity. Some examples:

- What do you get when you cross a murderer with a breakfast food? A cereal killer.
- What is the difference between leaves and a car? One you brush and rake, the other you rush and brake.
- Pasteurise: too fast to see. (An example of redefinition wordplay) [8]

Other types of puns and jokes include knock-knock jokes and Tom Swifty puns [8]. Jokes such as “yo mama” jokes or “Chuck Norris facts” on the other hand are based on complex semantic and pragmatic specifications and relationships that are outside the scope of phonetic-pun-based computational humor.

## 3 Methodology

### 3.1 Punning Riddles

Punning riddles such as of the form “What do you get when you cross A and B? C” usually incorporate at least two elements in both the question and the answer, and the relationships between the elements are either semantic or phonetic. The program for this riddle uses the WordNet semantic relations

and the CMU pronunciation dictionary, both included in NLTK, to generate, given user input, a set of words or terms that exhibit such relationships. In particular, it takes the user input (say, A1) and finds semantic relations of it: synonyms, hypernyms, associated words, etc. For each relation (B1) a homophone or near-homophone (B2) is found, and then semantic relations of B2 are found (A2) to complete a set.

A good set of words generated should be able to be made into a punning riddle where the A's are the elements of the question and the B's are the elements of the answer, combined in some form. This combination and application to a template is to be implemented. *Note:* a near-homophone is a word that is a limited number of phonetic changes away from a word. Such words can be found either by iteration over a dictionary and using a minimum-edit distance algorithm or by recursive generation of possible similar words of the original and dictionary lookup.

### 3.2 “What do you get when you cross” generator

A programmer named Jess Johnson has one of the only available computational humor projects available, on his website [6]. His program, written in Lisp, uses a user-written pre-prepared database of semantic and homophone relations and a specific set of rules and methods to determine the precise linguistic form for riddle generation. I have translated the entirety of that code into the Python language, accomplishing the same results. However, the methods used in that program reflect the same schematic method used in [4] and other research projects in punning riddle generation. An example of such a schema can be seen in Figure 1 After a schema is planned, it is implemented in a program that finds a corresponding combination of words, possibly by receiving input for one word to start. The steps after that include applying a proper template and correcting for surface features that may need adjusting due to linguistic structures, e.g. the presence of indefinite articles. Johnson's program provided for such template requirements. However, the source [6] provided several possible improvements, e.g. “More complete phonetic information,” and “More complete vocabulary. The vocabulary is somewhat contrived.” Using the same resources as for my original punning riddle program provides these improvements, and along with this existing schema-template implementation, should result in a functional punning riddle generator similar to projects like JAPE and STANDUP described earlier.

### 3.3 Palindromes

To generate any palindrome of characters, the method is simple: pick any string and append itself reversed. However, the goal of a useful palindrome generator is to generate those that make sense in English. The first step to that goal is to be able to generate palindromes made up entirely of valid words. The main parts of the method to do this are a stack holding the current state of the attempt and an algorithm to segment a string.

Random words are picked from a word list and added to the stack while the string joining all the words in the current stack is reversed and stored as the tail of the current state. After a word is added and the tail is created, the segmentation algorithm is attempted on an iteration over each incremental substring by letter since the last added word. The points of successful substring segmentation are kept in memory and used to determine when the stack has gotten too big that the tail cannot be segmented into possible English words, at which point the stack is popped and new words are tried. The algorithm is finished once the last successful tail substring segmentation coincides with a word boundary, meaning the stack+tail combination forms an English palindrome phrase.

### 3.4 Acronyms

To construct reanalyzed or new acronyms out of existing words, a given input of a word or phrase serves two purposes. First, the letters of which constitute it form the backbone of the acronym, so that the input is the acronym itself. Second, the input is the seed for all the words or phrases which will be possibilities to fill-in each letter slot in the acronym.

Those words can be of two sorts: semantically related words to the input such as synonyms, hypernyms, or related concepts, or associated words, i.e. those that describe it, are used frequently with it, or could otherwise be relevant. The former are easier to retrieve because lexica such as WordNet readily contain functions returning such words. The latter, however, are not readily available in any database. Therefore they are approximated by accessing data such as dictionary definitions or encyclopedia articles and empirically or heuristically determining which words are the most relevant. Using a list of common English words, irrelevant or unuseful words are removed to leave those which are probably associated with the input term. Once a list of all such words are collected, they are picked according to first

letter to fit as many slots in the input acronym as possible.

### **3.5 Anagrams**

There are several methods of producing anagrams of input text, such as by iterative “brute force” methods where all permutations of characters are produced and then split in various ways, picking those combinations that use only valid English words. Some slightly more efficient methods use recursion, removing letters from the text in question based on the current word picked, backtracking when remaining letters are insufficient for forming a word, and ending when all letters have been used. However, this program uses a much more efficient method of searching for an anagram (for the goal is indeed to find at least one instance) using a hill-climbing algorithm.

The method is as follows: the input text is processed to produce a frequency array of letters, to which the text is unique. Each word in the English word list (dictionary) is similarly processed. Starting with a random list of words, the sum of the corresponding arrays are compared in distance to the target (input) array and picking from all possible changes (adding, removing, or replacing a word), the program picks the greatest improvement. The algorithm uses random-restart in cases where it gets stuck. To more quickly catch problems where no anagram exists, a matrix row reduction of the dictionary’s arrays is used to tell immediately whether or not there is a solution.

### **3.6 Syntax**

To filter out instances of each type of word play output that are not sensical, a grammar filter must be implemented. Using general structures of phrases and sentences, regular expressions can be made to represent such syntactic requirements. Once words in a given text (i.e., the output of one of the word play programs) are tagged according to part of speech, etc., the text can be checked against the filter.

## 4 Results

### 4.1 Punning Riddles

Currently the program succeeds in, given a starting word, finding a set of four words or terms according to the schema given in Figure 1. More often than not, an input word will generate at least one set. However, several problems need to be addressed. The WordNet lexicon and the CMU pronouncing dictionary it uses employs both British and American English words and spellings, and for example in the case of homophones such distinctions can lead to false selections of variations of the same word. Proper names, which for the most part are not usable in the types of jokes analyzed here, are included in WordNet as well. A slew of uncommon nouns, not suitable for simple puns, are also present, giving rise to nonsensical or hard-to-understand combinations. Furthermore, the use of similarly pronounced words does not restrict results to homophones or even rhymes, but includes words which may not intuitively be considered as similar sounding to be used in a pun, e.g. “wild” and “world”. An improvement to the pronunciation similarity method could be to vary the strictness of similarity based on word length. Finally, sets of words do not include pairs where one may be substituted into the other to form a pun answer. As a result, the vast majority of generated sets currently are not feasible to be inserted into a schema to make a punning riddle, for example, “rabbit-ιconey-ιphony-ιdissimulator.” Nevertheless, some sets can conceivably be used, e.g. one result is “rabbit-ιhare-ιfair-ιhonest”, which, fitting into the schema, can be made into a riddle such as “What do you call an honest rabbit? A fair hare.”

The punning riddle program translated from LISP can reproduce the original’s results using a specified, hard-coded set of words and relations. For example, ”WHAT DO YOU GET WHEN YOU CROSS A COW WITH A LEMON? sour milk”

### 4.2 Palindromes

The output of the program is successful in that it can generate many palindromes composed of valid English words. Over time, there does not appear to be much of a slowing down due to lack of possibilities. However, a palindrome that makes either semantic or syntactic sense in English is rare among those generated, since the algorithm takes no such other factors into account other

than spelling. Nevertheless, the grammar filter comes into play and works to filter noun phrases with reasonable success. Examples below marked with an “X” were accepted by the filter, showing that it allowed those phrases which may make some sort of sense in English. A few example outputs of the program, both nonsensical and acceptable:

- X race car
- X level level
- aid i a
- on no
- fine too o o ten if
- once pattern ret ta pec no
- no ill im million
- X red art trader
- never even
- X test set
- oh whose hall action no it call a hes oh who

The apparent problem is the use of extremely obscure words as well as the over-use of very common words. Also a problem is the fact that words are not picked in any order to fit a syntactic structure, which leads to nonsense. However, in some examples such as “red art trader,” the use of exclusively nouns and adjectives (the vast majority of a lexicon anyway) does not prove problematic. Nevertheless, an improvement would be some sort of model or ruleset by which the program picks words other than at random in order to yield more successfully sensible palindromes, instead of just weeding them out upon completion.

### 4.3 Acronyms

The use of internet sources (primarily the OneLook dictionary website) to retrieve associated words to fill acronyms showed marked improvement (24.8). Some examples of the output:

- ORDER = Orderliness Rules Decree Edict Rescript
- BAD = Below Average Decency
- STUPID = Stunned — Unintelligent Person — Dolt
- GOD = Graven Omnipotent Deity
- BUSH = Born Under Sophistication Herbert
- CIA = Collecting Independent Activities
- CIA = Collecting Intelligence Abroad
- LAW = Legal Activity —
- WORD = Writings of Restricted Discussion

Although the success of output is largely subjective, there are several levels of evaluation. First, some tries leave blank spaces, and are immediately failures. Second, words such as “order” may fill all spaces with related words, but may not make sense otherwise. Some acronyms do get filled with phrases that make sense, e.g. WORD above, but the phrases may not make sense in the context of the word it forms (although they may, depending on the context in which the acronym might be used, such as the title of a project or organization). Finally, several input words do yield acronyms that make sense, such as BAD and CIA above.

### 4.4 Anagrams

The brute force method for generating anagrams of a given word or phrase is very inefficient compared to the hill-climbing algorithm, as can be seen by the runtimes in Figure 2. However, the brute force algorithm does list all possibilities of anagrams, including permutations. For example, the input “enter word” (also the prompt used in the program) yields: [’red’, ’went’,

'or'], ['red', 'or', 'went'], ['or', 'went', 'red'], ['or', 'red', 'went'], ['went', 'or', 'red'], ['went', 'red', 'or'], ['went', 'order'], ['order', 'went']. The hill-climbing algorithm quickly produces an anagram for an input (or otherwise says “no solution” in the case where it is not possible, such as for the input “Vivaek”). Examples are “multi vites” yielding “it must live” and “enter word” yielding “order went”.

## 5 Conclusion

For the types of word play discussed here, there are many problems yet to be overcome. There is a barrier of sophistication in terms of language knowledge which is still being explored in other areas of computational linguistics. Furthermore, resources were not sufficient for testing some of these problems, and more comprehensive word lists and information databases will lead to greater results. Nevertheless, it is clear that it is very possible to achieve sophisticated computer generators of word play and punning riddles with applications to computer interfaces and educational programs for the future.

## References

- [1] Binsted, K., Bergen, B., Coulson, S., Nijholt, A., Stock, O., Strapparava, C., ... Manurung, R. (2006). Computational Humor. *Intelligent Systems*, 21(2):59-69.
- [2] Ritchie, G. (2005). Computational mechanisms for pun generation. In *Proceedings of the 10th European Natural Language Generation Workshop*, pages 125-132, Aberdeen.
- [3] Ritchie, G. (2001). Current directions in computational humour. *Artificial Intelligence Review*, 16(2):119-135.
- [4] Binsted, K. and Ritchie, G. (1994). An implemented model of punning riddles. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 633-638, Seattle, USA.
- [5] Ritchie, G., Manurung, R., Pain, H., Waller, A., Black, R. and O'Mara, D. (2007). A practical application of computational humour. In *Pro-*

*ceedings of the 4th International Joint Conference on Computational Creativity*, pages 91-98, London.

- [6] Johnson, J. (2008, March 1). How to write original jokes (or have a computer do it for you) [Web log post]. Retrieved from <http://grok-code.com/12/how-to-write-original-jokes-or-have-a-computer-do-it-for-you/>
- [7] Stock, O. and Strapparava, C. (2005, June). HAHAcronym: A Computational Humor System. In *Proceedings of the ACL Interactive Poster and Demonstration Sessions*, pages 113-116, Ann Arbor.
- [8] Puns and other word play. (2001, August 1). *BBC - h2g2*. <http://www.bbc.co.uk/dna/h2g2/A592643>

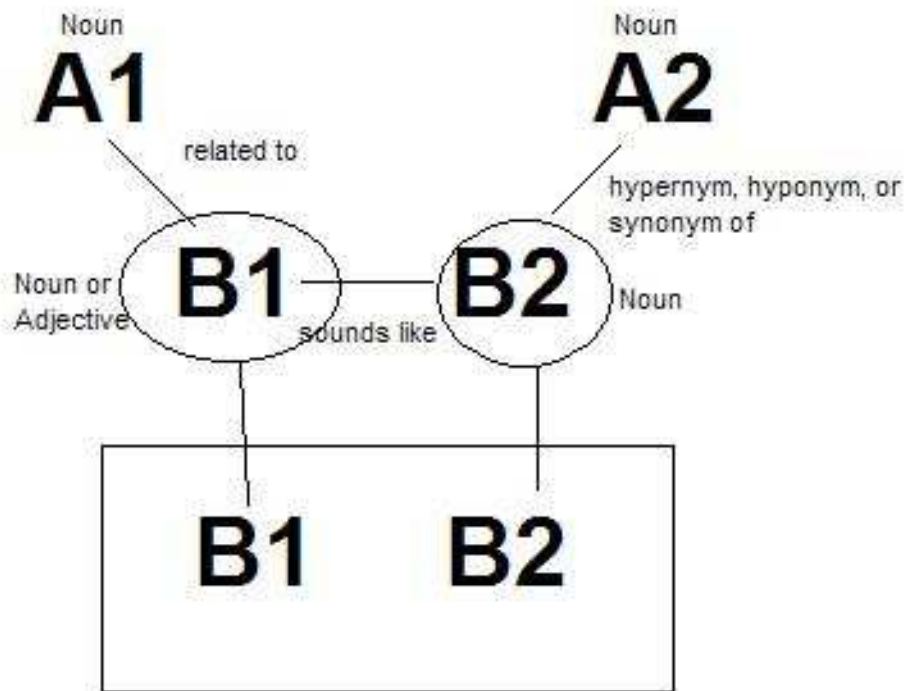


Figure 1: Schema for the “What do you get when you cross” joke

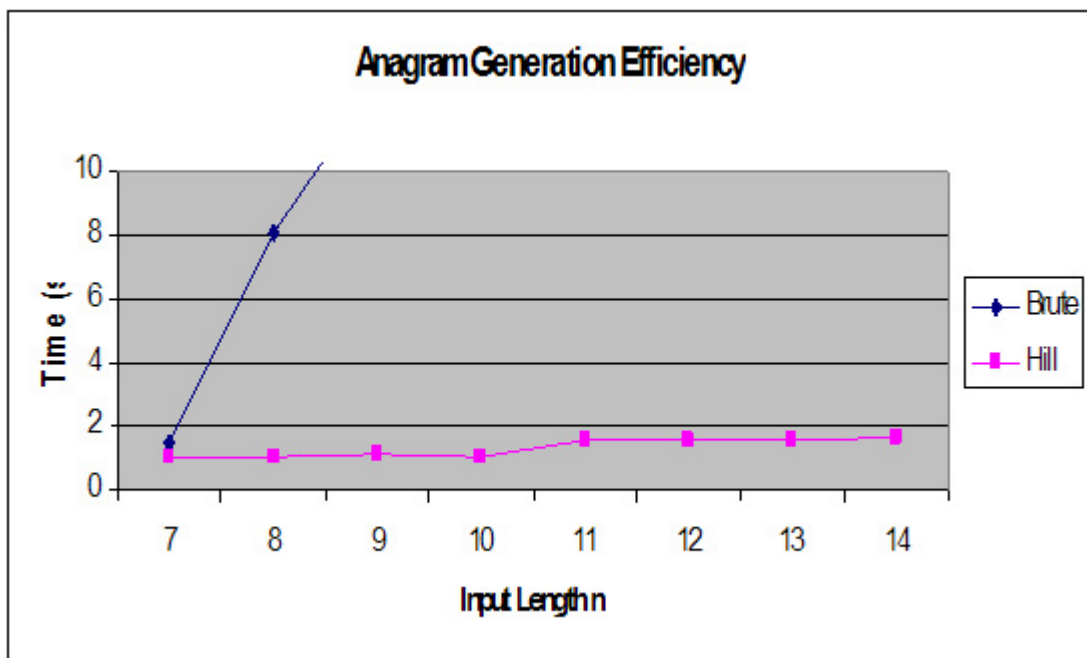


Figure 2: Runtime efficiency for brute force and hill-climbing anagram generation methods