Iambic IBM AI: The Palindrome Discovery AI Project

Eric V. Siegel
Computer Science Department
Columbia University
New York, NY 10027
evs@cs.columbia.edu

Abstract

In this paper, I describe an AI laboratory assignment in which students implement standard search techniques and explore heuristic measures of their own design for a palindrome discovery system. The system successfully derives palindromic sequences of words, many of which are meaningful, and achieves what is to the author's knowledge the first automatic generation of palindromes. Code is made available to students which implements the state space for palindrome search. This makes a large-scale problem accessible to introductory AI students by harnessing their knowledge of natural language. Students were motivated by the intrigue of discovering new palindromes.

1 Introduction

Most computer science degree programs have introductory courses in artificial intelligence (AI) [5], and, despite high variety in approaches to teaching AI, search is the first technique covered by most introductory AI textbooks. This is because search is a fundamental concept, serving as the basis for a great portion of advanced AI techniques such as planning, natural language generation, game playing, constraint-satisfaction (e.g., configuration) and automatic induction (machine learning). It is a rudimentary component of AI's conceptual framework, needed for understanding and thinking about most AI problems and solutions.

However, for many introductory AI students, the concept of search remains abstract and unmotivated. This is because systems that employ little more than search in their technique tend to only approach small toy-problems or micro-worlds. The difficulty is, most interesting problems are too advanced in that they require additional knowledge not yet available to a new AI student. However, just as AI trends have been moving to techniques that handle larger, "real world" scope, so too have many introductory laboratory assignments [9], such as the application of machine learning to face recognition [8] and the board game Othello [3].

In this paper, I describe an AI laboratory assignment in which students implement standard search techniques and explore heuristic measures of their own design for a palindrome discovery system. The system successfully derives palindromic sequences of words, many of which are meaningful (e.g., "SUN, a Java Janus."), and achieves what is to the author's knowledge the first automatic generation of palindromes. Code is made available to students which implements the state space for palindrome search (available on-line [11]). This makes a large-scale problem accessible to introductory AI students by harnessing their knowledge of natural language. Students were motivated by the intrigue and challenge of discovering new, original palindromes. This project could also provide an exercise in recursion, queuing functions, search and/or OOP for cs1 or cs2.

2 The Charm of Palindromes

Palindromes, which are spelled the same forward and backwards, are likely to attract the interest of many students because of their intrinsic appeal – it is fascinating to see what is possible when placing such a mundane, syntactic constraint on a sequence of letters that are to combine at a profoundly higher, semantic level.

Classic palindromes created by humans include, "Madam, I'm Adam", "A man, a plan, a canal; Panama", "Doc note, I dissent, a fast never prevents a fatness; I diet on cod", "Net safety? Byte fasten" (possibly the first palindromic joke, created by the author; coincidentally, the system subsequently found "net-safe fasten"), "On AI, play Al piano" (created by the author[1]) and, "Satan oscillate my metallic sonatas." Such

[1] Relevance: the author sings with a piano to his classes

a palindrome is the "Holy Grail" of automated palindrome generation, tantalizing and motivating our students.

The infatuation and potential obsession with palindromes is expressed by many web sites (easily located with a search engine). For example, freenet.buffalo.edu/~cd431/palindromes.html lists hundreds of prime palindromes, and http://www.sirius.com/~asavage/savagepalindromes.html says, "Ever since I first heard that little bit of esoteric verse, I was addicted." Other similar word games include anagram creation, which has been implemented (see "Anagram Genius" at http://mmm.mbhs.edu/~bconnell/anagrams.html). A recently developed AI system performs at the level of a human expert on solving New York Times crossword puzzles, another word game of constraints [4].

Students should spend a few minutes attempting to create palindromes manually. In doing this they can begin to formalize the process they are following as an algorithm.

3 AI Search for Palindromes

For the purposes of this assignment, we approach the modest problem of finding a sequence of English words and proper nouns that is palindromic, with no constraints on grammaticality or meaningfulness. Therefore, resulting palindromes must be manually filtered to determine which ones are worthy of exposition. Incorporating additional heuristics to ensure grammaticality is a viable project for more advanced AI projects, and for the moment will provide the student food for thought.

To discover a palindromic string of words is a constraint satisfaction problem (CSP). In this case, the constraints are that each word come from a finite dictionary, and that each letter be the same as its symmetrically corresponding letter.

The first step towards approaching a CSP is to define the state space to explore with a search algorithm. Some constraints must be alleviated in determining the initial state from which to search – we may start either with a palindromic string of letters and hope to form words, or start with a string of words and hope to form a palindrome. The latter approach suffers from the potential problem that every change of a word affects a number of letters, and therefore can violate a number of palindromic constraints. However, words could be sought that minimize such violations, and this becomes closer to the former approach, which is the one we adopt.

A palindromic string of letters is defined simply by half of the string. Therefore, each search state is defined by a string of letters which, when read forward and then backwards, or vice versa, is a palindrome. An odd-length palindrome is created by skipping the letter where the direction is reversed. The initial state is created by selecting a word randomly from the dictionary, or with a word determined by the student.

Operations that transform a state into a successor state simply *realize* a word within the string of letters, either forward or backwards, which may or may not entail adding letters to the string. For example, consider the initial state "safety". The word "byte" can be realized in reverse, resulting in "safetyb".

Each state also determines what words have been realized within the string of letters, in both directions, and which letters are as-yet unused, in both directions. The current example has one word going forward, one unused letter going forward ("b"), one word going backwards, and three unused letters going backwards ("fas").

Continuing with this example, these three letters start the word "fasten", which can be realized in reverse, resulting in "netsafetyb". This state has two words going in reverse, but only one going forward. The letters "net" as well as "b" have not been used in the forward direction. "Net" is a word in the dictionary, so it can be realized (this operation does not actually add any letters), resulting in the same string of letters, but with only one unused letter, "b".

Goal states are recognized if all letters are used in both directions, or if all but one letter at either the beginning or end of the string are used. Therefore, the current example is recognized as a goal, and is read forward and then backwards, skipping the "b" on the way forward, adding spaces between words: "net safety byte fasten".

In general, possible operations are defined by realizing words that "hang off" the beginning or end of the string where there are unused letters. Symmetrically, these operations are also defined for the string going in reverse.

These operations can result in *islands*, sequences of unused letters that are "trapped" mid-string, surrounded by used letters. Therefore, additional operations are defined which realize a word that consumes all or part of an island, in either direction. Such operations never change or add letters; they just "eat up" unused but preexisting letters, committing them to be words (which already happened to fortunately be there).

There are often few (or no) options for operations that consume part of an island, compared to options that involve adding letters. Therefore, the successor function is constrained so that if an island exists, successor states only include those resulting from operations that consume part of the island. As a result, if there is an

[12], including two song on AI topics [11].

island, but there are no words in the dictionary to fill (part of) the island, the successor function returns the empty set.

This built-in heuristic of the successor function serves as an effective pedagogical example of the most-constrained-variable heuristic for CSPs [1, 10]. It forces the searching algorithm to fill islands first, so that less time is wasted on states with islands that have no real words. This can be thought of as a "failure", e.g., at a terminal of depth-first search.

A dictionary of 80,282 word forms was derived from the CELEX lexical database [2] and the spell-checking dictionary on Unix systems. This includes many proper nouns (e.g., countries and common names), acronyms, and all inflections and word forms (e.g., singular and plural nouns, verb inflections and declensions).

4 Student Assignment

We provide for the students an implementation of the state space in Java. In particular, this is simply a class, PalindromeState, with a method, Expand(), that returns a set (Vector) of PalindromeStates. This is the set of successor states achievable by legal search operations. PalindromeState can be initialized with a string, or it will select a random word for initialization. Also available are boolean GoalTest(), and a print() method. The full homework assignment and implementation are available on-line (as well as a cool 3-D animation) [11].

4.1 Uninformed search

Given the implementation, it is very simple for students to immediately implement an uninformed search (a.k.a. blind search) from scratch using a queue. Alternatives include breadth-first, uniform cost, depth-first, depth-limited, and iterative deepening search – students can be assigned to implement a subset of these.

For the reader not familiar with such search techniques, here is one example:

```
Depth-First-Search(state s)
    if (s.GoalTest())
        s.print()
    else if (s is not too long)
        for each member, x, of s.Expand()
            Depth-First-Search(x)
```

When implemented with a queue/stack instead of recursion, the difference between search methods is simply a matter of alternative effects of the enqueue operation.

One pragmatic concern is that many states may have too large a number of successors, on the order of the dictionary's size. For example, "atop", with one word going forward, and one word, "pot", going backwards, has a dangling "a". Successors will include all words starting with that letter. Therefore, Expand() can be optionally flagged for only a (random) subset of successors, given a maximum size for this subset. This adds a stochastic element to palindrome generation, so multiple runs from a given initial state (word) provide multiple palindromes with that word.

This is a modification of canonical search strategies, and provides an opportunity to explore a second heuristic implicitly implemented by the successor function: If a word can be realized (touching an edge, or leaving only one unused letter on the edge) without adding new letters to the state, the resulting state is invariably included in the set of successors (even if the rest of the set is selected stochastically), since it is closer to being a goal state.

4.2 Heuristic functions

In order to do informed search methods such as greedy search, A* search, hill-climbing, or simulated annealing, students must implement a heuristic function that produces a numerical rating for a state.

PalindromeState has several additional methods to access characteristics of the state in order to implement this type of evaluation measure. This measure should be implemented by students as a method for a class deriving from (subsumed by) PalindromeState. The following methods are available, and reflect measures over the entire palindromic string (not just the "half-string" that suffices to define a state):

- `int length()` - number of letters
- `int nWords()` - number of realized words
- `char getLetter(int i)` - letter at location i
- `boolean used(int i)` - is letter i part of a realized word?

The range of possible heuristics is open-ended, and up to the student's imagination. Reasonable components include:

- sizes and numbers of islands
- sizes and numbers of unused strings that touch an edge
- number of unused characters
- number of consonants in a row
- all of the above relative to total number of letters

4.3 Comparing search techniques

Students are now in a position to conduct empirical experiments comparing various searches and heuristics. The goal is to minimize the number of search nodes generated per discovery of a palindrome. This "failure rate" or "density function" can be measured over multiple search runs, given a choice of: heuristic, search method, maximum number of words per palindrome, and maximum successor function set size. Furthermore, this process may be modified by placing an upper limit on the number of expanded nodes before restarting the search, since some initial states prove to be difficult to search from. Additionally, when a palindrome is found, the search does not have to end, since many other palindromes may be found from earlier choicepoints in this same search.

4.4 Other extensions

For advanced or extra credit projects, many other extensions to this system could be used to improve the grammaticality or subjective desirability of resulting palindromes. These can be in the form of heuristics, search constraints, or post-processing. Some possibilities include:

- Incorporating selectional constraints on words based on part-of-speech to improve grammaticality.
- Filter the system's output with a natural language parser to find grammatical palindromes.
- Methods to find palindromes that follow collocational constraints on words, e.g., "take-bath," "make-deposit," or "look-up" [13].
- Methods to put semantically related words together in the same palindrome, e.g., with WordNet, a lexicon that encodes semantic relationships [7].
- Heuristics designed to increase the desirability of palindromes could be evaluated by manually counting the number of discovered palindromes deemed meaningful. Furthermore, such heuristic measurements could be used to order resulting palindromes in an attempt to minimize the number that must be searched manually for meaningfulness.
- Alterations to the dictionary:
  - slang words
  - other languages ("Amigo no gima.")
  - the subset of words that are dominated with common letters
  - the subset of words with simpler letter patterns, e.g., not too many consonants in a row
  - a simple, small word set ("Madam, I'm Adam.")

5 Results

Students were entertained and thus educated by this project. They were motivated and delivered impressive write-ups. Midterm exam results indicate that most students understand the principles of search techniques. In the end, students were very roused when I presented the list of their colleagues' resulting palindromes.

Because the homework assignment write-up, derived from this paper, is relatively long (available on-line [11]), students were initially intimidated to start. But after being convinced that the actual coding on their part was to be minimal, the conceptual scale of the project resulted in a great amount learned for a small amount of implementation.

Numerical experimental results demonstrate significant differences in efficiency across search methods and heuristic functions. For example, one student reported the average number of states searched per discovered palindrome as 2,338, 107, and 32,971 for depth-first, greedy and hill climbing search procedures, respectively. In this case, a maximum length of six words was set, maximum successor function set size was five, hill climbing restarted (a maximum of 1000 times) at dead ends, the heuristic for the latter two was simply the number of unused letters, and results are averaged over 50 runs each.

The following results of the palindrome discovery system, under the guidance of student code, were deemed meaningful by students, who readily shared them with the instructor (a longer list is available on-line [11]). Punctuation and capitalization for all palindromes in this section were determined manually. *Iambic IBM AI. / Sex if fat affixes. / Sad, I misuse Jesus, I, Midas. / Masseuse sues Sam. / Massive Levis, Sam! / Infidels Led Ifni. / Spacecraft farce caps. / Eureka mixes sex – I make rue. / Len, serf Fresnel.* (This student's friend Len's computer is named Fresnel!) */ Sex or excel? Alec xeroxes. / Trade man named art. / Red dustcart tame hem attracts udder. / Red limpid dip milder. / Sub in money: yen, omnibus. / Sun ate Thor. Oh, tetanus!*

The remaining palindromes in this section are from uninformed, depth-first search with a maximum of 6 words per palindrome. About 1% of produced palindromes were deemed meaningful by the author. The implementation produces several thousand palindromes a day.

The following were deemed grammatical by the author, and were produced from searches seeded with random words: *Infinitesimal clam I set in Ifni. / Twenties acne? Encase it, Newt. / Name none, Dr. Arden: one man. / Revolted I – Bidet, lover! / Deepmined gossip piss, Ogden – I'm peed. / Wow! Wop powwow. / Elf farm raffle. / Net-safe fasten. / Madame Nile pips pipeline, madam. / Tangy gnat. / Evade Dave. / Re-sampled art? Trade LP maser. / A mini-mower crew-o-minima. / Nomad do-gooder, redo. O, goddam – on! / Redo? O Goddam – on, nomad do-gooder! / Rue to base: Saboteur! / Wo, honey's deeds – ye nohow! / Hex, a jam! Ajax, eh? / A ha-ha-foolery: Gyre loofah, aha! / ETA misdeeds? I mate.*

The following were not really deemed grammatical, but win honorable mention: *Gentlewoman I dine mo' – welt neg. / Spacecapsule dome remodel U.S. pace caps. / Sex elicit ore erotic ilexes / A retaliatory pyro tail at ERA. / Nil murderer, evil liver, ere drumlin. / Nob Beebe, oh Phoebe, ebb on.*

The following were produced for this paper from searches seeded with one of the words: *AI, OOP, Java, taught, SIG,* or *computer*: *Poor OOP. / SUN, a Java Janus.* (A two-faced God) */ Dumb, O Java; Java job mud. / ALA me taught: Evil liveth Guatemala. / De-*

bra, Java, LSD, Slav; ajar bed. / Plato grasp OOP – poops argot alp. / Poofs note rap – pareto NSF-OOP. / Poops? I lewd! We LISP OOP. / Poor iambics USC IBM air OOP. / Ahem, ANSI SIG: Isis name – ha! / Ah computer, fret up; mocha. / "Adieu; grasp OOP - oops!" argue Ida. / None: Decimal SIG, Islamic Eden, on.

6 Conclusions

Working on a problem that students care about solving and that operates over full-scale data is felicitous for learning introductory AI concepts. With the palindrome discovery project, the student's first experience with AI works at a scale and complexity on par with real-world problems, but is guaranteed gratifying results. Moreover, this project provides an opportunity to illustrate and explore implicit heuristics that are implemented as constraints in the successor function, e.g., "islands-first," a most-constrained-variable heuristic.

This problem hooks students intuitively by tapping into their understanding of natural language. Informally, formulating a sentence is a search ("searching for the right words"), and, in fact, work in automatic natural language generation formulates sentence generation as a search to fulfill multiple constraints, such as maximal grammatical nesting, selectional constraints between words [13], and the logical ordering of ideas [6].

The development of heuristic functions for palindrome search is elusive, thought-provoking, and challenging. Since some work better than others, students are exposed to experimental work immediately, which is appropriate for a new field such as AI that is predominated by research work.

Parts of this assignment should also be appropriate for cs1 or cs2 to exercise concepts in recursion, queues, search, and/or object oriented design. In general, search algorithms could be taught as fundamental computer science concepts, instead of limiting them to AI courses.

Acknowledgments

Thanks to Hongyan Jing and Dragomir R. Radev for help with the dictionary of word forms. Thanks to my mom, Lisa A. Schamberg, for sparking my interest in palindromes. Thanks to Mom, Alexander "sex elicits Alex; elastic ilexes" Chaffee, Eleazar Eskin, David Evans, Melissa L. Holcombe, and Carl L. Sable for comments on a previous version of this paper.

Finally, the following AI students provided palindromes from their systems that were chosen for inclusion in Section 5: Arie Addi, Jun Hou, Tarun Kapoor, Boyle Lee, Daniel Liu, Gautam Nair, Corey Robert Tripp, Jenny Weisenberg, and Haoqiang Zheng. The numerical results in Section 5 were derived by Ying Wang.

References

[1] Bitner, J., and Reingold, E. Backtrack programming techniques. *Communications of the Association for Computing Machinery 11*, 18 (1975).

[2] CELEX. The celex lexical database. CD-ROM, 1995. Centre for Lexical Information, Max Planck Institute for Psycholinguistics, Nijmegen.

[3] Eskin, E., and Siegel, E. Genetic programming applied to Othello: Introducing students to machine learning research. In *30th Technical Symposium of the ACM Special Interest Group in Computer Science Education* (March 1999).

[4] Keim, G., Shazeer, N., Littman, M., Agarwal, S., Cheves, C., Fitzgeral, J., Grosland, J., Jiang, F., Pollard, S., and Weinmeister, K. Proverb: The probabilistic cruciverbalist. In *Proceedings of the National Conference on Artificial Intelligence* (1999), AAAI Press.

[5] McCauley, R., and Manaris, B. Comprehensive report on the 1997 survey of departments offering csac/csab-accredited degree programs. Tech. rep., The Center for Advanced Computer Studies, University of Southwestern Louisiana, 1998.

[6] McKeown, K. *Text Generation*. Cambridge University Press, New York, 1985.

[7] Miller, G., Beckwith, R., Felbaum, C., Gross, D., and Miller, K. Introduction to WordNet: An online lexical database. Tech. rep., 1993.

[8] Mitchell, T. *Machine Learning*. The McGraw-Hill Companies, Inc., New York, 1997.

[9] Russell, S., and Norvig, P. A modern, agent-oriented approach to AI instruction. In *Proceedings of the AAAI Fall Symposium on Innovative Instruction for Introductory AI* (New Orleans, Nov. 1994), AAAI Press.

[10] Russell, S., and Norvig, P. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, New Jersey, 1995.

[11] Siegel, E. *My home page (for the homework project, on-line palindrome system and results thereof)*. http://www.cs.columbia.edu/~evs/, 1999.

[12] Siegel, E. Why do fools fall into infinite loops: Singing to your computer science class. In *4th Annual Conference on Innovation and Technology in Computer Science Education (SIGCSE-sponsored)* (June 1999).

[13] Smadja, F., and McKeown, K. Using collocations for language generation. *Computational Intelligence 7*, 4 (December 1991), 229–239.