Artificial Intelligence (W4701)
Homework 1
Search: Problems, Practicalities, and Palindromes

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

Abstract

SUN, a Java Janus.

For this assignment, 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 deduction of palindromes. Code is made available to students which implements the state space for palindrome search.

Due date: Oct. 5. However, you are encouraged to do this by Sept. 30, since at that time a 1.5 week assignment will be handed out.

1 Reading and Written Exercises

1.1 Reading

Chapters 1 - 4 from Russel and Norvig (Section 4.3 is optional). Also, you are responsible to read this paper to understand the state space for palindrome search, even if you choose to do the alternative search assignment.

For students who do not wish to program in Java, alternative programming assignments are given at the end of this writeup, below.

1.2 Written Work

- Exercises 3.1.
- Could bidirectional search be used for iterative improvement problems such as n-queens, VLSI and palindrome creation? If so, how? If not, why not?
- For your favorite palindrome (created by anybody or anything), show a possible (search) path (i.e., series) of operations that could have derived it. You'll want to read the rest of this paper before doing this.
- Exercises 4.4, 4.5, 4.6, 4.8.
- Exercise 4.13 (parts a, b, c and d only – no implementation required)
- Show a trace of A* search on the U2 puzzle, with the heuristic simply being the value of the slowest person who still needs to cross the bridge. Assume the search space is defined to only allow operations that bring 2 people across the bridge towards the destination, or one person back, always with the flashlight. To remind you of the problem: four people need to get across the bridge, 2 at a time at most, with a flashlight. They take 1, 2, 5 and 10 minutes, respectively. When two go together, the go at the speed of the slower person. This can be done in 17 minutes. Follow a link from the course webpage to try out a demo.
- Describe why the U2 heuristic above is admissible. Then, invent an admissible heuristic for the U2 puzzle that is better. In 2 or 3 sentences, provide an informal proof that your heuristic is indeed the better of the two (cf. the text for help on comparing heuristics).
- *Read this question a few times in order to understand it – it is secretly a crash course in machine learning.* Describe the initial state, goal test, operators, a reasonable heuristic and a path cost function (just 1 or 2 sentences each) for a system that searches for a good rule to classify a person according to her or his gender, nationality, hair color and eye color as to whether the person would be allowed into the (fictional) Exclusive Styx Fanclub. Assume you have a database of people and whether or not they were accepted (and your rule, developed from this data, will make a generalization about type of person get's accepted). Assume the rule is of the form "If eyecolor = blue and nationality = Canadian and gender = WILDCARD (either gender accepted) and haircolor = green then AcceptedAsMember". There are many ways to approach this problem, so there are many acceptable formulations for the operators, etc.

2 Introduction

Infinitesimal clam I set in Ifni.

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.

For this homework, you will implement standard search techniques and explore heuristic measures of your own design for a palindrome discovery system. The system successfully derives palindromic sequences of words, many of which are meaningful (e.g., "Name none, Dr. Arden: one man."), and achieves what is to the author's knowledge the first automatic deduction of palindromes. Code is made available to students which implements the state space for palindrome search (available on-line at http://www.cs.columbia.edu/~evs/ai/).

3 The Charm of Palindromes

Revolted I – Bidet, lover!

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 list of words that are to combine at a profoundly higher, semantic level. To the author's knowledge, there has been no previous work in automatic palindrome generation.

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 palindrome joke, created by the author; coincidentally, the system subsequently found "net-safe fasten"), and, "Satan oscillate my metallic sonatas." Such a palindrome is the "Holy Grail" of automated palindrome generation, tantalizing and motivating our students.

The infatuation and potential obsession with palindromes is reflected by many web sites easily located with a web search engine. For example, freenet.buffalo.edu/~cd431/palindromes.html lists hundreds of premier 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 system performs at the level of a human expert on solving crossword puzzles, another word game of constraints [3].

Students can spend a few minutes on their own attempting to create palindromes. In doing this they can think about how the process they are following could be formalized as an algorithm.

4 AI Search for Palindromes

Elf farm raffle.

For the purposes of this assignment, we approach the modest problem of finding sequences of English words and proper nouns that are palindromic, with no constraints on their 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 for the search. 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 find 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 user.

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 is therefore 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 forward direction. "Net" is a word in the dictionary, so it is 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, surround by used letters. Therefore, additional operations are defined which realized a word that consumes all or part of an island, in either direction. Such operations never change or add more letters to the string; they just "eat up" unused but pre-existing letters, committing them to be words (which already happened to fortunately be there).

There are often less (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 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, 6]. 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).

5 Student Assignment

Dumb, O Java; Java job mud.

You're job is to implement depth-first, greedy, A*, and hill-climbing search. Also implement a special "greedy depth-first" in which the queuing function adds new nodes according to their heuristic value (that is the order in which the new nodes are added to the front is according their heuristic value, but when new nodes are added, the order of the nodes already in the queue does not change – all the successors of the expanded node are added to the front, but are ordered first). Additional algorithms can be optionally implemented.

Additionally, at least 3 heuristic functions should be implemented. These heuristics can be similar to one another.

Several combinations of search algorithms and heuristic functions are compared for success at finding palindromes, according to the details given below in Section 5.3.

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.

5.1 Uninformed search

Given the implementation, it is simple for students to immediately implement an uninformed search (a.k.a. blind search) from scratch using a queue.

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, an option that is probably necessary is to flag Expand() for only a (random) subset of successors. This adds a stochastic element to palindrome generation, so multiple runs from a given initial state (inputted word) will 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 (within an island, 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 complete palindrome.

5.2 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.

The development of heuristic functions for palindrome search is elusive, thought-provoking, and challenging. Since some are bound to 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.

PalindromeState also has several 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

Additional heuristics may be designed to increase not only the likelihood of finding a palindromic string of words, but also to increase the subjective quality of discovered palindromes, e.g., number of words, average size of words, etc.

5.3 Comparing search techniques

You 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" should 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 (the latter two may be left at their default values – see the code's README file). 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 (your choice). 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 (your choice).

Multiple runs may need to go overnight to collect enough data.

5.4 Writeup and Collaboration

Make an organized table of your results, and an accompanying paragraph that describes the table's contents, and what choices you made. Then write another short paragraph drawing any intuitive conclusions you can from the results shown in your table – which search and heuristic was most successful, and can you hazard a guess as to why?

You have the option to team up and collaborate with other students for this part. Although everyone is responsible for their own implementation of the search algorithms, you may team up and design complementary sets of experiments, if you so desire. In this case, hand in two separate submissions, but the writeup summarizing the results can be the same across submissions – make it explicit who you worked with. It is recommended to not work with more than one other person, and a team of 3 is the max allowed.

5.5 Other extensions – OPTIONAL

Many other optional extensions to this system can be used to improve the grammaticality or subjective desirability of resulting palindromes. These are listed here to primarily just to provide food for thought. These can be in the form of heuristics, search constraints, or post-processing. Some possibilities include:

- 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.")

- Heuristics designed to measure the subjective desirability of palindromes can be evaluated by manually counting the number of meaningful or grammatical palindromes discovered over multiple searches. Furthermore, the heuristic measurements can be used to order resulting palindromes in an attempt to minimize the number that must be searched manually for meaningfulness.
- The following advanced extensions are most definitely outside the scope of this assignment:
  - Allow it to realize words such that they wrap around the edge of the "half-string". Certain words that are partially palindromic can do this, e.g., "Adam" in "Palm, Adam; lap", and "Toyota" in "A Toyota".
  - 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".
  - Methods to put semantically related words together in the same palindrome, e.g., with the Wordnet, a lexicon that encodes the semantic relationships between words [5].

6 Results

Students are asked to email the instructor (evs@cs.columbia.edu) with particularly good palindromes to be added to an on-line collection.

The following example results 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 current implementation produces several thousand palindromes a day.[1] Punctuation and capitalization for all palindromes in this section were determined by the author.

The following were produced from searches seeded with one of the words: *AI, OOP, Java, taught, SIG,* or *computer*:

- On AI, play Al piano.
- Poor OOP.
- SUN, a Java Janus. (God)
- Dumb, O Java; Java job mud.
- ALA me taught: Evil liveth Guatemala.
- Debra, 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!

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!
- Wo, honey's deeds – ye nohow!
- Hex, a jam! Ajax, eh?
- A ha-ha-foolery: Gyre loofah, aha!

[1]The author is glad to email you a slew if you'd like to hash through some!

- ETA misdeeds? I mate.

The following were not 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.

7 Related Work

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 [7], and the logical ordering of ideas [4].

8 Alternative to Palindromes

For students who do not wish to program in Java, your assignment is to implement the same search strategies and do the same comparisons of heuristics and search as described above, except choose one of the following search problems:

- 8-puzzle
- U2 problem with varying band sizes (4 members, 5, etc.)
- That solitary game in which pegs jump over other pegs and then you get to remove a peg and the goal is to leave only one peg.
- Palindromes – implement it from scratch, using my dictionary if you like. If you choose to do this you can formalize the search space in a different way if you have another idea as to how to do this. But in any case, be warned, it will be a fair amount of overhead.
- Anagrams.
- Another alternative of your choosing (but email the instructor for permission first).

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—Dutch, English, German. CD-ROM, 1995. Centre for Lexical Information, Max Planck Institute for Psycholinguistics, Nijmegen.

[3] 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.

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

[5] Miller, G., Beckwith, R., Felbaum, C., Gross, D., and Miller, K. Introduction to wordnet: An on-line lexical database. Tech. rep., 1993.

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

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