ON PRUNING TREES FOR PANGRAMS

LEONARD GORDON
Tucson, Arizona

In the February and May issues of **Word Ways**, Lee Sallows describes how he constructed a pangram machine (a special-purpose digital computer) to discover self-referential sentences of the form "This sentence contains five a's, two b's, ... and one z", after he realized that the program he had written for a general-purpose digital computer would trake 31.7 million years to run. There's no doubt that building the pangram machine was a personally satisfying accomplishment -- although I am afraid that it became his Galatea.

However, he gave up too easily on the general-purpose computer. Better pruning logic could have saved the day; in particular, his attempt to survey the entire board after each series of moves is, I fear, a loser.

Some years ago, I chose to compusolve the four-by-four sliding block puzzle (the 15 Puzzle). Recreational mathematics books discuss the trivial matter of possible vs. impossible arrangements, but the challenge is to find a minimum-move sequence from one arrangement to another. When planning my attack, I realized that the problem would take forever unless I introduced tree-pruning logic. The obvious approach is to monitor the ideal distance from home and cut when it exceeds the allowed. I discussed the problem with my brother, Jerry, a former electrical engineer with Sylvania who built gadgets like Sallows' pangram machine back in the 1950s. When I complained that surveying the board after each move would be too costly in computer time, he proceeded to write his first (and only) computer program to show me how it should be done. Calculate the ideal distance at the beginning of the search and then modify it after each move. When block x moves from cell y to cell z, read the change, T(x,y,z), from a table prepared earlier.

Tree-pruning logic is applicable as well to finding pangrams, and, in fact, is the key to their successful construction by computer. But, unless the programmer uses logic like mine, he will not get a program that converges in reasonable time no matter how fast his language and machine. (Did Sallows realize this when he decided to construct a pangram machine instead? There is nothing in his article to indicate it.)

It took me about a week to write and debug a program for the pangram problem. Handling the logic was easy enough since I was familiar with it, but there is considerable bookkeeping involved (plenty of room for typos). After my initial run using one of Sallows' results as a test case, I estimated that it might take from three to twelve hours to search for a solution to a problem. Over a period of another two to three weeks, I reduced that to 30 minutes to two hours by introducing branch-cutting logic. Here is one of the first pangrams my computer found -- a fitting sequel to Sallows' hundred:

> this umteenth dumb pangram has five a's, two b's, one c, three d's, thirty-two e's, six f's, three g's, nine h's, ten i's, one j, one k, one l, four m's, twenty-one n's, fourteen o's, two p's, one q, eight r's, twenty-seven s's, twenty-one t's, five u's, six v's, seven w's, three x's, five y's, and one z

I do not know how my time of 26.2 minutes to solution, 45.7 minutes to exhaust the search for this particular problem, compares with Sallows' machine. His discussion is vague, but I think he says it's about two hours per run after he has chosen reasonable search limits. (In this case mine were e 28-35, f 6-9, g 2-5, h 5-10, i 9-13, l 1-4, n 18-23, o 12-17, r 6-10, s 25-30, t 20-25, u 2-6, v 3-6, w 6-11, x 2-5 and y 3-6. These limits are tight; I made a good guess.)

It's better to expand the limits somewhat so as to exhaust the search in about two hours - a compromise between run (worrying) time and the likelihood of finding a solution. I use QuickBasic 4.0 to program on a 286/12 machine. I estimate that someone using a faster language on a faster home computer could reduce my search time by a factor of 30 - possibly even 100; I have no idea what the run time would be on a mainframe computer.

Sallows estimates that one of eight sentences has a solution, and that one of 64 has a second solution as well. I have not tested those estimates, but I suspect one could do better if wider search limits were used. Sallows found only one sentence with three solutions. I found that the first sentence in his list of 100 has two solutions in addition to the one he gave. For this search, I used the same limits as above except that a slightly wider range (16-23) was allowed for n. The time to exhaust the search was 70.7 minutes.

This first pangram has

five	five	five	a's
one	one	one	b
one	one	one	c
two	two	two	d's
twenty-eight	twenty-nine	twenty-eight	e's
six	six	eight	f's
five	four	four	g's
eight	eight	seven	h's
twelve	twelve	thirteen	i's
one	one	one	j
one	one	one	k
three	three	one	l's

two	two	two	m's	
sixteen	nineteen	twenty-one	n's	
twelve	twelve	fifteen	o's	
two	two	two	p's	
one	one	one	q	
seven	eight	six	r's	
twenty-eight	twenty-six	twenty-six	s's	
twenty	twenty	twenty-one	t's	
three	three	two	u's	
six	five	five	v's	
nine	nine	nine	w's	
four	three	three	x's	
four	four	five	y's	and
one	one	one	z	

Here is a different approach to finding self-referential sentences. Start with a sentence like "This aeinoprst pangram has (or sentence contains)...". Relax the convergence criteria to allow a few errors, determine which letter-sums are wrong, add or subtract these letters to the aeinoprst store, and anagram the result to something meaningful. In this case, "a" remains in the store, while "p" may be replaced by b, c, k, j, etc. My own results are not very interesting, but someone skilled at anagramming might want to pursue this. At least this approach brings the problem into the realm of wordplay instead of pure algebra. Send me a base sentence and a suggested set of letters to play with, and I'll send you some toys.