A Genetic Algorithm for the Longest Common Subsequence Problem

Brenda Hinkemeyer and Bryant A. Julstrom
Department of Computer Science, St. Cloud State University
St. Cloud, MN 56301 USA
brenda@nikosha.net, julstrom@stcloudstate.edu

ABSTRACT

A genetic algorithm for the longest common subsequence problem encodes candidate sequences as binary strings that indicate subsequences of the shortest or first string. Its fitness function penalizes sequences not found in all the strings. In tests on 84 sets of three strings, a dynamic programming algorithm returns optimum solutions quickly on smaller instances and increasingly slowly on larger instances. Repeated trials of the GA always identify optimum subsequences, and it runs in reasonable times even on the largest instances.

Categories and Subject Descriptors

G.2.1 [**Mathematics of Computing**]: Discrete Mathematics—*Combinatorics*; I.2.8 [**Problem Solving, Control Methods, and Search**]: Heuristic Methods

General Terms

Algorithms

Keywords

Strings, longest common subsequence, genetic algorithm

1. INTRODUCTION

A subsequence of a string S is a sequence obtained by deleting zero more characters from S. Given two or more strings S_1, S_2, ..., S_K over an alphabet Σ, the Longest Common Subsequence (LCS) problem seeks the length of a longest subsequence found in all of the strings. A dynamic programming algorithm for the LCS problem due to Irving and Fraser [1] requires time that is $O(n^K)$, where n is the length of the strings.

2. AN EVOLUTIONARY CODING

For evolutionary search in the LCS problem, encode candidate sequences as binary strings as long as the shortest of the given strings (or the first if their lengths are equal) S_1. In a chromosome `c[·]`, `c[i]` $= 1$ indicates that S_1`[i]` is in the sequence `c[·]` represents; `c[i]` $= 0$ that it is not.

The fitness function rewards longer sequences; strongly rewards a chromosome for each string in which its subsequence appears; rewards a chromosome whose subsequence is as long as S_1; and very strongly penalizes a chromosome whose subsequence is not found in all of the strings. In the following sketch, v is a variable in which `c[·]`'s fitness $f($`c[·]`$)$ is developed, ℓ is the length of the subsequence `c[·]` represents (that is, the number of 1s in `c[·]`), m is the number of strings that the subsequence matches, S_1 is the shortest or first string, and K is the number of strings in the instance.

$v \leftarrow \ell + 30 \times m$
if $\ell = |S_1|$
  $v \leftarrow v + 50$
if $m = K$
  $v \leftarrow 3000 \times v$
else
  $v \leftarrow -1000 \times v \times (K - m)$
$f($`c[·]`$) \leftarrow v$

Random chromosomes are generated by assigning their positions 0 or 1 with equal probabilities. All chromosomes represent candidate sequences, so positional operators—k-point crossover and position-by-position mutation—can be applied to them.

3. A GENETIC ALGORITHM

A generational genetic algorithm uses the coding and operators just described. It initializes its population with random chromosomes. It chooses chromosomes to be parents in 2-tournaments, generates offspring chromosomes from parents by applying crossover or mutation with equal probabilities, and is 1-elitist. The GA can handle instances with any number of strings. In the tests described below, the GA's population contained 100 chromosomes. The probability that mutation flips each entry was $1/n$. The non-best chromosomes were re-initialized if no improvement in the fittest chromosome occurred for 100 generations. The GA ran until it found an optimum solution, which was always known.

4. TESTS

Test LCS instances, each with $K = 3$ strings, have alphabet size $|\Sigma| = 2$, 4, or 26, corresponding to binary, DNA, or English-alphabet strings; lengths $n = 100$, 200, 400, 800, 1600, 3200, or 6400; and maximum common subsequence lengths equal to 10%, 50%, 90%, or 100% of their strings' lengths. For each, S_1 was chosen and a subsequence of it was specified. This subsequence then appeared in the other strings, whose remaining positions were filled in such a way that no longer subsequence was common to all the strings.

Copyright is held by the author/owner(s).
GECCO'06, July 8–12, 2006, Seattle, Washington, USA.
ACM 1-59593-186-4/06/0007.

Table 1: The trials of the DPA and the GA on the 28 LCS instances with "DNA" strings; $|\Sigma| = 4$. For each instance, the table lists the length n of its strings, the length of a longest common subsequence as a percentage of n, the time that the DPA required to find the longest subsequence length, the shortest and mean times the GA required to find the longest length, and the standard deviation of those times. The DPA could not run on the larger instances.

Instance		DPA	GA time (s)		
n	%	time (s)	best	mean	stdev
100	10%	0.094	0.204	3.204	15.655
	50%	0.204	69.875	267.482	406.935
	90%	0.079	0.187	1.451	3.189
	100%	0.203	0.203	0.248	0.034
200	10%	0.782	1.047	1.298	0.139
	50%	0.734	2.016	2.345	0.206
	90%	0.625	1.297	1.942	0.306
	100%	0.719	0.844	1.028	0.106
400	10%	4.313	4.579	5.162	0.361
	50%	4.297	12.109	12.294	0.144
	90%	3.391	7.000	9.264	0.869
	100%	4.219	3.953	4.625	0.410
800	10%	1590.406	19.875	21.779	1.193
	50%	1420.532	37.625	41.700	2.284
	90%	936.125	35.157	39.507	2.159
	100%	1300.547	16.516	18.897	1.273
1600	10%	—	83.610	92.017	5.633
	50%	—	164.765	180.656	10.059
	90%	—	148.890	166.621	10.371
	100%	—	72.781	78.786	4.346
3200	10%	—	347.609	382.175	23.236
	50%	—	687.438	736.156	32.872
	90%	—	645.781	703.531	28.021
	100%	—	288.547	323.813	18.164
6400	10%	—	1433.047	1566.138	68.087
	50%	—	2938.422	3208.825	157.626
	90%	—	2788.547	2983.347	137.954
	100%	—	1231.718	1392.686	92.700

One instance was generated for each combination of $|\Sigma|$, n, and LCS length.

The dynamic programming algorithm (DPA) of Irving and Fraser and the genetic algorithm were implemented in Java and executed on an AMD 3400+ processor with one Gbyte of memory running at 2.2 GHz under Windows XP. The DPA was run once and the GA 30 independent times on each problem instance. Table 1 summarizes the results of these trials on the instances with "DNA" strings ($|\Sigma| = 4$); results on the binary and English-alphabet strings were similar.

The DPA's times were similar for each string length and small—no more than a few seconds—for $n = 100$, 200, and 400. On the instances with $n = 800$, the DPA's times suddenly ballooned. On the largest instances, the DPA required more memory than was available.

When $n \leq 400$, the GA's times were slightly longer than but comparable to those of the DPA. With $n = 800$, the GA's times grew, but nowhere near as much as the DPA's times. On the larger instances, the GA continued to identify longest common subsequences in reasonable times.

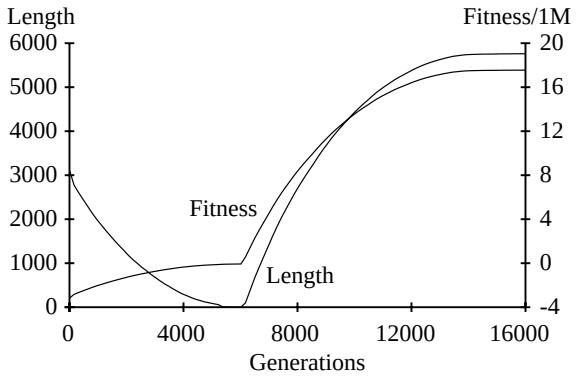

Figure 1: The length and fitness of the most fit sequence in the GA's population on the binary strings of length $n = 6400$ with LCS length $= 0.9n$.

In general, the GA generates short sequences that are found in all the strings, then extends them until one is as long as possible. Figure 1 illustrates this behavior through one of the GA's runs. Fitness increases monotonically, as it must under elitism, but the length of the fittest sequence drops at first, and its fitness is negative. Initially, the fittest sequence gets shorter as the GA searches for a subsequence that occurs in all the strings. Once such a subsequence is found, both length and fitness of the fittest subsequence increase, as the GA identifies longer and longer common subsequences.

Finally, because the GA represents candidate subsequences by selecting characters from the shortest or first string in the target LCS instance, it returns not only the length of a long common subsequence but the sequence itself. In contrast, the dynamic programming algorithm requires a second pass over its primary data structure to retrieve the LCS [1].

5. CONCLUSION

A genetic algorithm for the longest common subsequence problem encodes candidate sequences as binary strings that indicate selections of characters from the shortest or first given string. The GA was compared to a dynamic programming algorithm on 84 instances of the LCS problem, each with three strings over alphabets of two, four, or 26 characters, string lengths from 100 to 6400, and known LCS lengths of 0.1, 0.5, 0.9, and 1.0 times the string lengths. In repeated trials on the instances, the GA always found an optimum solution, a little more slowly than the dynamic programming algorithm on the smaller instances, but much more quickly on those with lengths of 800 and in reasonable times on the larger instances.

6. REFERENCES

[1] R. Irving and C. Fraser. Two algorithms for the longest common subsequence of three (or more) strings. In *Proceedings of the 3rd Annual Symposium on Combinatorial Pattern Matching*, pages 214–229, New York, 1992. Springer-Verlag.