The SAGA Cross: The Mechanics of Recombination for Species with Variable-length Genotypes

Inman Harvey

CSRP 22 , 1992

Cognitive Science Research Paper

Serial No. CSRP 22

The University of Sussex
School of Cognitive and Computing Sciences
Falmer
BRIGHTON BN1 9QH
England, U.K.

This paper appears in:
R. Männer and B. Manderick, editors,
Parallel Problem Solving from Nature 2, pages 269–278.
North-Holland, 1992

The SAGA Cross: The Mechanics of Recombination for Species with Variable-length Genotypes

Inman Harvey [a]

[a]School of Cognitive and Computing Sciences, University of Sussex, Brighton U.K.
inmanh@cogs.susx.ac.uk

1. Introduction

Genetic Algorithms (GAs) have traditionally tended to use genotypes of a predetermined fixed length. The designer of a particular GA, for use as an optimisation technique within a given search space, decides which parameters are to be represented on the genotype, how they are to be coded, and hence the genotype length. For each parameter there is a given position or set of positions on the genotype which unambiguously code for it. This can be loosely translated as: the allele (parameter or feature value) for a particular gene (parameter or feature) is coded for at a particular locus (genotype position). This makes it simple for a recombination genetic operator, therefore, to take the same crossover point in each parent genotype, and exchange homologous segments.

When variable-length genotypes (VLGs) are used, absolute position of some symbols on the genotype can usually no longer be used to decide what feature those symbols relate to. Some examples of ways around this problem are given in the next section. A related problem is, how can one organise a recombination operator so that the resulting offspring genotypes are, firstly, sensibly interpretable, and secondly, have inherited meaningful 'building blocks' from both parents.

VLG GAs have been proposed in various domains where they seem to allow a natural genetic representation for the problem under consideration, and the variety of domains is reflected in the variety of representations suggested. In this paper the motivation for needing VLGs is that of wanting to extend GAs so as to allow for open-ended evolution. Although GAs have borrowed ideas from natural evolution to use in function optimisation, what they have ignored is perhaps the most impressive feature of natural evolution: how over aeons organisms have evolved from simple organisms to ever more complex ones, with associated increase in genotype lengths. It has been suggested elsewhere that this feature of evolution will need to be used in the only practical way of developing autonomous robots [6, 10], and more generally this is an obvious approach to incremental design by evolution of engineering systems. The SAGA framework was introduced in [7] to incorporate the necessary extensions to standard GAs, and the present paper looks at the consequences for a recombination operator.

It will be suggested that in this context the identification of the locus of a 'gene', or that section of a genotype which codes for some particular feature, will necessarily be by use of an identifying template. The problem for recombination becomes then, given a randomly selected crossover point in one parent genotype, how to identify an appropriate

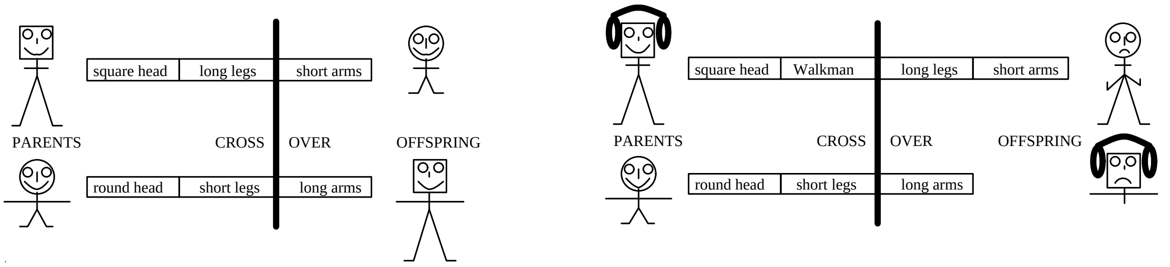

Figure 1. *A crossover operator which works well with fixed lengths may have sad consequences when unthinkingly applied to variable length genotypes.*

place to break the other parent genotype so as to exchange homologous sections as far as is possible. In this we will be aided by the fact that within the SAGA framework the genetic pool of a population will be largely converged to form a *species* or *quasi-species*; as shown in [7] and briefly summarised below.

idTd[(w)627.120 [illegible]

This solution relies on the hierarchical tree-decomposition of the genotype, and would not extend to genotype representations where the interactions between 'building blocks' cannot be so decomposed.

In Goldberg's Messy GAs, each locus on the genotype in effect carries its identification tag around with it. Instead of a crossover operator, *cut* and *splice* operators are used, which allow genotypes of any length to develop over time. But the number of loci is fixed at the start, and everything is in effect based on an underlying fixed-length representation, which may be underspecified or over-specified. In the former case, where a genotype does not contain an allele for every locus, the deficiencies are made up by a 'competitive template' scheme. In the latter case, conflicts can arise where the identity tag for a specific locus occurs more than once with different associated values; in this case an arbitrary rule is used, such as choosing the one nearest a specified end of the genotype. It should be noted that the solution to the under-specification problem relies on there being a predetermined number of loci, and cannot be extended to arbitrary numbers of loci.

Harp and Samad use a linear genotype to code for a neural network by having building blocks of a fixed length on the genotype code for the specification of an individual layer in the network, including the connectivity from that layer to other layers. The format for each block is the same, but the number of them is not fixed. A crossover operator can therefore be used which, when a crossover point in one parent genotype occurs inside a block, ensures that the crossover in the other parent genotype occurs in the same position within a block, thus exchanging homologous segm, of the

sense of searching around the current focus of a species for neighbouring regions which are fitter, or in the case of neutral drift, not less fit. Such a search takes place through application of genetic operators such as crossover, mutation or change-length. The latter two operators are discussed in [6]. In this paper we concentrate on crossover.

5. Where to cross

When evolving systems of arbitrary increasing complexity within the SAGA framework, it will be assumed that there are building blocks coded for along a linear genotype, and that interactions between such building blocks are mediated by some addressing system, as discussed earlier. For recombination it will be relevant that the population will be largely converged; any two parent genotypes will be broadly similar.

The SAGA cross has therefore the requirements that, given any chosen crossover point in one parent genotype, a crossover point in the other parent genotype needs to be chosen so as to minimise the differences between the swapped segments. This can be rephrased as: we should maximise the similarities between the two left segments that are swapped, and between the two right segments that are swapped. Please note that the VLG crossover problem that the SAGA cross handles *only* refers to the choice of the second complementary crossoaver.10.6sso

The rationale behind this algorithm is as follows: The length of the longest common subsequence of two strings A_{1i} and B_{1j} is to be written into $L(i,j)$. If $L(i-1,j-1)$, $L(i,j-1)$ and $L(i-1,j)$ are known, then $L(i,j)$ can be derived from them, the value depending also on whether or not the i^{th} symbol of A and the j^{th} symbol of B match.

$L(i,j)$ must be at least equal to the best of $L(i-1,j)$ and $L(i,j-1)$; and if the symbols do match, then $L(i,j)$ will be one better than $L(i-1,j-1)$. Algorithm B keeps track of the necessary amounts, and updates them within the j and i loops.

In Hirschberg's development, a further algorithm C is used to recursively use Algorithm B, by dividing a given problem into two smaller problems, bottoming out of the recursion when there are trivial subproblems. This is used to output the sequence which is the LCSS of A and B. The purposes of the present paper are rather different, and I have developed an algorithm D to solve the VLG crossover problem.

The initial step is to add a feature to algorithm B so that it will work with substrings, and equally well when comparing two strings enumerated from one end or from the other end. For this it is necessary to explicitly pass as inputs the initial and final indices for the substrings of A and B.

Algorithm D accepts input strings A_{1m} and B_{1n} of lengths m and n, and an integer c which represents the crossover point in A. As output it returns a vector M which keeps track of the current best-so-far candidates for a crossover point in B (which may be one point or a sequence of them). For intermediate calculations two vectors $L1(n+1)$, $L2(n+1)$ are used, which contain the outputs from two separate calls to algorithm B. Internal integer variables r, s and t are used

ABC-DCEF
B-E-C-D-C-D-F

	A	B	C		D	C	E	F	
-	0	0	0	3	3	2	2	1	BECDCDF
B	0	1	1	4	3	2	2	1	ECDCDF
BE	0	1	1	4	3	2	1	1	CDCDF
BEC	0	1	2	5	3	2	1	1	DCDF
BECD	0	1	2	4	2	2	1	1	CDF
BECDC	0	1	2	4	2	1	1	1	DF
BECDCD	0	1	2	3	1	1	1	1	F
BECDCDF	0	1	2	2	0	0	0	0	-

Figure 4. Algorithm D on ABC-DCEF (cross between C and D) and BECDCDF (best cross to be determined). On left, algorithm B gives in column C best scores matching substrings against ABC. On right, working backwards, best scores in column D. Central column shows best total (5 matches) given by splitting BEC-DCDF.

will decrease with occasional level stretches.

The purpose of lines 6 and 7 is to monitor this, and to store in M the values of i for the current best, or several best-equal, values of t. Hence when the loop finishes, the first s values in M contain the proposed crossover positions for maximising t, the sum of left and right LCSSs.

It is then possible to select at random one of the optimal positions, and return this as

7. Computational requirements

This

14 Thomas S. Ray. An approach to the synthesis of life. In J.D. Farmer, C.G. Langton, S. Rasmussen, and C. Taylor, editors, *Artificial Life II.* Addison-Wesley, 1992.

15 David Sankoff. Matching sequences under deletion/insertion constraints. *Proceedings of the National Academy of Science, USA*, 69(1):4–6, 1972.

16 Stephen F. Smith. *A Learning System based on Genetic Adaptive Algorithms.* PhD thesis, Department of Computer Science, University of Pittsburgh, USA, 1980.

17 R.A. Wagner and M.J. Fischer. The string-to-string correction problem. *Journal of the A.C.M.*, 21(1):168–17 , 1974.

Appendix

```
#include <stdio.h>
#include <string.h>
#define MAXLEN 1000  /* Max length of genes */
#define max(a,b) ((a)>(b) ? a : b)

/***********************************
A and B
```

```
  {
    last=1-(this=last); /* flip this/last   */
    for (j=0;j<lenb;j++)
      K[this][j+1]=
        (A[i]==B[(fwd==1 ? j : lenb-j-1)] ?
                /* are the chars matching ? */
          K[last][j]+1 :          /* yes or */
          max(K[this][j],K[last][j+1]));
                                      /* no */
  }

    /* internal calculations finished;
       copy into output                     */
  for (j=0;j<lenb+1;j++)
    L[j]=K[this][j];
}
```

```
/***********************************
Test program to read in file containing 2
gene strings, choose a random crossover point
in first, and select appropriate crossover
point in second.
***********************************/
main()
{
  FILE *fp;
  char gene1[MAXLEN]; /* strings for genes */
  char gene2[MAXLEN];
  int len1,len2;      /* lengths of genes  */
  int i,j,k,displace1,displace2;
  int cross1,cross2;  /* crossover points  */

  fp=fopen("genefile","r");
  fscanf(fp,"%s",gene1);
  fscanf(fp,"%s",gene2);
  fclose(fp);

  len1=strlen(gene1);
  len2=strlen(gene2);

    /* make sure cross1 is within gene1    */
  cross1=1+(random()%(len1-1));
  cross2=algd(gene1,gene2,len1,len2,cross1);

  printf("\n%d   %d\n",cross1,cross2);
}
```