Appendix 1

1 Important Algorithms

Algorithms which were found to be the most challenging are included in this section. The idea for the permutation generator was proposed to me by my project supervisor, whilst the rest are of my own design. By far the most challenging algorithm was the LDA, in §2.1.

1.1 Permutation Generator

Parameters: INTEGER max_number, INTEGER permutation_length, INTEGER ARRAY permut
Returns: Nothing – stores permutation in the permut array

max_number – The number of different values that may occur in the permutation. eg chromosome length
permutation_length – The number of random, unique values that are required
permut – Array which stores the resulting random numbers

This function generates unique random integers in the range: 0 to max_number-1. If permutation_length=max_number, then the permut array contains a true permutation. However, by setting permutation_length<max_number, a smaller quantity of random numbers, in the specified range, can be obtained. This is good for picking, for example, two unique genes in a chromosome.

DEFINE an ARRAY OF *max_number* INTEGERS (*initial* array)
STORE the numbers 0 to *max_number*-1 sequentially in the *initial* array

INTEGER *random_index*
FOR *i* ranged from 0 to *permutation_length* - 1
    SET *random_index* = random() MOD (*max_number* - *i*)
    SET *permut*[*i*] = initial[random_*index*]
    SET *initial*[*random_index*] = *initial*[*length* - 1 - *i*]
ENDFOR

1.2 Crossover Operators

All crossover operators have a cross function which accepts three pointers:

- p1 – pointer to parent1 chromosome
- p2 – pointer to parent2 chromosome
- c – pointer to a chromosome that is to be used to store the child

1.2.1 Order-based crossover

1.2.1.1 Introduction

Section 1: Generates an *index_mask* and an *id_mask*
Both masks are *n*-bit representations of type BITSTREAM, where *n* is the length of a chromosome. The BITSTREAM class uses an attached character array to store data and allows bitwise reading and writing operations on the mask.

The *index_mask* has high bits whether a gene is selected from parent2.
The bit-positions indexed by the gene_ids of these genes are set high in the *id_mask*.

The *index_mask* is generated by:

- setting all bits in the mask randomly
- If bit-*i* in the *index_mask* is high, then gene *i* in parent2 is selected.

The *id_mask* is generated by :

- scanning the *index_mask* for all high bits
- For high bit-*i*, look-up the gene_id of gene *i* in parent2 – set bit-(*gene_id*) in *id_mask* high

Section 2: Generate child chromosome
The child is provisionally a copy of parent1, except all genes with gene_ids flagged high in the *id_mask* are reordered according to their order in parent2.

1.2.1.2 Algorithm

```
SECTION 1
{
        SET length = length of parent1 chromosome
        SET mask_word_length = ceiling(length / 16)          // Number of 2-byte words required
        SET mask_char_length = mask_word_length*2            // Number of chars required

        DEFINE index_mask_array ARRAY OF mask_char_length UNSIGNED CHAR
        DEFINE index_mask BITSTREAM
        ATTACH index_mask_array to index_mask – call index_mask.attach_char_array function

        // Generate the index mask bit pattern
        FOR i ranged from 0 to mask_word_length-1
                APPEND random 16-bit number to index_mask – call index_mask.append_word
        ENDFOR

        CALL index_mask.reset_read_write()

        DEFINE id_mask_array ARRAY of mask_char_length UNSIGNED CHAR
        DEFINE id_mask BITSTREAM
        ATTACH id_mask_array to id_mask – call id_mask.attach_char_array function
        CALL id_mask.wipe_clean()

        // Generate the id mask by looking up the gene_ids of all genes flagged in the index mask bit pattern
        FOR i ranged from 0 to length-1
                IF (Bit-i of index_mask == 1)
                        SET t = gene_id of gene i of parent2
                        SET the Bit-t of id_mask to 1
                ENDIF
        ENDFOR

        CALL id_mask.reset_read_write()
}

SECTION 2
{
        SET index = -1
        FOR i ranged from 0 to length-1
                SET t = gene_id of gene i of parent1
                IF (Bit-t of id_mask == 0)                   // Gene_id not selected in parent 2
                        Copy gene i from parent1 to gene i of child
                ELSE                                         // Gene_id selected in parent 2
                        // Find the next gene in parent2 to be copied to the child
                        DO
                                SET result = bit value of next bit retrieved from index_mask
```

```
                WHILE (result = false)
                Copy gene index from parent2 to gene i of child
            ENDIF
        ENDFOR
}
```

Invalidate the fitness of the child

1.2.2 Position-based Crossover

1.2.2.1 Introduction

Section 1: Generates an *index_mask* and an *id_mask*
Both masks are generated as Section 1 of the order-based crossover algorithm.

Section 2: Generate child chromosome
All genes flagged by a high-bit in the *index_mask* are copied from parent2 to the corresponding genes in the child. Remaining genes are copied in the sequence found in parent1 to the child.

1.2.2.2 Algorithm

```
SECTION 1{     Same as Section 1 of order-based crossover algorithm }

SECTION 2
{
        // copy all chosen genes from parent2 to child
        // Examines all bits in the index_mask
        FOR i ranged from 0 to length-1
                IF (Bit-i of index_mask == 1)
                        Copy gene i from parent2 to gene i of child
                ENDIF
        ENDFOR

        // Genes not flagged in the index_mask are copied from parent1 to the next available
        // gene location in the child.
        SET ci = -1
        SET pi = -1
        DO
                // find next gene to take from parent1 – store index of next gene in pi
                DO
                        INCREMENT pi
                        SET t = gene_id of gene pi in parent1
                WHILE (pi<length AND Bit-t of id_mask==1)

                IF (pi==length)                         // all parent1 genes searched
                        BREAK out of LOOP

                // find next gene in child which can store the parent1 gene – store index of next gene in ci
                DO
                        INCREMENT ci
                        SET t = gene_id of gene ci in parent2
                WHILE (pi<length AND Bit-t of id_mask==1)

                Copy gene pi in parent1 to gene ci in child

        WHILE (ci<length-1)

        Invalidate the fitness of the child
}
```

1.2.3 Edge-recombination Crossover

1.2.3.1 Introduction

Two classes are required in addition to the crossover algorithm:

allele_map class: *Stores up to 4 edge references*

Functions:

- add_edge : adds a specified edge to the allele_map
- remove_edge : removes a specified edge from the allele_map
- get_edge_count : returns the number of edges stored (maximum of 4)
- clear : sets number of edge stored to zero

edge_map class: *Stores an array of allele_maps. Number of elements is equal to number of genes in a chromosome.*

Functions:

- add_to_edge_map : provided with an array of integers forming a valid permutation, this adds all edges to the edge_map
- remove_allele_from_map : provided with a single integer, *i* – all edges *i* in all allele_maps are deleted.
- choose_allele_with_fewest_edges : chooses the next allele to be placed in the child
- get_map_length : returns the number of allele_maps stored
- reset_all : clears all allele_map objects

1.2.3.2 Algorithm

Reset the edge_map
Add permutations of both parent1 and parent2 to the edge_map
SET *picked_id* = gene_id of the zeroth gene of either parent1 or parent2 (randomly determined)

DEFINE child_permutation ARRAY OF chromosome_length INTEGER

```
FOR all genes i in the chromosome
        SET child_permutation[i] = picked_id
        IF (i==last_gene)
                BREAK out of FOR LOOP

        REMOVE picked_id edges from edge_map
                ( map.remove_allele_from_map(picked_id) )
        CHOOSE next allele to place in child – store result in picked_id
                (picked_id=map.choose_allele_with_fewest_edges(picked_id) )
ENDFOR
```

SET the child chromosome to have a permutation identical to *child_permutation*
Invalidate the fitness of the child

1.3 Mutation Operators

All standard mutation operators have a mutate function which accepts two pointers:

- p – pointer to the parent chromosome
- c – pointer to a chromosome that is to be used to store the child

1.3.1 Inversion Mutation

```
SET length = length of parent chromosome
DEFINE picked_indexes AS ARRAY OF 2 INTEGERS
CALL permutation(length, 2, picked_indexes)
SWAP picked_indexes elements if NOT in ASCENDING order

FOR all genes i from START of chromosome upto, but not including, FIRST picked_index gene
        COPY genes i from parent to child
ENDFOR

FOR all genes i from FIRST picked_index gene upto, and including, SECOND picked_index gene
        COPY genes i from parent to child in reverse order
ENDFOR

FOR all genes i from the gene after SECOND picked_index gene to the END of the chromosome
        COPY genes i from parent to child
ENDFOR

Invalidate the fitness of the child
```

1.3.2 Shunt Mutation

```
SET length = length of parent chromosome
DEFINE picked_indexes AS ARRAY OF 2 INTEGERS
DO
        SET permut_indexes[0] = random() MOD length
        SET permut_indexes[1] = random() MOD length
        SWAP picked_indexes elements if NOT in ASCENDING order
        SET shunt_length = difference between the two picked_indexes elements
WHILE (shunt_length==length-1 OR shunt_length==0)

DO
        SET position = random() % (length-shunt_length)          // insertion position to shunt to
WHILE (position==picked_indexes[0])                               // ensure 0 position move not possible

SET i=0
// copy genes from start of chromosome upto either beginning of shunt length or shunt insertion position
WHILE (i<position AND i<picked_indexes[0])
        COPY genes i from parent to child
        INCREMENT i
ENDWHILE

SET j=picked_indexes[0]
SET ci = i

IF (i==picked_indexes[0] AND i<position)          // if reached shunt start before insert position found
        ADD shunt_length+1 to i
        // continue copying genes - skip shunt length - until shunt insertion position got to
        WHILE (ci<position AND i<length)
                Copy gene i from parent to gene ci of child
                INCREMENT i and ci
        ENDWHILE
```

```
    WHILE (j<=picked_indexes[1] AND ci<length)          // copy shunt genes into child
        Copy gene j from parent to gene ci of child
        INCREMENT j and ci
    ENDWHILE

    WHILE (ci<length)                                   // copy rest of parent genes into child
        Copy gene i from parent to gene ci of child
        INCREMENT i and ci
    ENDWHILE
ELSE                                // reached insert position before shunt start reached
    WHILE (j<=picked_indexes[1] AND ci<length)          // copy shunt genes into child
        Copy gene j from parent to gene ci of child
        INCREMENT j and ci
    ENDWHILE

    // copy remaining genes before the shunt genes upto start of shunt
    WHILE (i<picked_indexes[0])
        Copy gene i from parent to gene ci of child
        INCREMENT i and ci
    ENDWHILE

    ADD shunt_length+1 to i         // continue copying genes, skipping the shunt length
    WHILE (ci<length)                       // copy rest of parent genes after shunt into child
        Copy gene i from parent to gene ci of child
        INCREMENT i and ci
    ENDWHILE
ENDIF

Invalidate the fitness of the child
```

2 T-test Code (supplied by Mr D.W. Corne)

```
/*

   Compile with:                          gcc ttest.c -o tt -lm
   or, if that doesn't work, with:   /packages/gnu/bin/gcc ttest.c -o tt -lm



    Simple statistics -- the t test.

  Say you run a GA 8 times on a minimisation problem adnd you get these results
  (best from each run):

  15 8.5 7 19 10 12 11 8.4

  Say you run hillcliming 10 times on the same problem and you get these
results:

  14.1 12.1 24 26 19.3 11.4 11.1 14.1 6.2 11.8


   Which seems to be the better algorithm ?

  If you put these results into separate files, called say `ga' and `hc', then
you
  can use this program to run a `t test'.   Data points in the files must be
separated
by whitespace (new lines, tabs,   spaces, or any mixture). On this example,
running:

   ./tt ga hc

  gives the following output:

------------------------------------------------------------
  file ga:   8 points, mean 11.3625, variance 15.7084, standard dev 3.96338
  file hc:   10 points, mean 15.01, variance 38.4854, standard dev 6.20366
  t value :    -1.43988

  t relates to confidence that first file results are better than
  second file results. If minimising, t may be negative. Just multiply
  it by -1

  In this case, with a total of 18 data points, the t value you need in order to
have
       90% confidence in the ga results being better than the hc results is >=
1.337
  For 95% confidence, the t value must be >= 1.746
  For 99% confidence, the t value must be >= 2.583
------------------------------------------------------------


------------------------------------------------------------
   So, we can have 90% confidence that the GA is better on the basis of these
results.
   You can have any number of data points, but at least two in each file. With
small numbers
  of points, it is of course much harder to get confident decisions. Eg: running the test
  on just the first three points in each file gives:
------------------------------------------------------------
file ga:   3 points, mean 10.1667, variance 18.0833, standard dev 4.25245
file hc:   3 points, mean 16.7333, variance 40.6033, standard dev 6.37207
t value :    -1.48469
t relates to confidence that first file results are better than
second file results. If minimising, t may be negative. Just multiply
it by -1
  In this case, with a total of 6 data points, the t value you need in order to
have
  90% confidence in the ga results being better than the hc results is >= 1.533
  For 95% confidence, the t value must be >= 2.132
  For 99% confidence, the t value must be >= 3.747
```

```
-----------------------------------------------------------
*/
/* **************************************************/
/*                                                  */
/*   compile with:   gcc ttest.c -o tt -lm          */
/*                                                  */
/*   usage:     ./tt <file1> <file2>                */
/*                                                  */
/****************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#define TRUE 1
#define FALSE 0
#define MAXDATA 5000
double getdouble(FILE *, double *, int);
double a[MAXDATA],   b[MAXDATA];
double get_tvalue(int , int );
FILE     *ap, *bp;
int      an,   bn;
int n;
extern double sqrt(double);
int get_input(FILE *f, double *arr);

void main(int argc, char **argv)
{
   int i;
   double am, bm, asv, bsv, sv, s, t, x, lsasv, lsbsv, ls;
     /* get the data in */
/*
                  char mark[10]="ga.txt";
                  char baz[10]="hc.txt";
                  argv[1]=mark;
                  argv[2]=baz;
*/
     ap = fopen(argv[1], "r");
     bp = fopen(argv[2], "r");
     get_input(ap, a);   an = n; fclose(ap);
     get_input(bp, b);   bn = n; fclose(bp);
   /* work out the means */
  am=0; for(i=0;i<an;i++) am+=a[i];   am /= (double)(an);
  bm=0; for(i=0;i<bn;i++) bm+=b[i];   bm /= (double)(bn);

   /* work out dual sample variance */
   asv = 0; for(i=0;i<an;i++) asv += ((a[i]-am)*(a[i]-am));
   lsasv = asv/((double)(an)-1.0);

   lsasv = asv/((double)(an)-1.0);

   bsv = 0; for(i=0;i<bn;i++) bsv += ((b[i]-bm)*(b[i]-bm));
   lsbsv = bsv/((double)(bn)-1.0);
   sv = (asv + bsv) / (double)(an + bn - 2);

   s = sqrt(sv);

   t = (am - bm) / ( s * sqrt( (1.0/an)   + (1.0/bn) ) ) ;

   ls = (am - bm) / ( sqrt ( (lsasv/an) + (lsbsv/bn)));
     printf(" file %s:   %d points, mean %g, variance %g, standard dev %g\n",
                        argv[1], an, am, asv/(double)(an-1),
sqrt(asv/(double)(an-1)));
     printf(" file %s:   %d points, mean %g, variance %g, standard dev %g\n",
                                    argv[2], bn, bm, bsv/(double)(bn-1),
sqrt(bsv/(double)(bn-1)));

     printf(" t value :     %g \n\n", t);
     printf(" t relates to confidence that first file results are better than \n");
     printf(" second file results. If minimising, t may be negative. Just multiply\n");
     printf(" it by -1 \n\n");
printf("   In this case, with a total of %d data points, the t value you need in order to have\n",an+bn);
printf("          90%% confidence in the %s results being better than the %s results is >= %g \n",
         argv[1], argv[2], get_tvalue((an+bn)-2, 90));
printf("     For 95%% confidence, the t value must be >= %g \n", get_tvalue((an+bn)-2,95));
printf("     For 99%% confidence, the t value must be >= %g \n", get_tvalue((an+bn)-2,99));
```

```
if ((an>=30) && (bn>=30)) printf("\n\n Large sample statistic is %g\n", ls);
if ((an>=30) && (bn>=30)) printf("\n\n Large sample statistic is %g\n", ls);
}
double get_tvalue(int npoints, int conf)
{
   double r;
   switch(npoints){
   case 1: if(conf==90) r = 3.078; else if(conf==95) r = 6.314 ; else r =
31.821; break;
   case 2: if(conf==90) r = 1.886; else if(conf==95) r = 2.920 ; else r = 6.965; break;
   case 3: if(conf==90) r = 1.638; else if(conf==95) r = 2.353 ; else r = 4.541; break;
   case 4: if(conf==90) r = 1.533; else if(conf==95) r = 2.132 ; else r = 3.747; break;
   case 5: if(conf==90) r = 1.476; else if(conf==95) r = 2.015 ; else r = 3.365; break;
   case 6: if(conf==90) r = 1.440; else if(conf==95) r = 1.943 ; else r = 3.143; break;
   case 7: if(conf==90) r = 1.415; else if(conf==95) r = 1.895 ; else r = 2.998; break;
   case 8: if(conf==90) r = 1.397; else if(conf==95) r = 1.860 ; else r = 2.896; break;
   case 9: if(conf==90) r = 1.383; else if(conf==95) r = 1.833 ; else r = 2.821; break;
   case 10: if(conf==90) r = 1.372; else if(conf==95) r = 1.812 ; else r =
2.764; break;
   case 11: if(conf==90) r = 1.363; else if(conf==95) r = 1.796 ; else r =
2.718; break;
   case 12: if(conf==90) r = 1.356; else if(conf==95) r = 1.782 ; else r =
2.681; break;
   case 13: if(conf==90) r = 1.350; else if(conf==95) r = 1.771 ; else r =
2.650; break;
   case 14: if(conf==90) r = 1.345; else if(conf==95) r = 1.761 ; else r =
2.624; break;
   case 15: if(conf==90) r = 1.341; else if(conf==95) r = 1.753 ; else r =
2.602; break;
   case 16: if(conf==90) r = 1.337; else if(conf==95) r = 1.746 ; else r =
2.583; break;
   case 17: if(conf==90) r = 1.333; else if(conf==95) r = 1.740 ; else r =
2.567; break;
   case 18: if(conf==90) r = 1.330; else if(conf==95) r = 1.734 ; else r =
2.552; break;
   case 19: if(conf==90) r = 1.328; else if(conf==95) r = 1.729 ; else r =
2.539; break;
   case 20: if(conf==90) r = 1.325; else if(conf==95) r = 1.725 ; else r =
2.528; break;
   case 21: if(conf==90) r = 1.323; else if(conf==95) r = 1.721 ; else r =
2.518; break;
   case 22: if(conf==90) r = 1.321; else if(conf==95) r = 1.717 ; else r =
2.508; break;
   case 23: if(conf==90) r = 1.319; else if(conf==95) r = 1.714 ; else r =
2.500; break;
   case 24: if(conf==90) r = 1.318; else if(conf==95) r = 1.711 ; else r =
2.492; break;
   case 25: if(conf==90) r = 1.316; else if(conf==95) r = 1.708 ; else r =
2.485; break;
   case 26: if(conf==90) r = 1.315; else if(conf==95) r = 1.706 ; else r =
2.479; break;
   case 27: if(conf==90) r = 1.314; else if(conf==95) r = 1.703 ; else r =
2.473; break;
   case 28: if(conf==90) r = 1.313; else if(conf==95) r = 1.701 ; else r =
2.467; break;
   case 29: if(conf==90) r = 1.311; else if(conf==95) r = 1.699 ; else r =
2.462; break;
   default: if(conf==90) r = 1.282; else if(conf==95) r = 1.645 ; else r =
2.326; break;
   }
return(r);
}
int get_input(FILE *f, double *arr)
{
   double v;
   n = 0;
   while(getdouble(f, &v, FALSE)!=0) arr[n++]=v;
                return 0;
}
/*
 *
 * Peter Ross' getint etc functions.
 */
/******************************************************************/
/* Get the next number from the input: put it in the location   */
/* addressed by second argument. This function returns 0 on      */
```

```
/* EOF. If stopateol is true, it returns -1 when it hits \n      */
/* (after which some other procedure has to read past the \n), */
/* otherwise it continues looking for the next number.           */
/* A number has an optional sign, perhaps followed by digits,   */
/* perhaps followed by a decimal point, perhaps followed by      */
/* more digits. There must be a digit somewhere for it to count*/
/* as a number. So it would read any of:                         */
/*   -.5                                                         */
/*   -0.5                                                        */
/*   -.5.7                                                       */
/* as minus-a-half. In the last case, it would read .7 next      */
/* time around.                                                  */
/*     There doesn't seem to be a neat and reliable way to do    */
/* all this, including stopateol, using scanf?                   */
/****************************************************************/
double
getdouble (FILE *file, double *valaddr, int stopateol)
{
   int c;
   int found = FALSE, indecimal = FALSE;
   int sign = +1;
   double n = 0.0, p = 1.0;
   /* First find what looks like start of a number - the first digit. */
   /* And note any sign and whether we just passed a decimal point.    */
   do {
      c = fgetc(file);
      if(c == EOF) return (0);
      else if(stopateol && c =='\n') return(-1);
      else if(c == '+' || c == '-') {
         sign = (c == '+')? +1 : -1;
         c = fgetc(file);
         if(c == EOF) return (0);
         else if(stopateol && c =='\n') return(-1);
      }
      if(c == '.') {
         indecimal = TRUE;
         c = fgetc(file);
         if(c == EOF) return (0);
         else if(stopateol && c =='\n') return(-1);
      }
      if(c >= '0' && c <= '9') {
         found = TRUE;
      } else {
         sign = +1;
         indecimal =   FALSE;
      }
   } while(!found);
   /* Now we've got digit(s) ... */
   do {
      n = 10.0*n + c - '0';
      p = 10.0*p;
      c = fgetc(file);
      if((c < '0') || (c > '9')) {
         found = FALSE;
         /* We've run out. If we already saw a decimal point, return now */
         if(indecimal) {
            if(c != EOF) ungetc(c,file);
            *valaddr =   sign * n/p;
            return(1);
         } else p = 1.0;
      }
   } while(found);
   /* We ran out and we didn't see a decimal point, so is this a decimal? */
   if(c != '.') {
      /* No, give it back to caller */
      if(c != EOF) ungetc(c,file);
      *valaddr = sign * n;
      return(1);
   } else {
      /* It is. Step past it, carry on hoping for more digits */
      c = fgetc(file);
      while(c >= '0' && c <= '9') {
         n = 10.0*n + c - '0';
         p = p*10.0;
         c =   fgetc(file);
      }
```

```
    /* We've run out of digits but we have a number to give */
    if(c != EOF) ungetc(c,file);
    *valaddr = sign * n/p;
    return(1);
  }
}
/* Use getdouble() above but convert result to int. */
int
getint (FILE *f, int *valaddr, int stopateol)
{
  int r;
  double x;
  r = getdouble(f,&x,stopateol);
  *valaddr = (int)x;
  return(r);
}
/* Use getdouble above but convert result to long. */
int
getlong (FILE *f, long *valaddr, int stopateol)
{
  int r;
  double x;
  r = getdouble(f,&x,stopateol);
  *valaddr = (long)x;
  return(r);
}
```