Challenges in Computer Science

Assignment 4 - genetic search algorithm

Rick van der Zwet
<hvdzwet@liacs.nl>

LIACS
Leiden Universiteit
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

May 30, 2007

1 Introduction

The assignment given out during the seminar of Natural Computing Group of LIACS[1] will have the following context:

> You are required to implement an Evolutionary Algorithm for tacking the low-autocorrelation problem. Given your implementation, run your algorithm on strings of the lengths given the table, and report your results. A MATLAB code for the objective function is given to you in the following location: `http://www.liacs.nl/home/oshir/code/merit.m` (Backup at Appendix 7.1)
> Its documentation: `http://www.liacs.nl/home/oshir/code/autocorr.pdf`

2 Problem

The given problem, the so-called *low-autocorrelation problem of binary sequences* is subject to actual research and is of big interest for industrial applications, e.g. communications and electrical engineering. Its description follows.
Feasible Solutions: Binary Sequences $\overrightarrow{y} \in \{-1, +1\}^n$
Objective Function:

$$f(\overrightarrow{y}) = \frac{n^2}{2 \cdot E\overrightarrow{y}} \longrightarrow maximization \tag{1}$$

[1] http://natcomp.liacs.nl/

s.t.

$$E(\overrightarrow{y}) = \sum_{k=1}^{n-1} \left(\sum_{i=1}^{n-k} y_i \cdot y_{i+k} \right)^2 \tag{2}$$

Find a way using genetic algorithm to get the optional solutions

3 Theory

A genetic algorithm will use 'evolution' to get the best possible results. It will normally be executed in the following order

1. Generate N number of random parents
2. Determine best parents
3. start generating offspring
4. Determine best offspring
5. Combine the best parents and offspring to number of N new parents
6. Check if end condition is matched else go-to 3 again

There are a few well known mutations to generate new nodes

- Crossover: Combine 2 parts of 2 nodes to a new node. Example (using crossover point 3):

```
A = 0 1 0 | 1 1 0
B = 1 1 1 | 0 0 0
A' = 0 1 0  0 0 0
B' = 1 1 1  1 1 0
```

- Mutation: Change a few values in a node. Example using mutation point 3.

```
A = 0 1 0 1 1 0
A' = 0 1 1 1 1 0
```

3.1 Algorithm

I have introduced one new mutation way, cross-flip. It will flip all values starting from a certain point. Example, using point 3

```
A = 0 1 0 1 1 0
A' = 0 1 1 0 0 1
```

This will be my algorithm

```
01: Generate N random nodes on stack @A
02: for n in 0 to size A
03:     exit loop if run for $round_size times
04:     check mutation(n) children are better and unshift to stack @A
05:     check cross-flip(n) children are better and unshift to stack @A
06:     check crossover(n) children are better and unshift to stack @A
07:     if n on end of the stack then remove all but A[0:5], add 5 random
08:        strings and do crossover on all variants.
09:     if improvement then set n to beginning
10: add best result (@A[0]) to the @W array but keep the array sorted and
    not longer then 10
11: check if we have been here $maxround times, exit and give result
12: check if we have been here such that ($times % $winnercompare) == 0
       start generating special results  by combining the best results of @W
       into @A and go-to 02;
13: start putting 75 random strings into @A and go-to 02:
```

4 Implementation

I wrote my implementation in Perl code, with a stack implementation and used as much references as possible.

5 Experiments

I ran all tests 5 times and will display the best merit factor and corresponding string. Vectors are written in run-length notation: each figure indicates the number of consecutive elements with the same sign.

Table 1: Experiment results

Size	m-factor	vector
3	4.5	2,1
4	4.0	1,3
5	6.5	3,1,1
10	10.0	3,3,1,2,1,1
15	7.5	3,3,1,3,1,2,1,1
20	7.7	5,1,1,3,1,1,2,3,2,1
30	6.7	4,2,5,1,2,3,1,2,1,1,1,1,3,1,1,1
50	5.0	1,1,1,1,1,1,1,2,2,3,4,1,7,2,2,2,1,2,1,1,5,4,1,1,2
75	4.7	1,1,4,5,2,1,1,2,4,4,1,3,3,2,1,1,1,1,1,2,2,3,1,4,2,2,1,3,1,1,2,1,4,1,3,1,2

6 Conclusion

The algorithm works pretty well on the small size numbers. The big numbers (n ¿ 20) are not scoring good, cause the script/program is causing time problems.

Writing the implementation in Perl is too slow. It's not generation the bigger rates at a decent speed[2], cause the Perl code is not optional yet. When looking at the profiler output (Appendix 7.3), the merit function could be optimised to get a higher throughput.

7 Appendix

7.1 merit.m

```
%An objective function for the low-autocorrelation problem.
%Author: Ofer M. Shir, 2004; oshir@liacs.nl.
%-------------------------------------------------------------------------
function [f] = merit (pop)
% Given a population of binary sequences, this function calculates
% the merit function according to the formula specified in the exercise
% description. The input pop is the given matrix.
% The output f is the merit factor calculated (row vector).

n = size(pop,1)
m = size(pop,2)
E = zeros(1,m)
%Calculated efficiently in a matrix-notation; auxilary matrices - Y1,Y2
%- are initialized in every iteration. They are shifted form of the
%original y vectors. The diaganol of the dot-squared Y2*Y1 matrix is
%exactly the inner sum of merit function.
for k=1:n-1
    Y1=pop(1:n-k,:)
    Y2=pop(k+1:n,:)'
    E=E+((diag(Y2*Y1)).^2)'
end

%The output:
f = (n*n*ones(1,m))./(2*E)
```

7.2 merit.pl

```
#!/usr/bin/env perl
```

[2] one round of n = 100 takes at a dual core Pentium 4 3Ghz and 2GB RAM about 2 minutes

```
use strict;

our $debug = 0;
our %taken;

our $length = 0;
our $round_size = 100;
our $maxstack =  150;
our $working_mfactor = 0;
our $best_mfactor = 0;
our $best_vector;
our $refill = 0;
our $maxrounds = 20;
our @winners;
our $winnercompare = 10;

our $total = 0;

sub message {
    our $debug;
    my $level = shift;
    my $message = shift;
    if ( $level => $debug ) {
        print "DEBUG [$level]: $message\n";
    }
}

sub merit {
    my $y = shift;
    our $length;
    my $e = 0;
    my $f;
    my $t;

    foreach my $k (1 .. ($length - 1)) {
        $t = 0;
        for my $i (0 .. ($length - 1 - $k)) {
            $t += $$y[$i] * $$y[$i+$k];
        }
        $e += $t ** 2;
        }

    $f = ($length ** 2) / ( 2 * $e);
```

```
    printer('DEBUG merit', $f, $y) if $debug > 5;
    return($f);
}

sub randarray {
    our $debug;
    my @output = ();
    my ($i,$v);
    message(1,"New randarray of length $length");
    foreach $i ( 0 .. ($length - 1)) {
        $v = int(rand() + 0.5);
        $output[$i] = ( $v ) ? 1 : -1;
    }
    return(\@output);
}

sub flip {
  my $rref = shift;
  my $number = shift;
  my $v = $$rref[$number];
  $v = ($v == 1) ? -1 : 1;
  $$rref[$number] = $v;
}

sub try_merit {
    our @stack;
    our %taken;
    our $best_mfactor;
    our $best_vector;
    our $working_mfactor;

    my $try_array = shift;
    my $new_mfactor;
    my $result = 0;
    my $array2string = join('',@{$try_array});

    if ( not exists ($taken{$array2string}) ) {
        $new_mfactor = merit($try_array);
        $taken{$array2string} = $new_mfactor;

        if ($new_mfactor > $best_mfactor) {
           $best_mfactor = $new_mfactor;
           $best_vector = $try_array;
```

```
            $result = 1;
            unshift(@stack,$try_array);
        }
    }
    if ( $debug == 5 ) {
        print "==== " . scalar(@stack) . "====\n";
        print join("\n",@stack) . "\n";
    }
    return($result);
}

sub combine {
    our $length;
    our $debug;
    message(1,"Working on combine");

    my $result = 0;
    my $loop;
    my $input = shift;
    my $input2 = shift;
    my @result_l;
    my @result_r;


    my $middle = int($length / 2);
    @result_l = (@{$input}[0 .. $middle],
                 @{$input2}[ $middle + 1 ..  ($length-1) ]);
    @result_r = (@{$input2}[0 .. $middle],
                 @{$input}[ $middle + 1 ..  ($length-1) ]);

    if ($debug > 3) {
        print "Orig 1: ";
        print @{$input};
        print "\n";
        print "Orig 2: ";
        print @{$input2};
        print "\n";
        print "Combine l: ";
        print @result_l;
        print "\n";
        print "Combine r: ";
        print @result_r;
        print "\n";
```

```
    }
    message(1,"DEBUG: Done on combine");
    return(\@result_l,\@result_r);
}

sub crossover {
    my $input = shift;
    my $rev_array = shift;
    my $result = 0;
    my ($try_array_l,$try_array_r)  = combine($input,$rev_array);

    if ( try_merit($try_array_l) ) {
        $result = 1;
    }

    if ( try_merit($try_array_r) ) {
        $result = 1;
    }
    return($result);
}


sub mutations {
    our $length;

    my @try_array;
    my $result = 0;
    my @test_array = @_;

    message(1,"Working on mutations");

    foreach my $i ( 0 .. ($length - 1) ) {
        my @try_array = ();
        @try_array = @test_array;
        flip(\@try_array, $i);

        if ( try_merit(\@try_array) ) {
            $result = 1;
        }
    }
    message(1,"Done on mutations");
    return($result) ;
}
```

```
sub crossovers {
    our $length;

    my @try_array;
    my $result = 0;
    my @test_array = @_;

    my $i;
    my $j;

    message(1,"Working on crossovers");
    foreach $i ( 0 .. ($length - 1) ) {
        my @try_array = ();
        @try_array = @test_array;
        foreach $j ( $i .. ($length - 1) ) {
            flip(\@try_array, $j);
        }

        if ( try_merit(\@try_array) ) {
            $result = 1;
        }
    }
    message(1,"Done on crossovers");
    return($result);
}
```

```
sub randmutate {
    our $length;
    my $switchtimes = shift;
    my $switchnumbers = shift;
    my @test_array = @_;
    my $result = 0;
    my @try_array;
    my $i;
    my $j;
    my $n;
    my @flipped;
```

```
    message(1,"Working on randmutate");

    foreach $i ( 0 .. $switchtimes ) {
        my @try_array = ();
        @try_array = @test_array;
        @flipped = ();
        foreach $j ( 0 .. $switchnumbers ) {
            $n = int(rand($length - 1));
            if ( not exists $flipped[$n] ) {
                flip(\@try_array,$n);
                $flipped[$n] = 1;
            }
        }

        if ( try_merit(\@try_array) ) {
            $result = 1;
        }
    }
    return($result);
}

sub printer {
    my $message = shift;
    my $mfactor = shift;
    my $input = shift;
    return("Message : $message \n" .
           "Vector  : " . join(',',array2string($input) . "\n" .
           "mfactor : $mfactor \n" .
           "--\n");
}



our @stack = ();
our $best_mfactor = 0;
our $best_vector;

sub array2string {
    my $input = shift;
    my @result = ();
    my $n = 1;
    foreach my $i (1 .. $#$input) {
        if ($$input[$i] != $$input[$i - 1]) {
```

```
            push(@result,$n);
            $n = 1;
        }
        else {
            $n++;
        }
    }
    push(@result,$n);
    return(@result);
}

sub string2array {
    my @output = ();
    my $input = shift;
    $input = split(//,$input);
    my $c = 0;
    my $v = 1;

    my ($n,$i);

    for $n ( 0 .. length($input) - 1) {
        for $i ( 1 .. @{$input}[$n] ) {
            $output[$c] = $v;
            $c++;
        }
        $v = ( $v == 1) ? -1 : 1;
    }

    return(@output);
}


#
# Vul de stack met een random arrays
sub init {
    our $length = shift;
    our @stack;
    our $best_mfactor;

    my ($n,$y);

    @stack = ();
    $best_mfactor = 0;
```

```
    foreach $n ( 1 .. 75 ) {
        $y = randarray();
        push(@stack,$y);
    }

}

sub reinit {
    our $refill;
    our $winnercompare;
    our $length;
    our @winners;
    our %taken;
    our @stack;
    our $best_vector;
    our $best_mfactor;

    my $maxwinners = 5;

    my ($result_l,$result_r);
    my ($i,$j);
    my ($n,$y);
    my ($r);

    %taken = ();
    @stack = ();

    $refill++;
    #push best array on good place
    $i = merit($best_vector);
    print printer("Round best vector",$i,$best_vector);
    my $ws = $#winners - 1;
    my $done = 0;
    foreach $n ( 0 .. $ws )  {
        $j = merit($winners[$n]);
        if ( $i > $j ) {
            foreach $r ($ws .. ($n + 1)) {
                $winners[$r] = $winners[$r-1];
            }
            $winners[$n] = $best_vector;
            $done = 1;
            last;
```

```
        }
    }
    if ( ($done == 0) && ($ws < $maxwinners) ) {
        push(@winners,$best_vector);
    }
    print printer("Current best vector",merit($winners[0]),$winners[0]);

    if ( $refill == $winnercompare ) {
        push(@stack,@winners);
        $refill = 0;
        $r = scalar(@stack) - 1;
        foreach $i ( 0 .. ($r - 1)) {
            foreach $j ( ($i) .. $r ) {
                ($result_l,$result_r) = combine($stack[$i],$stack[$j]);
                push(@stack,$result_l);
                push(@stack,$result_r);
            }
        }
        $best_mfactor = merit($winners[0]);
    }
    else {
        foreach $n ( 1 .. 75 ) {
            $y = randarray();
            push(@stack,$y);
        }
        $best_mfactor = 0;
    }
}

sub main {
    our $working_mfactor;
    our $round_number;
    our $maxstack;
    our $maxrounds;

    my $rounds = 0;
    my $number = 0;
    my $improvement = 0;
    my $result;
    my $stacksize;
    my ($i,$j,$r,$y);

    while ( 'true' )
```

```
    {
        $total++;

        $number = ($improvement) ? 0 : $number + 1;
        print "Number: $number\n" if $debug;

        $working_mfactor = merit(@stack[$number]);
        if ( $debug > 1 ) {
            message(1,
            printer("Working vector",$working_mfactor,@stack[$number]));
        }

        message(1,"\n" . $number . " of " . $stacksize );
        $improvement = 0;

        $result = try_merit(@stack[$number]);
        if ($result > 0 ) {
            message(1,
            printer("better array", $best_mfactor,@stack[$number]));
            $improvement = 1;
        }

        my @rev = reverse(@{$stack[$number]});
        $result = crossover($stack[$number],\@rev);
        if ($result > 0 ) {
            message(1,
            printer("better combine", $best_mfactor,@stack[$number]));
            $improvement = 1;
        }

        $result = mutations(@{$stack[$number]});
        if ($result > 0 ) {
            message(1,
            printer("better mutation", $best_mfactor,@stack[$number]));
            $improvement = 1;
        }

# Experimental 'crossover make'
        $result = crossovers(@{$stack[$number]});
        if ($result > 0 ) {
            message(1,
            printer("better crossovers", $best_mfactor,@stack[$number]));
            $improvement = 1;
```

```
        }

# Not very usefull, not value added
#        $result = randmutate($length, $length / 3,@{$stack[0]});
#        if ($result > 0 ) {
#            printer("better randmutate", $best_mfactor,@stack[0]);
#            next;
#        }

        if ( $improvement ) {
            print "Stack size: " . scalar(@stack) . "\n";
            print "best max mfactor " . $best_mfactor . "\n";
            print "total done " . $total . "\n";
            print "round " . $rounds;
            print "\n---\n";
        }

        if ( ($total % $round_size) == 0 ) {
            print "==============================\n";
            print "=== Completed Round $rounds ===\n";
            print "==============================\n";
            $rounds++;
            reinit();
            if ( $rounds == $maxrounds ) {
                last;
            }
            $r = <> if $debug > 3;
            $improvement = 1;
            print "==============================\n";
            print "=== New       Round $rounds ===\n";
            print "==============================\n";
            next;
        }

        $stacksize = scalar(@stack);
        if ( $stacksize > $maxstack ) {
            while ( scalar(@stack) >= 5 ) {
                pop(@stack);
            }
        }
        message(1,"$number of " . scalar(@stack) . "\n");;
        $stacksize = scalar(@stack);
        if ( (($number +2) > $stacksize) ) {
```

```
            message(1,"Start EVOLVING");
            $number = 0;
            while ( scalar(@stack) > 5 ) {
                pop(@stack);
            }

            foreach ( 5 .. 9 ) {
                $y = randarray();
                push(@stack,$y);
            }

            foreach $i ( 0 .. 4) {
                foreach $j ( 5 .. 9 ) {
                    my ($result_l,$result_r) = combine($stack[$i],$stack[$j]);
                    push(@stack,$result_l);
                    push(@stack,$result_r);
                }
            }
            message(1,"Done EVOLVING");
        }

        #pause during debugging
        $r = <> if $debug > 4;
    }
}

$debug = 0;
$round_size = 100;
$maxstack = 150;
$maxrounds = 20;
$winnercompare = 5;
init($ARGV[0]);
main();

print "\n\n\n";

print "================================\n";
print "=== Results                  ===\n";
print "================================\n";

$best_vector = $winners[0];
$best_mfactor = merit($best_vector);
print printer("Final", $best_mfactor,$best_vector);
```

```
print "Total parents : $total\n";
print "Total rounds  : $maxrounds\n";
print "Round parents : $round_size\n";
print "Winner compare: $winnercompare\n";
print "Max stack     : $winnercompare\n";
print "Vector size   : $length\n";
```

7.3 profiler output

```
Total Elapsed Time = 33.07120 Seconds
  User+System Time = 32.94120 Seconds
Exclusive Times
%Time ExclSec CumulS #Calls sec/call Csec/c  Name
 85.9   28.31 28.312  99474   0.0003 0.0003  main::merit
 6.00   1.975  1.975 990000   0.0000 0.0000  main::flip
 5.22   1.718 17.022   2000   0.0009 0.0085  main::crossovers
 3.26   1.073 15.030   2000   0.0005 0.0075  main::mutations
 2.84   0.935 28.587 126000   0.0000 0.0002  main::try_merit
 0.50   0.166  0.163   2529   0.0001 0.0001  main::combine
 0.25   0.083  0.164     20   0.0042 0.0082  main::reinit
 0.22   0.074 34.493      1   0.0736 34.492  main::main
 0.16   0.054  1.075   2000   0.0000 0.0005  main::crossover
 0.15   0.048  0.042   1365   0.0000 0.0000  main::randarray
 0.08   0.027  0.027  18532   0.0000 0.0000  main::message
 0.06   0.019  0.019    134   0.0001 0.0001  main::array2string
 0.03   0.009  0.028    134   0.0001 0.0002  main::printer
 0.00       - -0.000      1        -      -  strict::bits
 0.00       - -0.000      1        -      -  strict::import
```