The low-autocorrelation problem

Practical Assignments Natural Computing, 2009

Rick van der Zwet
<hvdzwet@liacs.nl>

19 december 2009

1 Introduction

The report is focused on 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 goes as follows.

Feasible Solutions: Binary Sequences $\vec{y} \in \{-1, +1\}^n$
Objective Function:

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

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

2 Problem description

Due to the huge 'exploding' possibilities it is not possible to walk trough the whole list of possibilities, so we need alternative approaches to tackle this problem. First we will try the *Monte Carlo Search algorithm [MCS]* and next try *Simulated Annealing [SA]*.

MCS is all about random sequence generation and trying to find a good solution, do a small adjustment on the solution and compare the new solution again.

SA takes a more and less the same approach, but it also accept small 'losses' from time-to-time. But as time passes it get less likely to accept 'bad solutions'.

Tabel 1: Best known values of low-autocorrelation problem

n	Best known f
20	7.6923
50	8.1699
100	8.6505
200	7.4738
222	7.0426

3 Statistics

Of course many people have run numerous computer hours on finding the best possible fitness as shown in table 1. The algorithms used to find those numbers are not found.

4 Approach

The *MCS* is implemented straight-forward from the Wikipedia page[1]. The mutation choosing is flipping one bit in the array.

For the *SA* implemented also comes straight from 'the book' [2], the process choosing for the cooling-down sequence is taken from the dutch Wikipedia page 'Simulated annealing', which is nice indicator when to choose something when the solution is worse (logically, better solutions will always be accepted). [3]: $p = e^{\frac{f(i)-f(j)}{c}}$

5 Implementation

The code is written in *Octave*[4] which is the open-source 'variant' of *MATLAB*©[5]. There are small minor differences between them, but all code is made compatible to to run on both systems. The code is to be found in Appendix 8.

As work is done remotely, the following commands are used:

[1] http://en.wikipedia.org/wiki/Monte_Carlo_method
[2] http://en.wikipedia.org/wiki/Simulated_annealing
[3] http://nl.wikipedia.org/wiki/Simulated_annealing
[4] http://www.gnu.org/software/octave/
[5] http://www.mathworks.com/products/matlab/

Tabel 2: Best known values of low-autocorrelation problem

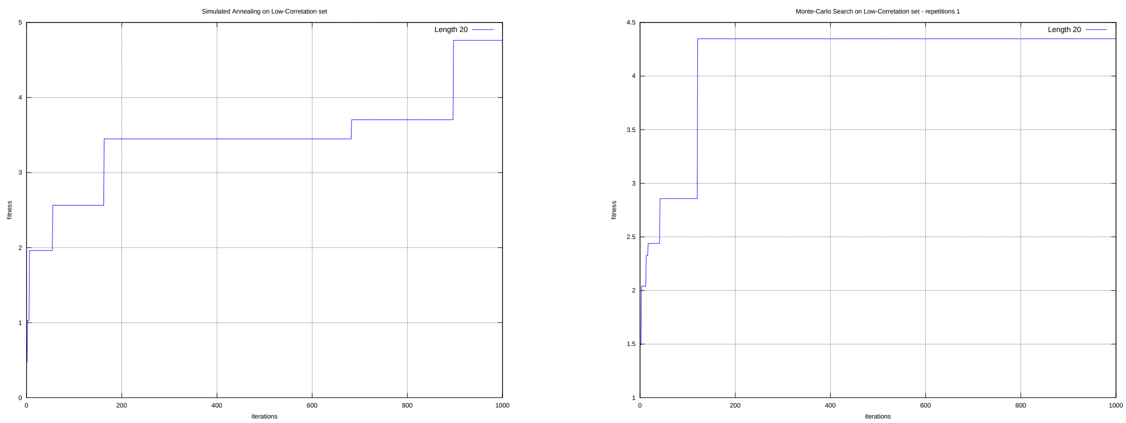
n	Best fitness		
	known	MCS	SA
20	7.6923	4.3478	4.7619
50	8.1699	2.4752	2.6882
100	8.6505	1.8342	1.7470
200	7.4738	1.8678	1.5733
222	7.0426	1.5657	1.4493

```
matlab-bin -nojvm -nodesktop -nosplash -nodisplay < %%PROGRAM%%

octave -q %%PROGRAM%%
```

6 Results

All experiments are run 5 times the best solution is chosen and will be resented at table 2. Iteration size is set to 1000. For $n = 20$ the best fitness history is shown at figure 1.



(a) SA

(b) MCS

Figuur 1: Fitness throughout the iterations

7 Conclusions

Looking at the graphs the fitness was still increasing so a larger iteration size would make the fitness a better result. Secondly the *SA* is preforming much worse on large number than the *MCS* one. It seems to the temperature function is not working as expected.

Both algorithms are preforming much worse then the best found solutions.

8 Appendix 1

```matlab
% An objective function for the low-autocorrelation problem.
% Author: Ofer M. Shir, 2004; oshir@liacs.nl.
%-----------------------------------------------------------------------------
function [f] = autocorrelation(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 diagonal 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);
```

```
% Simulated Annealing low-autocorrelation program
% BSDLicence
% Rick van der Zwet - 0433373 - <hvdzwet@liacs.nl>
% $Id: initseq.m 43 2009-12-17 22:18:48Z rick $

function s = initseq(n)
  % Generate a random column s={-1,1}^n
  s = rand(n,1);
  s = round(s);
  s = s - (s == 0);
end
```

```matlab
% Monte-Carlo Search Algoritm on low-autocorrelation program
% BSDLicence
% Rick van der Zwet - 0433373 - <hvdzwet@liacs.nl>
% $Id: mcs-call.m 46 2009-12-18 01:43:33Z rick $

% Brute-force result of length 20
% best_20 = [-1,1,1,-1,-1,-1,1,1,-1,1,-1,-1,-1,1,-1,1,1,1,1,1];

%% Basic variables
iterations = 1000;
length = 222;
repetitions = 1;

% Plot the stuff
[iteration_history, fitness_history] = mcs(length,iterations);

plot(iteration_history,fitness_history);
title(sprintf('Monte-Carlo Search on Low-Corretation set - repetitions %i',
      repetitions));
ylabel('fitness');
xlabel('iterations');
grid on;
legend(sprintf('Length %i',length));
print(sprintf('mcs-fitness-%f.eps',max(fitness_history)),'-depsc2');
max(fitness_history)

```

```
% Monte-Carlo Search Algoritm on low-autocorrelation program
% BSDLicence
% Rick van der Zwet - 0433373 - <hvdzwet@liacs.nl>
% $Id: mcs.m 46 2009-12-18 01:43:33Z rick $

% Brute-force result of length 20
% best_20 = [-1,1,1,-1,-1,-1,1,1,-1,1,-1,-1,-1,1,-1,1,1,1,1,1];
% autocorrelation(best_20);

function [iteration_history,fitness_history] = mcs(length, iterations)
  iteration_history = [];
  fitness_history = [];

  best_fitness = 0;
  for iteration = 1:iterations
    % Generate a random column s={-1,1}^n
    n = length;
    s = rand(n,1);
    s = round(s);
    s = s - (s == 0);

    % Find whether we are better than everything else
    fitness = autocorrelation(s);
    if (fitness > best_fitness)
      best_value = s;
      best_fitness = fitness;
    end
    iteration_history = [ iteration_history, iteration ];
    fitness_history = [ fitness_history, best_fitness ];
  end
end
```

```
% Simulated Annealing low-autocorrelation program
% BSDLicence
% Rick van der Zwet - 0433373 - <hvdzwet@liacs.nl>
% $Id: mutation.m 43 2009-12-17 22:18:48Z rick $

function new = mutation(old)
  loc = randint(1,length(old));
  old(loc) = old(loc) * -1;
  new = old;
end
```

```
% Simulated Annealing low-autocorrelation program
% BSDLicence
% Rick van der Zwet - 0433373 - <hvdzwet@liacs.nl>
% $Id: randint.m 43 2009-12-17 22:18:48Z rick $

function number = randint(low,high)
  number = round(rand() * (high - low)) + low;
end
```

```
% Simulated Annealing low-autocorrelation program
% BSDLicence
% Rick van der Zwet - 0433373 - <hvdzwet@liacs.nl>
% $Id: sa-call.m 46 2009-12-18 01:43:33Z rick $

%% Basic variables
iterations = 10000;
length = 222;


% Always use the same innitial values
s = initseq(length);

% Plot the stuff
[iteration_history, fitness_history] = sa(s,iterations);

plot(iteration_history,fitness_history);
title(sprintf('Simulated Annealing on Low-Corretation set'));
ylabel('fitness');
xlabel('iterations');
grid on;
legend(sprintf('Length %i',length));
print(sprintf('sa-fitness-%f.eps',max(fitness_history)),'-depsc2');
max(fitness_history)
```

```
% Simulated Annealing low-autocorrelation program
% BSDLicence
% Rick van der Zwet - 0433373 - <hvdzwet@liacs.nl>
% $Id: sa.m 44 2009-12-17 22:23:16Z rick $

function [iteration_history,fitness_history] = sa(seq, stopLimit)
  fitness = 0;
  temperature = stopLimit;

  iteration_history = [];
  fitness_history = [];

  for iteration = 1:stopLimit
    % Generate new mutation
    newseq = mutation(seq);

    new_fitness = autocorrelation(newseq);

    % Better is always accept
    if (new_fitness > fitness)
      fitness = new_fitness;
      % disp(rot90(newseq,-1));
      temperature = temperature - 10;
    else
      % Make the next 'move' less atractive
      temperature = temperature + 1;

      % Accept on an certain probability
      if (temperature < 1)
        break;
      else
        % XXX: Some more cleaver cooling would be great
        if( exp(1)^((fitness - new_fitness) / temperature) >
rand())
          seq = newseq;
         end
       end
    end
    iteration_history = [ iteration_history, iteration ];
    fitness_history = [ fitness_history, fitness ];
  end
```

```
end
```