Supporting Information

Assay design considerations for use of affinity aptamer amplification in ultra-sensitive protein assays using capillary electrophoresis.†

Kris P.F. Janssen,[a] Karel Knez,[a] Jeroen Pollet,[a] Scott J. Roberts,[b] Jan Schrooten[c] and Jeroen Lammertyn*[a]

Modified aptamer affinity measurement

Confirmation of primer modified aptamer affinity by is achieved by means of capillary electrophoresis. For this purpose, the aptamers were diluted to the desired concentration in freshly prepared Tris Glycine buffer, pH 8.4 (Sigma). Tris Glycine buffer was also used for the preparation of a stock solution of 2',7',Bis, (2, Carboxyethyl),5, (And,6), carboxyfluorescein (Sigma) at a concentration of 960 nM for use as an internal standard. A serial dilution of the target protein was also prepared. Following dilution, the APT1 solution was incubated at 75 °C for 20 minutes and subsequently cooled and stored on ice to prevent multimers or badly folded aptamers. Samples were prepared by mixing 10µl of each solution (internal standard, aptamer, protein) followed by a 10 minute incubation period before analysis. Separations were achieved using a Beckman P/ACE 2200 CE,LIF system (Beckman,Coulter, Fullerton, CA, USA) equipped with fused,silica capillaries (50 µm ID, 360 µm OD) with 40.2 cm total length, detector at 30 cm. Capillaries were pretreated by pumping methanol, 1 M NaOH, 0.1 M NaOH, deionized H2O, and finally Tris Glycine buffer, pH 8.4 through the capillary for 30 min each. Further experimental conditions are summarized in Table S1.

Action	Duration (s)
Rinse 20 psi – 0.1M NaOH	120
Rinse 20 psi – Buffer	120
Inject 1 psi 4 – Sample	4
Inject 0.1 psi – Water	1
Separate 15 kV	760
Rinse 10 psi – Buffer	60

Table S1.

Supporting Information

Between each electrophoretic separation the capillary was rinsed to remove any residual sample from the capillary walls. Laser Induced Fluorescence detection was achieved using the 488 nm line of a 3 mWArgon-ion laser (Beckman,Coulter) for excitation and emission was collected through a 520 ±10 nm band pass filter. Data were recorded and analyzed with Karat 32 v8.0 software (Beckman,Coulter) by integration of the electropherogram peaks corresponding to the different species. All areas were corrected for elution time and for variations in injection volume using an internal standard(carboxyfluorescein).

In order to fit a suitable expression (Expression E1) for the equilibrium binding constant (K_d) to the obtained data, concentrations of aptamer-protein complex were always expressed in terms of the fraction of total aptamer bound to protein. The fluorescence signal for the aptamer-protein complex was therefore normalized using the maximum signal obtained in samples saturated with protein, where all aptamer can be expected to be bound to the protein. This way, a value of the equilibrium binding constant can be derived by fitting equation E1 to the data.

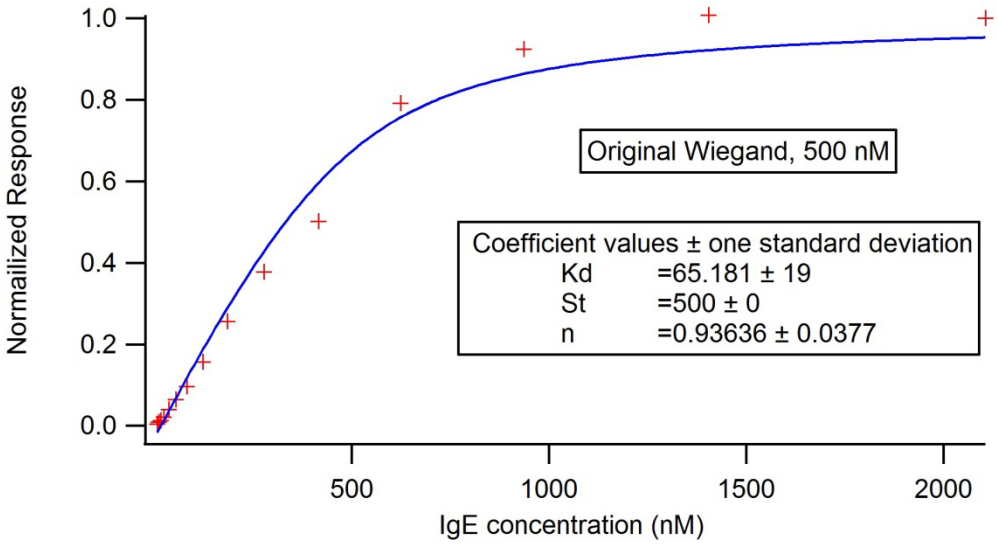

Figure S1: Binding curve of original, unmodified Wiegandaptamer. The aptamer concentration was fixed at 500nM and IgE concentrations ranged from 0 to 2106 nM of IgE.

Supporting Information

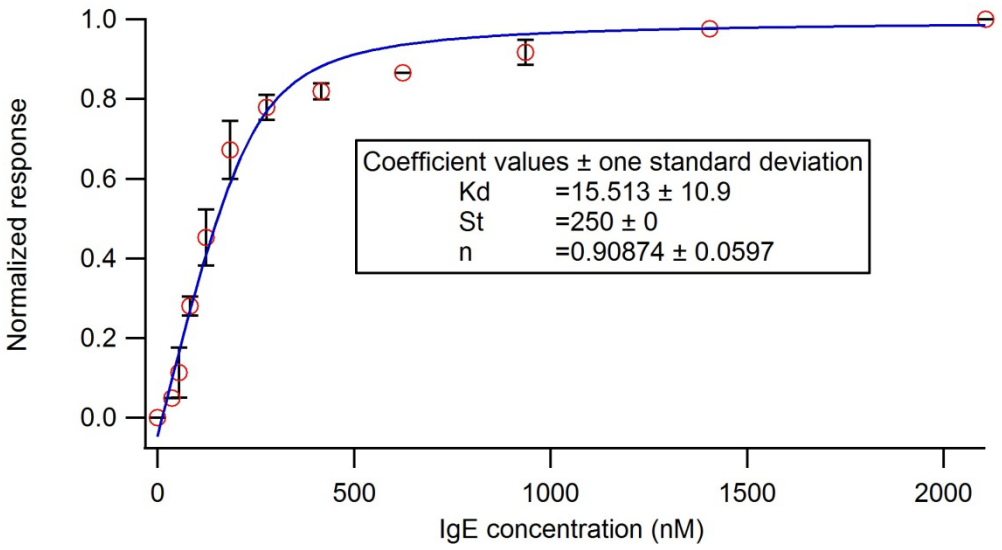

Figure S2: Binding curve of modified Wiegandaptamer, using primer set 4 (table 1, main manuscript). The aptamer concentration was fixed at 250 nM and IgE concentrations ranged from 0 to 2106 nM of IgE.

These data show that modification of the aptamer sequence with primer sites did not negatively impact aptamer affinity, more so, the affinity even improved slightly from 65nM to 15 nM.

The electrophoretic separations used to determine the K_d of the aptamers used also allow us to accurately determine the time window for collection of the aptamer-protein complex since it is clearly seen that the complex will elute within 3.5 minutes while any free aptamer will only elute after approximately 6 minutes (Figure S3).

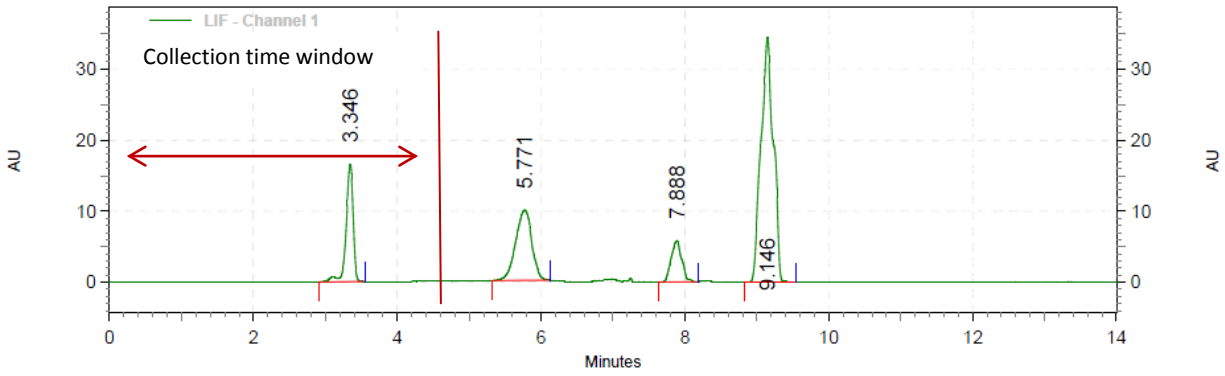

Figure S3: Determining the collection time window for successful complex collection. Signals are:aptamer-hIgE complex (3.346), free aptamer (5.771), Internal standards (7.888 and 9.146).

Equilibrium binding calculations

For calculation of equilibrium concentrations of bound and unbound species during aptamer-target complexations, following equilibrium equation was derived (E1):

Supporting Information

$$Fraction\ Aptamer\ Bound = \frac{[DNA] + [Pro] + K_d - \sqrt{[DNA]^2 - 2[DNA][Pro] + 2K_d[Pro] + [Pro]^2 + 2K_d[DNA] +}}{2[DNA]}$$

This equation was obtained by solving the system:

$$\frac{d}{dt} DNA(t) = -k1\ DNA(t)\ Prot(t) + k2\ Cplx(t)$$

$$\frac{d}{dt} Cplx(t) = k1\ DNA(t)\ Prot(t) - k2\ Cplx(t)$$

$$\frac{d}{dt} Prot(t) = -k1\ DNA(t)\ Prot(t) + k2\ Cplx(t)$$

Conservation laws:

$$DNA(t) + Cplx(t) = C1$$

$$-DNA(t) + Prot(t) = C2$$

Supplementary qPCR data

Initially, primer sets 1 and 2 were evaluated for artifact formation during qPCR cycling. To this end, reaction mixtures were prepared using conditions as described in the main manuscript in the presence of:

- Only forward primer and no template (P1)
- Only reverse primer and no template (P2)
- Only forward and reverse primer and no template (P1P2)

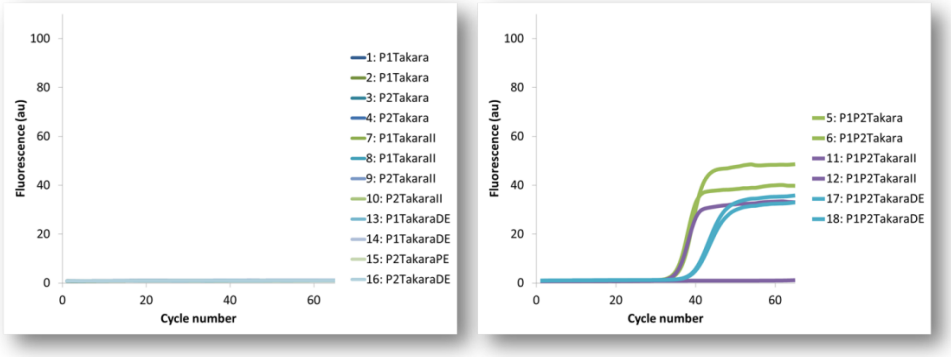

Figure S4: Evaluation of primer set 2 (Table 1, main manuscript) artifact formation using different qPCR mastermix formulations.

Results of this experiment show that each primer in itself does not give rise to the accumulation of detectable product during PCR cycling but when both primers are present, significant accumulation of product becomes apparent, indicating that these primers form cross-dimers that eventually participate in the amplification reaction. These dimers will thus compete with the template during amplification.

Supporting Information

The results shown in Figure S4 are based on experiments with primer set 2, primer set 1 gave similar results (not shown).

After performing these experiments, new primer sets were sought, both an existing primer for β-actin gene amplification (BactaBactas, Figure S4, set 3 Table 1, main manuscript). As well as *in silico* designed primer sets (Figure S5).

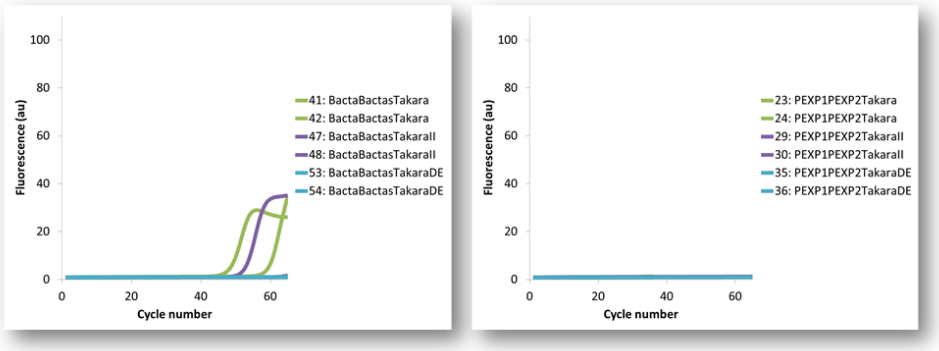

Figure S5: Comparison of primer set 3 and set 4 (Table 1, main manuscript) artifact formation using different qPCR mastermix formulations.

These results show that primer set 1 performs better with the use of the proper commercially available qPCR master mix formulation with significant reduction of artifact formation. However, with the use of the *in silico* designed primer sets, the formation of artifacts during qPCR cycling could be completely suppressed. Primer sets 5 and 6 display identical behavior to set 4 when this experiment is performed using those primer sets (not shown).

Supporting Information

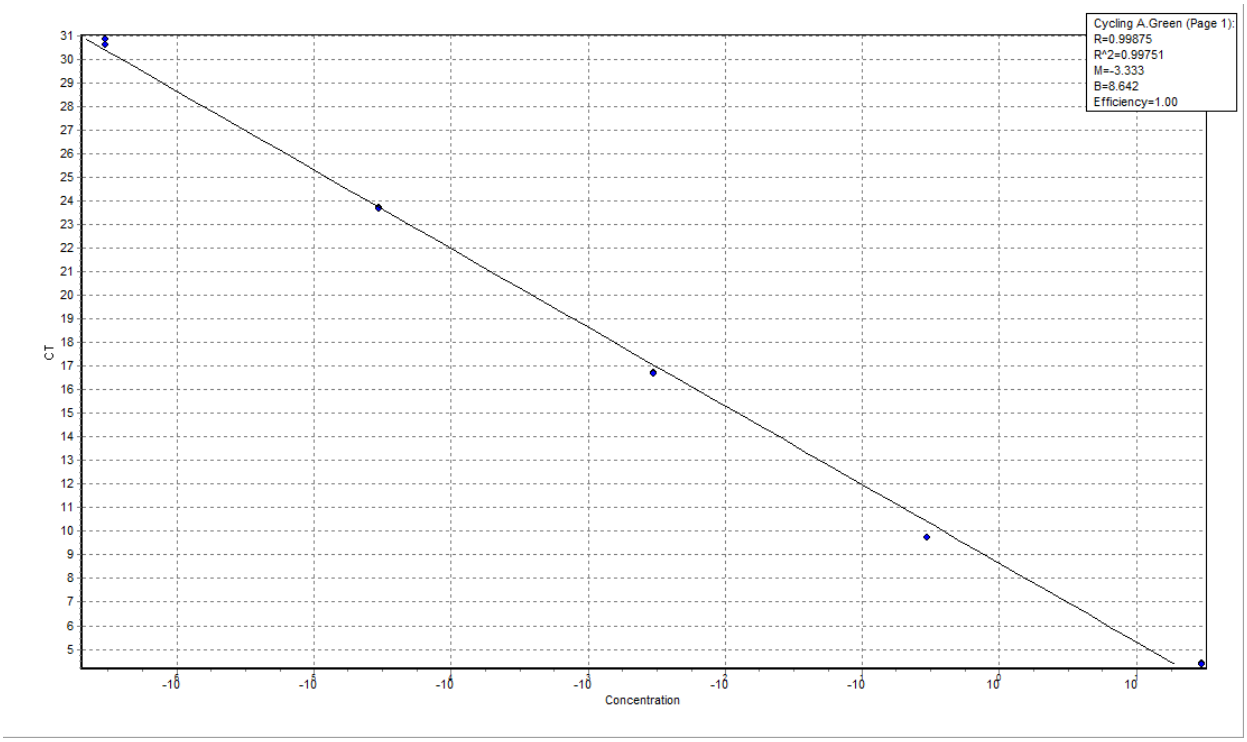

Figure S5: Calibration curve for a dilution series of the Wiegandaptamer modified with primer set 5 in the concentration range from 30 nM to 3 10^{-7}nM

Primer design script

```
% Do we want to do performance profiling?
profile_required = 1;

ifprofile_required
profileon
end

% How many primers do we wish to generate and screen?
primercount_per_length = 500000;
probecount = 2;

% How long do we want our primers to be?
primer_min_length = 20;
primer_length_range = 3;
primer_max_length = primer_min_length + primer_length_range - 1;
probeaddedlength = 0;
probelength = primer_min_length + probeaddedlength;

% Design criteria
hairpin_max_length = 3;
dimer_max_length = 3;
limiting_conc_M = 1000e-9;
excess_conc_M = 1000e-9;
probe_conc_M = 200e-9;
salt_conc_M = 0.05;

% Total amount of primers to generate
primercount_total = primercount_per_length * primer_length_range;

% Which Tm calculation algoritm to use?
%
% 1 - Basic (Marmur et al., 1962)
% 2 - Salt adjusted (Howley et al., 1979)
% 3 - Nearest-neighbor (Breslauer et al., 1986)
% 4 - Nearest-neighbor (SantaLucia Jr. et al., 1996)
% 5 - Nearest-neighbor (SantaLucia Jr., 1998)
% 6 - Nearest-neighbor (Sugimoto et al., 1996)
tmalgo = 5;

% Use consensus calculation for Tm?
useconsensus = 1;
```

Supporting Information

```
% Primer Tm limits
primer_lim_tm_min = 58;
primer_lim_tm_max = 62;
primer_ex_tm_min = 58;
primer_ex_tm_max = 62;

% Probe Tm limits
probe_tm_min = 40;
probe_tm_max = 70;

% Primer pctGC limits
primer_lim_pctgc_min = 40;
primer_lim_pctgc_max = 65;
primer_ex_pctgc_min = 40;
primer_ex_pctgc_max = 65;

% Probe pctGC limits
probe_pctgc_min = 40;
probe_pctgc_max = 65;

% Maximum difference in Tm for a primer set
tmdiffmax = 3;
tmdiffmin = 0;

% Do we want the primers to end in G or C or GC/CG?
doublegcclampstrict = 1;
doublegcclampnorm =0;

% Do we allow quads, triples or doubles?
quadallowed = 0;
tripleallowed = 0;
doubleallowed = 0;



% Allocate memory for the primer and probe list in advance, this will
% reduce execution time from 10+ hrs to 1000 secs for 1000000
% 20 nucleotide primers...
%primer_lim_list = repmat(char(0),primercount_per_length,primer_min_length);
primer_lim_list = cell(primercount_total,1);
primer_ex_list = cell(primercount_total,1);
%probelist = repmat(char(0),probecount,probelength);
probelist = cell(probecount,1);

% Allocate memory for the struct that contains all primer info that needs
% to be calculated. To my knowledge, there is no way to get the size in
% bytes/chars/whatever... of a struct. So we need to feed repmat an actual
% struct. Hence the use of oligoprop('CGTACGTACGTAGTACCGTACG');
primerprops_lim = repmat(oligoprop(randseq(primer_max_length)),1,primercount_total);
primerprops_ex = repmat(oligoprop(randseq(primer_max_length)),1,primercount_total);
probeprops = repmat(oligoprop(randseq(probelength)),1,probecount);

% Fill the primer list with random primer sequences of the required length
for i=0:primer_length_range - 1
for j=1:primercount_per_length
primer_lim_list(j + primercount_per_length * i) = cellstr(randseq(primer_min_length + i));
end;
end;

for i=0:primer_length_range - 1
for j=1:primercount_per_length
primer_ex_list(j + primercount_per_length * i) = cellstr(randseq(primer_min_length + i));
end;
end;

for i=1:probecount
probelist(i) = cellstr(randseq(probelength));
end;

% Calculate all properties of each primer, setting restrictions on Hairpin
% stem length and self dimer length. The shorter, the more stringent.
for i=1:primercount_total
primerprops_lim(i) = oligoprop(char(primer_lim_list(i)),...
'Salt',salt_conc_M,...
'Primerconc',limiting_conc_M,...
'HPBase', hairpin_max_length,...
'Dimerlength', dimer_max_length);
end;

for i=1:primercount_total
primerprops_ex(i) = oligoprop(char(primer_ex_list(i)),...
'Salt',salt_conc_M,...
'Primerconc',excess_conc_M,...
'HPBase', hairpin_max_length,...
'Dimerlength', dimer_max_length);
end;

for i=1:probecount
probeprops(i) = oligoprop(char(probelist(i)),...
'Salt',salt_conc_M,...
```

Supporting Information

```
'Primerconc',probe_conc_M,...
'HPBase', hairpin_max_length,...
'Dimerlength', dimer_max_length);
end;

% All "markings" of bad primers are archived by generating a logical
% indexing vector of dimensions primercount_per_length * 1 (1 Column).
% Mark all primers with bad %GC in the result set.
primer_lim_gc = [primerprops_lim.GC]';
bad_primers_lim_gc = primer_lim_gc<primer_lim_pctgc_min |...
primer_lim_gc>primer_lim_pctgc_max;

primer_ex_gc = [primerprops_ex.GC]';
bad_primers_ex_gc = primer_ex_gc<primer_ex_pctgc_min |...
primer_ex_gc>primer_ex_pctgc_max;

probegc = [probeprops.GC]';
bad_probes_gc = probegc<probe_pctgc_min |...
probegc>probe_pctgc_max;

% Mark all primers with bad Tm in the result set.
primer_lim_tm = cell2mat({primerprops_lim.Tm}');
if (useconsensus)
primer_lim_TMconcensus = Melo2004TM(primer_lim_tm(:,3),primer_lim_tm(:,5),...
primer_lim_tm(:,6),GCclass(primer_lim_gc),cellfun('length', primer_lim_list));
bad_primers_lim_tm = primer_lim_TMconcensus(:,1) <primer_lim_tm_min |...
primer_lim_TMconcensus(:,1) >primer_lim_tm_max;
else
bad_primers_lim_tm = primer_lim_tm(:,tmalgo) <primer_lim_tm_min |...
primer_lim_tm(:,tmalgo) >primer_lim_tm_max;
end

primer_ex_tm = cell2mat({primerprops_ex.Tm}');
if (useconsensus)
primer_ex_TMconcensus = Melo2004TM(primer_ex_tm(:,3),primer_ex_tm(:,5),...
primer_ex_tm(:,6),GCclass(primer_ex_gc),cellfun('length', primer_ex_list));
bad_primers_ex_tm = primer_ex_TMconcensus(:,1) <primer_ex_tm_min |...
primer_ex_TMconcensus(:,1) >primer_ex_tm_max;
else
bad_primers_ex_tm = primer_ex_tm(:,tmalgo) <primer_ex_tm_min |...
primer_ex_tm(:,tmalgo) >primer_ex_tm_max;
end

% Mark all probes with bad Tm in the result set.
probetm = cell2mat({probeprops.Tm}');
if (useconsensus)
probeGCclass = GCclass(probegc);
probeTMconcensus = Melo2004TM(probetm(:,3),probetm(:,5),...
probetm(:,6),probeGCclass,cellfun('length', probelist));
bad_probes_tm = probeTMconcensus(:,1) <probe_tm_min |...
probeTMconcensus(:,1) >probe_tm_max;
else
bad_probes_tm = probetm(:,tmalgo) <probe_tm_min |...
probetm(:,tmalgo) >probe_tm_max;
end

% Mark all primers that form hairpins and/or self-dimers.
bad_primers_lim_dimers  = ~cellfun('isempty',{primerprops_lim.Dimers}');
bad_primers_lim_hairpin = ~cellfun('isempty',{primerprops_lim.Hairpins}');

bad_primers_ex_dimers  = ~cellfun('isempty',{primerprops_ex.Dimers}');
bad_primers_ex_hairpin = ~cellfun('isempty',{primerprops_ex.Hairpins}');

bad_probes_dimers  = ~cellfun('isempty',{probeprops.Dimers}');
bad_probes_hairpin = ~cellfun('isempty',{probeprops.Hairpins}');

% Mark all primers that lack a GC clamp.
bad_primers_lim_clamp = MarkGCClamp(primer_lim_list, doublegcclampnorm, doublegcclampstrict);
bad_primers_ex_clamp = MarkGCClamp(primer_ex_list, doublegcclampnorm, doublegcclampstrict);

% Mark all probes that have a 5' G that could quench our reporter dye.
%bad_probes_5G = lower(probelist(:,1)) == 'g';
fiveg = regexpi(cellstr(probelist),'^g','ONCE');
bad_probes_5G = ~cellfun('isempty',fiveg);


% Mark all primers that contain repeats of identical bases.
% regexpi will return the index at which the match occurs.
repeats_lim = regexpi(cellstr(primer_lim_list),'a{2,}|c{2,}|g{2,}|t{2,}');
bad_primers_lim_repeats_2 = cellfun(@(x) length(x)>2,repeats_lim);
repeats_lim = regexpi(cellstr(primer_lim_list),'a{3,}|c{3,}|g{3,}|t{3,}');
bad_primers_lim_repeats_3 = cellfun(@(x) ~isempty(x),repeats_lim);
repeats_lim = regexpi(cellstr(primer_lim_list),'a{4,}|c{4,}|g{4,}|t{4,}');
bad_primers_lim_repeats_4 = cellfun(@(x) ~isempty(x),repeats_lim);
clearrepeats_lim;
repeats_ex = regexpi(cellstr(primer_ex_list),'a{2,}|c{2,}|g{2,}|t{2,}');
bad_primers_ex_repeats_2 = cellfun(@(x) length(x)>2,repeats_ex);
repeats_ex = regexpi(cellstr(primer_ex_list),'a{3,}|c{3,}|g{3,}|t{3,}');
bad_primers_ex_repeats_3 = cellfun(@(x) ~isempty(x),repeats_ex);
repeats_ex = regexpi(cellstr(primer_ex_list),'a{4,}|c{4,}|g{4,}|t{4,}');
bad_primers_ex_repeats_4 = cellfun(@(x) ~isempty(x),repeats_ex);
```

Supporting Information

```
clearrepeats_ex;

% repeats = regexpi(cellstr(probelist),'a{2,}|c{2,}|g{2,}|t{2,}','ONCE');
% bad_probes_repeats = ~cellfun('isempty',repeats);
repeats_prb = regexpi(cellstr(probelist),'a{2,}|c{2,}|g{2,}|t{2,}');
bad_probes_repeats_2 = cellfun(@(x) length(x)>2,repeats_prb);
repeats_prb = regexpi(cellstr(probelist),'a{3,}|c{3,}|g{3,}|t{3,}');
bad_probes_repeats_3 = cellfun(@(x) ~isempty(x),repeats_prb);
repeats_prb = regexpi(cellstr(probelist),'a{4,}|c{4,}|g{4,}|t{4,}');
bad_probes_repeats_4 = cellfun(@(x) ~isempty(x),repeats_prb);
clearrepeats_prb;

% Mark all primers that do not contain all nucleotide types.
primers_lim_lackingA = regexpi(cellstr(primer_lim_list),'[A]','ignorecase');
bad_primers_lim_lackingA = cellfun('isempty',primers_lim_lackingA);

primers_lim_lackingT = regexpi(cellstr(primer_lim_list),'[T]','ignorecase');
bad_primers_lim_lackingT = cellfun('isempty',primers_lim_lackingT);

primers_lim_lackingG = regexpi(cellstr(primer_lim_list),'[G]','ignorecase');
bad_primers_lim_lackingG = cellfun('isempty',primers_lim_lackingG);

primers_lim_lackingC = regexpi(cellstr(primer_lim_list),'[C]','ignorecase');
bad_primers_lim_lackingC = cellfun('isempty',primers_lim_lackingC);

primers_ex_lackingA = regexpi(cellstr(primer_ex_list),'[A]','ignorecase');
bad_primers_ex_lackingA = cellfun('isempty',primers_ex_lackingA);

primers_ex_lackingT = regexpi(cellstr(primer_ex_list),'[T]','ignorecase');
bad_primers_ex_lackingT = cellfun('isempty',primers_ex_lackingT);

primers_ex_lackingG = regexpi(cellstr(primer_ex_list),'[G]','ignorecase');
bad_primers_ex_lackingG = cellfun('isempty',primers_ex_lackingG);

primers_ex_lackingC = regexpi(cellstr(primer_ex_list),'[C]','ignorecase');
bad_primers_ex_lackingC = cellfun('isempty',primers_ex_lackingC);

probes_lackingA = regexpi(cellstr(probelist),'[A]','ignorecase');
bad_probes_lackingA = cellfun('isempty',probes_lackingA);

probes_lackingT = regexpi(cellstr(probelist),'[T]','ignorecase');
bad_probes_lackingT = cellfun('isempty',probes_lackingT);

probes_lackingG = regexpi(cellstr(probelist),'[G]','ignorecase');
bad_probes_lackingG = cellfun('isempty',probes_lackingG);

probes_lackingC = regexpi(cellstr(probelist),'[C]','ignorecase');
bad_probes_lackingC = cellfun('isempty',probes_lackingC);

% Combine matrices containing marked positions of bad primers for each
% criterium. This will generate a logical indexing vector of dimensions
% primercount_per_length * number of properties checked (6 in this case).
bad_primers_lim = [bad_primers_lim_gc,...
bad_primers_lim_tm,...
bad_primers_lim_dimers,...
bad_primers_lim_hairpin,...
bad_primers_lim_clamp,...
    bad_primers_lim_repeats_2,...
    bad_primers_lim_repeats_3,...
    bad_primers_lim_repeats_4,...
bad_primers_lim_lackingA,...
bad_primers_lim_lackingT,...
bad_primers_lim_lackingG,...
bad_primers_lim_lackingC];

bad_primers_ex = [bad_primers_ex_gc,...
bad_primers_ex_tm,...
bad_primers_ex_dimers,...
bad_primers_ex_hairpin,...
bad_primers_ex_clamp,...
    bad_primers_ex_repeats_2,...
    bad_primers_ex_repeats_3,...
    bad_primers_ex_repeats_4,...
bad_primers_ex_lackingA,...
bad_primers_ex_lackingT,...
bad_primers_ex_lackingG,...
bad_primers_ex_lackingC];

% Store the position of all good primers.
% First, all(~bad_fwdprimers,2) generates a column containing 0 if any
% criterium is not met. This column contains primercount_per_length values. Secondly,
% find() returns the actual index of primers that meet all criteria in the
% original list.
%
% With B =
% 0 0 0 0 1 0 -> All yields 0
% 1 1 1 1 1 1 -> All yields 1
% 0 0 0 1 1 0 -> All yields 0
%
% C = All (B, 2) yields -> Check along dimension 2 (Columns)
```

Supporting Information

```
%
% C =
% 0
% 1
% 0
%
% Thus, we get a colum containing the positions of good
% primers. This column holds =<primercount_per_length values.
good_pos_primers_lim = find(all(~bad_primers_lim,2));
good_pos_primers_ex = find(all(~bad_primers_ex,2));

% Create a list containing only the good primers for further processing.
% we do this by dumping the contents of the original primer_lim_list into a new
% list for good primers only. This gives us a matrix of char with
% primer_min_length columns and a number of rows equal to the number of good
% primers.
%good_primers = primer_lim_list(good_pos_primers,:);
good_primers_lim = primer_lim_list(good_pos_primers_lim);
good_primers_ex = primer_ex_list(good_pos_primers_ex);

% Also store the properties for these good primers.
good_prop_primers_lim = primerprops_lim(good_pos_primers_lim);
good_prop_primers_ex = primerprops_ex(good_pos_primers_ex);

% Create histogram for the primer Tm distribution.
figure

hist(primer_lim_TMconcensus);

figure

hist(primer_ex_TMconcensus);

% Also store the consensus TM.
good_consensusTM_primers_lim = primer_lim_TMconcensus(good_pos_primers_lim);
good_consensusTM_primers_ex = primer_ex_TMconcensus(good_pos_primers_ex);

% Count the number of good primers.
N_good_primers_lim = numel(good_prop_primers_lim);
N_good_primers_ex = numel(good_prop_primers_ex);

% Display this intermediate data.
figure

% We want to both display the results for each criterion test as well as a
% global result for each primer. A primer fails if any criterion is not
% matched, hence the expansion of the bad_fwdprimers logical indexing
% matrix with an extra column.
imagesc([bad_primers_lim any(bad_primers_lim,2)]);

% Set some basic graph properties.
title('Filtering candidate forward primers');
ylabel('Primer ID');
xlabel('Criteria');

set(gca,...
'Xtick',...
    1:13);

set(gca,...
'XtickLabel',...
char({'%GC',...
'Tm',...
'SDim',...
'HP',...
'GCc',...
'Rep 2',...
'Rep 3',...
'Rep 4',...
' -A',...
' -T',...
' -G',...
' -C',...
'all'}));

%set(gca,'Position',[0.1 0.11 .7 0.81]);

annotation(gcf,'textbox','String','OK','Color','w',...
'Position',[0.92 0.8 0.07 0.06],'BackgroundColor',[0 0 0.6275]);

annotation(gcf,'textbox','String','NOK','Color','w',...
'Position',[0.92 0.72 0.07 0.06],'BackgroundColor',[0.502 0 0]);

% Combine matrices containing marked positions of bad probes for each
% criterium. This will generate a logical indexing vector of dimensions
% primercount_per_length * number of properties checked. We can re-use some of the
% analysis data already generated for the primer screening. In fact, only
% Tm is a separate criterion.
bad_probes = [bad_probes_gc,...
```

Supporting Information

```
bad_probes_tm,...
bad_probes_dimers,...
bad_probes_hairpin,...
    bad_probes_repeats_2,...
    bad_probes_repeats_3,...
    bad_probes_repeats_4,...
    bad_probes_5G,...
bad_probes_lackingA,...
bad_probes_lackingT,...
bad_probes_lackingG,...
bad_probes_lackingC];

% Store the position of all good probes. As done also with primers.
good_pos_probes = find(all(~bad_probes,2));

% Create a list containing only the good primers for further processing.
% we do this by dumping the contents of the original primer_lim_list into a new
% list for good primers only. This gives us a matrix of char with
% primer_min_length columns and a number of rows equal to the number of good
% primers.
good_probes = probelist(good_pos_probes,:);

% Also store the properties for these good primers.
good_prop_probes = probeprops(good_pos_probes);

% Also store the consensus TM.
good_consensusTM_probes = probeTMconcensus(good_pos_probes);

% Count the number of good primers.
N_good_probes = numel(good_prop_probes);

% Display this intermediate data.
figure

% We want to both display the results for each criterion test as well as a
% global result for each primer. A primer fails if any criterion is not
% matched, hence the expansion of the bad_fwdprimers logical indexing
% matrix with an extra column.
imagesc([bad_probes any(bad_probes,2)]);

% Set some basic graph properties.
title('Filtering candidate probes');
ylabel('Probe ID');
xlabel('Criteria');

set(gca,...
'Xtick',...
    1:13);

set(gca,...
'XtickLabel',...
char({'%GC',...
'Tm',...
'SDim',...
'HP',...
'Rep 2',...
'Rep 3',...
'Rep 4',...
' 5G',...
' -A',...
' -T',...
' -G',...
' -C',...
'all'}));

%set(gca,'Position',[0.1 0.11 .7 0.81]);

annotation(gcf,'textbox','String','OK','Color','w',...
'Position',[0.92 0.8 0.07 0.06],'BackgroundColor',[0 0 0.6275]);

annotation(gcf,'textbox','String','NOK','Color','w',...
'Position',[0.92 0.72 0.07 0.06],'BackgroundColor',[0.502 0 0]);


% Start checking cross-hybridisation. We check primer pairs first.
% Cross dimerization can occur between the forward and reverse primer if
% they have a significant amount of complementarity. The primers will not
% function properly if they dimerize with each other. To check for
% dimerization, align every forward primer against every reverse dimer,
% using the swalignfunction, and keep the low-scoring pairs of primers.
% This information can be stored in a matrix with rows representing fwd
% primers and columns representing reverse primers. This calculation is
% quite time consuming, but you can reduce the time taken by noticing that
% there is no point in performing this calculation on primer pairs where
% the reverse primer is identical to the forward primer. The image in the
% figure shows the pairwise scores before being thresholded, low scores
% (dark blue) represent primer pairs that do not dimerize.
%
% Doing the cross-dimerization check only makes sense if we have actual
% primer candidates, more specifically, we need more than one good primer
```

Supporting Information

```
% to exist.
if ((N_good_primers_lim> 1) && (N_good_primers_ex> 1))

% Define an alignment scoring matrix.
scr_mat = [-1,-1,-1,1;-1,-1,1,-1;-1,1,-1,-1;1,-1,-1,-1;];

% Pre-allocate memory for the alignment scores.
primer_scores = zeros(N_good_primers_lim,N_good_primers_ex);

% Generate the alignment score matrix.
for i = 1:N_good_primers_lim
for j = 1:N_good_primers_ex
primer_scores(i,j) = swalign(char(good_primers_lim(i)),...
char(good_primers_ex(j)), ...
'SCORINGMATRIX',scr_mat,...
'GAPOPEN',5,...
'ALPHA',...
'NT');
end
end

% Generate an image based on the scoring matrix we just generated.
figure
imagesc(primer_scores)
title('Cross dimerization scores')
xlabel('Candidate reverse primers')
ylabel('Candidate forward primers')
colorbar

% Finally, we would like to generate a text list of the good primer pairs
% with all useful properties. We will only take those pairs which have a
% the lowest scores in the scr matrix.
minimum_score_primers = min(primer_scores(:));
maximum_score_primers = max(primer_scores(:));
primer_paircount = 0;

    Primers = sprintf('Pair\tPrimer\t%%GC\tmT\n\n');

% Could probably be more efficient but ...
if (useconsensus == 0)
for i = 1:N_good_primers_lim
for j = 1:N_good_primers_ex
if (...
                        (primer_scores(i,j) == minimum_score_primers)...
&& (abs(good_prop_primers_lim(i).Tm(tmalgo)-good_prop_primers_ex(j).Tm(tmalgo)) >tmdiffmin)...
&& (abs(good_prop_primers_lim(i).Tm(tmalgo)-good_prop_primers_ex(j).Tm(tmalgo)) <tmdiffmax)...
                        )
primer_paircount = primer_paircount + 1;

                    Primers = sprintf('%s%-21s\t%-8d\t%-8d\t%-4.4g\n%-21s\t%-8d\t%-8d\t%-4.4g\n\n',...
                        Primers,...
char(good_primers_lim(i)),...
good_pos_primers_lim(i),...
round(good_prop_primers_lim(i).GC),...
good_prop_primers_lim(i).Tm(tmalgo),...
char(good_primers_ex(j)),...
good_pos_primers_ex(j),...
round(good_prop_primers_ex(j).GC),...
good_prop_primers_ex(j).Tm(tmalgo));
end
end
end
else
for i = 1:N_good_primers_lim
for j = 1:N_good_primers_ex
if (...
                        (primer_scores(i,j) == minimum_score_primers)...
&& (abs(good_consensusTM_primers_lim(i)-good_consensusTM_primers_ex(j)) >tmdiffmin)...
&& (abs(good_consensusTM_primers_lim(i)-good_consensusTM_primers_ex(j)) <tmdiffmax)...
                        )
primer_paircount = primer_paircount + 1;

                    Primers = sprintf('%s%-21s\t%-8d\t%-8d\t%-4.4g\n%-21s\t%-8d\t%-8d\t%-4.4g\n\n',...
                        Primers,...
char(good_primers_lim(i)),...
good_pos_primers_lim(i),...
round(good_prop_primers_lim(i).GC),...
good_consensusTM_primers_lim(i),...
char(good_primers_ex(j)),...
good_pos_primers_ex(j),...
round(good_prop_primers_ex(j).GC),...
good_consensusTM_primers_ex(j));
end
end
end
end

disp(Primers)

end
```

Supporting Information

```
% Start checking cross-hybridisation probes and primers.
%
% Doing the cross-dimerization check only makes sense if we have actual
% primer candidates, more specifically, we need more than one good primer
% to exist.
if (N_good_primers_lim> 1 &&N_good_primers_ex> 1 &&N_good_probes> 1)


% Pre-allocate memory for the alignment scores.
probe_scores_lim = zeros(N_good_primers_lim,N_good_probes);

% Generate the alignment score matrix.
for i = 1:N_good_primers_lim
for j = 1:N_good_probes
probe_scores_lim(i,j) = swalign(char(good_primers_lim(i)),...
char(good_probes(j)), ...
'SCORINGMATRIX',scr_mat,...
'GAPOPEN',5,...
'ALPHA',...
'NT');
end
end

% Generate an image based on the scoring matrix we just generated.
figure
imagesc(probe_scores_lim)
title('Cross dimerization scores')
xlabel('Candidate probes')
ylabel('Candidate primers')
colorbar

% Pre-allocate memory for the alignment scores.
probe_scores_ex = zeros(N_good_primers_ex,N_good_probes);

% Generate the alignment score matrix.
for i = 1:N_good_primers_ex
for j = 1:N_good_probes
probe_scores_ex(i,j) = swalign(char(good_primers_ex(i)),...
char(good_probes(j)), ...
'SCORINGMATRIX',scr_mat,...
'GAPOPEN',5,...
'ALPHA',...
'NT');
end
end

% Generate an image based on the scoring matrix we just generated.
figure
imagesc(probe_scores_ex)
title('Cross dimerization scores')
xlabel('Candidate probes')
ylabel('Candidate primers')
colorbar


% Finally, we would like to generate a text list of the good primer pairs
% with all useful properties. We will only take those pairs which have a
% the lowest scores in the scr matrix.
minimum_score_probes_lim = min(probe_scores_lim(:));
maximum_score_probes_lim = max(probe_scores_lim(:));
minimum_score_probes_ex = min(probe_scores_ex(:));
maximum_score_probes_ex = max(probe_scores_ex(:));

av_minimum_score = (...
minimum_score_primers+ ...
minimum_score_probes_lim+ ...
minimum_score_probes_ex) / 3;

av_maximum_score = (...
maximum_score_primers+ ...
maximum_score_probes_lim+ ...
maximum_score_probes_ex) / 3;

    Primers = sprintf('Pair\tPrimer\t%%GC\tmT\n\n');

% We have 3 scoring matrices.
% Could probably be more efficient but ...
if (useconsensus == 0)
for i = 1:N_good_primers_lim
for j = 1:N_good_primers_ex
for k =1:N_good_probes
if (...
                            (primer_scores(i,j) == minimum_score_primers)...
&& (abs(good_prop_primers_lim(i).Tm(tmalgo)-good_prop_primers_ex(j).Tm(tmalgo)) >tmdiffmin)...
&& (abs(good_prop_primers_lim(i).Tm(tmalgo)-good_prop_primers_ex(j).Tm(tmalgo)) <tmdiffmax)...
&& (probe_scores_lim(i,k) == minimum_score_probes_lim)...
&& (probe_scores_ex(j,k) == minimum_score_probes_ex)...
                            )
```

Supporting Information

```
                        Primers = sprintf('%s%-26s\t%-8d\t%-8d\t%-4.4g\n%-26s\t%-8d\t%-8d\t%-4.4g\n%-26s\t%-8d\t%-8d\t%-
4.4g\n\n',...
                            Primers,...
char(good_primers_lim(i)),...
good_pos_primers_lim(i),...
round(good_prop_primers_lim(i).GC),...
good_prop_primers_lim(i).Tm(tmalgo),...
char(good_primers_ex(j)),...
good_pos_primers_ex(j),...
round(good_prop_primers_ex(j).GC),...
good_prop_primers_ex(j).Tm(tmalgo),...
char(good_probes(k)),...
good_pos_probes(k),...
round(good_prop_probes(k).GC),...
good_prop_probes(k).Tm(tmalgo));
end
end
end
end
else
for i = 1:N_good_primers_lim
for j = 1:N_good_primers_ex
for k =1:N_good_probes
if (...
                            (primer_scores(i,j) == minimum_score_primers)...
&& (abs(good_consensusTM_primers_lim(i)-good_consensusTM_primers_ex(j)) >tmdiffmin)...
&& (abs(good_consensusTM_primers_lim(i)-good_consensusTM_primers_ex(j)) <tmdiffmax)...
&& (probe_scores_lim(i,k) == minimum_score_probes_lim)...
&& (probe_scores_ex(j,k) == minimum_score_probes_ex)...
                            )

                        Primers = sprintf('%s%-26s\t%-8d\t%-8d\t%-4.4g\n%-26s\t%-8d\t%-8d\t%-4.4g\n%-26s\t%-8d\t%-8d\t%-
4.4g\n\n',...
                            Primers,...
char(good_primers_lim(i)),...
good_pos_primers_lim(i),...
round(good_prop_primers_lim(i).GC),...
good_consensusTM_primers_lim(i),...
char(good_primers_ex(j)),...
good_pos_primers_ex(j),...
round(good_prop_primers_ex(j).GC),...
good_consensusTM_primers_ex(j),...
char(good_probes(k)),...
good_pos_probes(k),...
round(good_prop_probes(k).GC),...
good_consensusTM_probes(k));
end
end
end
end
end

disp(Primers)

end


% Check how long the code took to execute.
ifprofile_required
profileviewer
profileoff
end
```