Technical Note: Using the Diagnostic Mode of SeedCount

Bruce Armstrong, 17 Feb 04

Introduction:
The diagnostic mode allows us to make much more precise calibrations of SeedCount due to its method of rapidly reiterating the calculations through a range of values without needing to reload the image, etc for each value.

Reference file:
To work accurately, this mode needs a reference file for the image being examined which contains manually assessed information on creases, blackpoint and discoloured kernels on a kernel by kernel basis and the mechanical screenings info for the entire sample as a separate section. The ref file also contains the blob numbers and x and y positions of each seed to ensure that the match between the ref file and the image being analysed is correct.

The reference file header also contains the location and name of the image file, the grain type and sample mass to allow accurate calculations.

Initialisation File:
The INI file has been expanded to include a large number of variables that are expected to have a strong effect on the accuracy of SeedCount and which need to be tested on real samples to determine the correct settings. The variables can be used in either production or diagnostic mode.

We can select which mode we are using by placing an x in front of the mode we do not want and removing from the mode we want. The following example will run in diagnostic mode:

[Miscellaneous]
Mode=Diagnostic
xMode=Production

In production mode, none of the options contained in the [Diagnostic] section will be used. The variables will instead be used with the values set in the [blackpoint], [Screenings] or [Creases] section.

In Diagnostic mode, SeedCount will check to see if a range of values have been set in the [Diagnostic] section and if so, use them. If they have not been set for a particular variable, it will then use the value set in the earlier sections just as the Production mode does.

Values and ranges are set by removing the “x” in front of them, eg:

xWid_W_BlackpointPopulationThreshold_Min=30
xWid_W_BlackpointPopulationThreshold_Max=40
xWid_W_BlackpointPopulationThreshold_Delta=2

will not be run, but instead will use this line from the [Blackpoint] section:

Wid_W_BlackpointPopulationThreshold=32

But the following will run, and the line above will then be ignored:

Wid_W_BlackpointPopulationThreshold_Min=30
Wid_W_BlackpointPopulationThreshold_Max=40
Wid_W_BlackpointPopulationThreshold_Delta=2

The above will cause the code to branch when it hits this part of the code and step through the various possibilities (eg 30, 32, etc til 40). The code will also output a datafile line for the result of each calculation. As the above is a blackpoint variable, it will appear in the blackpoint diag file.

There are currently 4 Diagnostic output files, which are:

*_BP_DIAG.txt

* _Crease_DIAG.txt

* _Dif_DIAG.txt

*_screening_DIAG.txt

where * is the name of the diagnostic reference file.

Range and Delta Selection:
This requires some experience, but in most cases the ini file will contain values that make a reasonable starting point. This is not always the case, as some variables contain "fine-tuning" values.

To select a correct range, it should be broad enough that you get a sweep of scores in the Diag files with your optimal value inside the range. If your "best" value is at one end of the range it is possible that an even better result is outside the range you have chosen and you will need to alter the range and rerun it. If you are running a very broad range, it may be that you can use a coarse delta (say 3 or 5) to rapidly step through it. If it looks to be fairly sensitive at some section of the output, you can then narrow the range and decrease the delta to find the optimum.

For some variables, such as the screenings, delta will be very small (eg 0.05) while for others a delta of 5 will be fine enough. Many variables will need a delta of 1. This can be determined by the function of the variable (some are integers so 1 is as fine as it gets) and the rate of score change from one increment to the next.

Multi-variables

There are some practical limits on how many variables should be active in the diagnostic mode at any one time. This is due to the multiplicative interactions.

For example, running 10 levels of one variable reruns that part of the code 10 times and produces 10 lines in that output file. But if 3 variables are activated and each has 10 levels, we wind up with 10 *10 * 10 = 1000 runs/combinations. The number of runs can be controlled somewhat by limiting the number of levels for each active variable or best by reducing the number of active variables. In practice I have found that about 200 or 300 runs is a reasonable max number, but wherever possible it is best to run only one active variable at a time. Wherever the action of the variables is mostly independent this works well. Where there are strong interactions it is necessary to run them together.

One alternative to this is where the active variables are in different parts of the code. For example, you can have an active variable in both the blackpoint and screenings areas and they will not interact and the total number of runs will just be additive: eg 10 BP runs and 10 screenings runs. This is a good way to maximise the output with a very small increase in run time as you can calibrate both

the Bp and screenings at the same time. Some of the white and discolour variables can also be run like this…

Though this will in theory work also for creases, in reality it is necessary to first maximise the crease detection before working on any of the others as poor crease detection will limit the effectiveness of the other systems as they depend on (non-) crease detection to select their seeds for analysis.

Multi-images

It may often be that it will be useful to run more than one image at a time. This will especially be the case for the final calibrations where we want to find the best all-round value for a set of images rather than optimise a single image. I am not sure how many images can be run as a set, but have so far run 4 without any problems. This may also be a useful way to expand the number of things that can be tested in a run series, as the program can be set up to run say 3 wheat samples and then 4 barley samples in the series. As each image produces its own output files there are no interactions between each image's variables. However, you can only use the one range for all of the wheat samples for example, so if you need quite different ranges for different images, this process will either require using a very broad range of values or segregating the images into similar groups and running them as different groupings. (eg red and white wheat – but our hope is to be able to run all wheats with the one calibration so we need to try them all together at first and only split them if this fails to give consistent results.)

Though by running multi-images you will lose some of the time gains from the diagnostic as it will still need to load each image, overall it means that you may be able to proceed with other work while the diag runs and it will save you having to change ref files or reactivate the same variables again and again. Using a slightly broader variable range may well be worthwhile. The downside is that there will be too many files for them all to fit in Excel's "recent file" list to make them quick to open. But if you have all the ref files and images in only one or two directories (eg barley and wheat) this will make the navigation fairly quick as the diag output files are put into the same directory as the ref file… the diag files are also structured to load quickly into Excel by just requiring you to use the next, finish buttons. A macro that sequentially loaded and closed the diag files would be handy, but I haven't tried to do that yet.

Each ref file to be used must be listed in the ini file and must have a sequential set of file numbers, starting at 1, in order to run. Unused ref files should be commented out. See how this is done in the ini file….

The path to the ref file in the ini file, and the path to the image in the ref file must both be accurate for it to work.

Running:

All the diag output files must be closed before beginning a new run. Failing to do so will prevent them being written to and the run will need to be done again.
After changing the ini file to reflect which variables and ref files are to be used, it must be saved. Then start SeedCount running. If it is running in diagnostics mode, a confirmation box will come up after a few seconds, followed by the "Analysing" box which will disappear when the run series is finished.

If any variables have been set to "illegal" values, the run will fail and you must find out which values were outside the proper range.

Interpretation

When the run is finished, it must be analysed to find the optimum. This usually is the highest score for Creases, BP and Dis and the lowest score for screenings (0 means a perfect match between SC and mech screenings.)

But some thought needs to go into this. The ideal is a perfect match – all creases found, no false positives, no false negatives. The reality is to look for the best compromise between the false negs and positives.

It is essential to track your changes. This can be done either on paper or by pasting the "best line" into a spreadsheet. I tend to use an Excel printout that has the heading row from the appropriate DIAG file, one line beneath it with all the initial values in it, but has about thirty blank lines/boxes below it. On each line record the chosen best result of the current run diag/ref file, giving all its scores and recording only the optimum value that was altered to get that result.

This way it is easy to glance down the list and see where significant improvements have come from. The last entry in any column is the current value for that variable. When pasting into a spreadsheet all values are pasted and it is harder to see what is being changed.

Once a new variable value has been determined, transfer it to the "production" section of the ini file by replacing the current value and turn off that diagnostic variable and move on to another by adding and removing "x"s.

Excel Calibration Test Files:
More detailed info on what is happening can be obtained from the various _debug files in the VB\Data directory or by pasting the blobs.txt file into the appropriate *.calibration test excel spreadsheet which will give you seed by seed data. Remind me to send these out. The blobs file will correspond to the last run in the series, so if you want a particular set of results you should set these constants and run it as a single option. Hidden columns can be seen by highlighting the headers of the adjacent columns or rows and right-clicking and selecting Unhide (or hide to reverse the operation by selecting the columns to hide).

For **creases**, false positives are somewhat more tolerable than false negs. Perhaps try to finish up with a system that has a few more false pos than false neg, but as few as possible of both… As noted above, creases must be optimised before the others, as errors in creases will carry over into irreparable errors in the other functions.

The crease code uses a slice down the middle of the kernel and then compares the average greyscale value of various rows and combinations of rows to find the creases. The slice radius (or width) is the number of columns each side of the kernel center that are used for the comparisons. Small variations in this may be useful, eg try perhaps 3 to 13 with a delta of 1.

Three compare structures are used: The top structure looks for a dark upper section and compares it against a lighter lower section. It tends to use a narrow (or no) gap between the rows compared and a fairly high difference before it marks the combination as a crease. The strip size (CONST_T_Strip_Size) is the number of rows used in each block of the compare. For example, a size of 2 means that each block has two rows averaged in it. A gap (CONST_T_Strip_Gap) of 1 means that there is one row between the blocks that is not used. For this example, if the offset is 1, the first (top) row of the seed is ignored. The next two rows are averaged, the fourth row is ignored and the 5^{th} and 6^{th} rows averaged. These averages are compared and if greater than the threshold (Top_Diff_Value), SC assumes it has found a crease. It then moves the entire structure one row down the seed and checks again to see if there is a "better" crease, etc until it reaches the bottom offset and stops. Control then passes to the next test structure.

The bottom structure is the same as the top one but is reversed and works up from the bottom of the seed. They both move somewhat across the center of the seed.

The Middle structure is similar but is sort of like a top and bottom structure combined. It is looking for a dark central crease and thus has three blocks and two gaps separating them. This structure can

vary from being narrow to quite large to find creases of varying widths. As an example, with a size of three and a gap of two, each structure will need 3+2+3+2+3 rows, which is 13 rows. We used to have three of these middle structures, one used from near the top to the center of the seed, one used around the center and another from the center towards the bottom. This was because of the seed darkening near the edges again. We reduced the three types to one, retaining the "top-mid" one. We were able to eliminate the other two structures by adding a slant factor to the difference values that compensated for the increasing darkness as we near the edges of the seeds. We also added a slant factor to the top and bottom structures. We use separate difference variables for comparing the top to mid and mid to bot blocks within this structure to compensate for any variation in lighting from the two directions.

The strip gaps and sizes happen within the crease code and we can't get them to output their current values so instead we include these settings in the crease diag file. It is possible to look at these values and count down the rows and work out which value combination has worked best.

A crease value of –1 means no crease was found. Creases are marked as the percent down the seed to the crease with 0 at the top and 100 at the bottom.

Setting Crease Constants:

The initial settings were chosen by working with only one of the three crease sections at a time. This was done by temporarily setting extremely high difference constants for the sections you did not want to find creases. Bookmark the initial settings as a comment.

You can then tune each section independently, aiming for low numbers of false positives each time as many of the false negs will be picked up by the other sections…

However, as this has already be done, it may be better to leave all sections active so their interactions on crease detection can be dealt with more effectively.

As the slant and difference values work together to select a crease, this is one area where it is best to run both these variables together. The Slant_T_Value is used for both the top and bottom sections, so it can be run with the top diff value and then with the bottom diff value. If it is different for them both, we will either need to make separate top and bottom slant variables or find a compromise value. Initial tests suggest that the slant is not needed for the top and bottom sections, but very useful for the middle section.

The difference values are perhaps the most critical part of the crease detection and need careful observation.

The offsets control which rows (lines) the various routines will have available for a compare. I think that the offsets are probably pretty well optimised now, but you may wish to test this a bit.

Again I think the strip sizes and strip gaps are likely pretty right, but some judicious testing on more images may make them better. As all of these variables are somewhat interrelated, rerunning an already reasonable calibration may result in a very good calibration. It should also result in a more robust calibration as the current one is only based on two samples.

For **BP**, initially ignore severity differences, as they can usually be set last… For BP I would rate false negs and pos as equally bad, so they should be "balanced". However, in doing this remember that some of the false negs are actually due to detecting false creases and conversely missed creases may lead to false positives.

With the BP, we can threshold in two different ways: dynamically and fixed. We can also do a combination of the two.

Briefly, fixed threshold requires us to set a greyscale threshold and all pixels darker than that are counted. Eg: Nar_W_BlackpointThresholdConstant=70

Then the seed is split into a left and right section and the number of dark pixels in each section is compared. If the difference is above the Bp population threshold, we may have BP. If it is not BP, we look at the total number of dark pixels in the seed. If they are above the Discoloured threshold, we have a disc seed. If both tests fail, it is just a 'normal' seed.

If it is BP, we then check to see if it has more dark pixels than the BP Severe threshold. If not, it is mild BP.

In the manual BP assessment in the ref file, -1 means that the seed has a crease and is not assessed for BP, 0 means no BP was seen, 1 means the BP is so slight that it is not counted commercially, 2 and 3 mean mild BP and 4 and 5 mean its severe.

However, it is more complex than this as the above method produces too many false results. As well as needing a min number of "extra" dark pixels on one side of the seed, we also require that side to have more than certain percent of the dark pixels (ie
Nar_W_LeftPct=65).

However, due to the angle the scanner views seeds near the edge of the tray and the ellipsoidal shape of the seeds, we remove a "slant" value from the "percent" for the end of the seed facing the nearest tray edge and add it to the end facing the tray center. The amount of slant decreases to zero at the tray center, at which position both ends of the seeds are viewed at the same angle. Eg:

new_bp_left_pct_value = bp_left_pct_value + (CONST_BP_Slant_L_Value * (x_mid - 1087));

Nar_W_Slant_L_Value=0.009

This works well for most seeds, but the fixed threshold is sensitive to the general image brightness and the kernel brightness. Thus seeds nearer the tray edge, which are more poorly lit, will tend to have more dark pixels.

So we bring in the dynamic threshold. This method tries to establish the general brightness of each kernel and set the threshold as a percentage of the general brightness. Assuming that the Bp will also look darker on a darker seed, this method should compensate for some lighting and sample color problems. Initially the pixels were sorted by brightness and a proportion of the total pixels counted from the brightest pixels were used. This way the presence of blackpoint was not able to influence the general brightness as it would if all of the pixels were used. Eg:

Nar_W_DynamicThreshCalcPercent=60

In practice it turned out that the dynamic brightness was influenced too much by bright pixels. Some of these bright pixels were reflections of the lamps off the glossier kernels, splits in the bran exposing the white endosperm and whitish "hair" regions on the wheat. This latter is esp a problem on red wheats where up to 12 to 15% of the apparent seed can be hair (eg BP-Ten and KV2).

This was dealt with by eliminating the brightest pixels and instead using a band of the more average pixels in each seed. Eg:

Nar_W_TopDynamicThreshCalcPercent=25

This combination uses the pixels in the 25 to 60% range of the kernel to produce the average kernel brightness.

The next step is to convert this brightness value into a threshold. This is done by using a percentage of this value. Eg:

Nar_W_BlackpointPercentOfReference=65

However, tests so far indicate that the purely dynamic threshold is too sensitive to kernel brightness. – This suggests that the intensity of the blackpoint is perhaps largely independent of the kernel color and is mainly affected by lighting levels.

So at this point a compromise seems to work best. At threshold that is approximately half fixed and half dynamic seems to reap the best of both methods. But this larger reference set should help prove this.

This can be tested like this:

Fixed only: Set the BP% of Ref to 0 and optimise for the BP Thresh Constant.

Dynamic only: Set the Bp Thresh Constant at 0 and optimise the BP% of ref.

If both of these approaches have difficulty producing an optimum that will work on all samples, try the Combination approach. Set ranges for the Threshold and % of ref constants so they will swing around about 50% of the values determined when using fixed or dyn only and see if there is a combination of the two that produces optimum results across the set of ref images….

When this seems to be "tuned", reset the pop threshold and severity thresholds again. The difficult part of BP detection is that all of the constants work together to select the BP, so altering one constant tends to "detune" most of the others to some extent.

This process must be done for both the narrow and wide sections.

Discoloured (dark and white (split)) are currently hard to optimise as we only have a few discoloured seeds in most trays that are crease down and they are often misidentified in the crease code due to their discolorations. But do your best to balance this.

The darks are found as "leftovers" from the BP selection

Use Nar_W_Dis_SeverityCutoff=60 where 60 is the number of dark pixels…

We can also find generally dark seeds using the BP dyn thresh cal values. This uses

Nar_W_Dyn_Dis_SeverityCutoff=75 as an example, where 75 is the average greyscale value of the pixels assessed by the Bp dynamic range. A dark discoloured seed is marked as a 6 in the Bp Adjustment column.

Whites (split seeds with the white endosperm showing) are usually distorted seeds that need to be removed from BP and screening tests. They have two settings. The first one sets the brightest X % of pixels to use, eg:

Nar_W_WhiteDynamicThreshCalcPercent=7

The second variable (Nar_W_Split_SeverityCutoff=243) is the min average brightness value of the above group that is needed to identify the seed as a “white” seed. If so identified, it is marked as a “400” in the seed type column. The “white” code is separate from the BP and screenings and can be run at the same time without creating massive runtime DIAG files (I think this is correct, but Tim should check on this…). But it shares the same DIAG file as the dark discoloured, so test these separately…

Screenings – aim for the lowest number as it is the best match. Points of particular concern are the 2.5 mm boundary in barley and the 2.0 mm in wheat as these are commercial break points. You will likely find it impossible to get a set of values that is optimal for all samples, but should be able to minimise the errors.

Start at the bottom end of the spectrum and work across it one boundary at a time. Ie, start at the 1.6 mm for wheat then work up to the 2.0, etc until done. (start at 2.0 for barley as it has no 1.6 mm values –ie its always 0) Each time try to optimise the group below the current threshold.

As mentioned before, this can be run at the same time as the BP optimisation…