Optimisation of card recognition routine

Emil Bagge
Abstract

Optimisation of card recognition routine

Emil Bagge

Card-dealing machines for the game Bridge are used to automate the time-consuming process of sorting cards. They require methods to recognize each card's suit and value during the process, as the sorting is predetermined. The machine considered in this thesis uses a webcam that feeds a 30 FPS video stream to a contour analysis algorithm. This thesis goes through and researches possible solutions for 3 different improvement areas: motion blur, degradation and colour recognition.

Motion blur occurs when the cards move around in the machine, resulting in heavily distorted images whose suit and value are difficult to recognize. I propose using a metric based on the variance of the Laplacian to recognize blurry images. Testing shows that this is an efficient and accurate method that allows the machine to save time by quickly discarding blurry images.

Degradation in the form of stains or colour loss risks breaking the connectivity of contours by distorting shapes and figures, making contour analysis unreliable. To deal with this I propose different morphological operations, such as closing and erosion, to quickly adjust these types of errors. By applying these methods, images whose suit and value were previously unrecognizable could be processed successfully. To compensate for the added run-time I propose implementing Otsu’s thresholding as a more efficient binarization method. Testing shows that it is 4 times faster than the old method. But since it is unable to binarize bright images I suggest using the old method as a fallback if Otsu’s method fails. More testing is needed to establish if time is ultimately saved.

Colour information could help the recognition but is currently not used. I propose a simple metric based on the amount of red pixels found by converting the RGB image into HSV and thresholding the hue channel. By only considering the center of the image, the thresholding becomes 3.5 times faster while also being less noisy than using the entire image. But since colour space conversion is a time-consuming process and the resulting information has limited use, it is unlikely that this method is worth implementing.

Out of the 3 different improvement areas that has been researched 4 methods are proposed, but only 2 show promise after testing; a blur metric based on the Laplacian and using morphological operations to fix distorted images. Otsu’s method is fast but unreliable and the redness metric results in very little value added relative to run-time cost.
1 Introduction

Card dealing machines for the game Bridge sort decks of cards in a predetermined order. This requires methods to identify each card’s suit and value during the process. The machine uses a webcam that feeds a contour analysis (CA) algorithm data through a 30 FPS video stream. The CA algorithm reads every frame, extracts contours and compares those to templates. Templates are stored contours and metadata that are added prior to run time as the recognition is based on comparisons. While the current algorithm works, it is not without its flaws and there are 3 main areas of improvement that relates to the machine either not being able to identify a card or worse, false positives (incorrectly identifying a card as another). The research questions given by these areas that this thesis will attempt to answer are the following:

- Would color recognition improve this system and where would it be implemented to have the most impact?
- What types of methods could be added to a real-time OCR algorithm to handle different types of degradation?
- Will the system be more accurate and faster by discarding distorted frames?

The following sections will go through each problem, analyze different methods and finally discuss and evaluate potential solutions with the research questions in mind.

1.1 Limitations

As seen in Fig 1, the human eye can easily distinguish which suit and value each card has. But the current algorithm cannot handle either of them for various reasons, which

![Examples of problematic images](image)

**Figure 1:** Examples of problematic images
will be discussed in the subsequent sections. But with unlimited time and computation power there exist methods, such as machine learning, that can accomplish this with high accuracy [Zhao et al., 2019]. Such complex methods are however not applicable in this setting, as the machine has very limited processing power and RAM memory (500MB) while being a real-time application with high demand for speed and accuracy. Solutions must be computationally simple and robust to stay efficient and accurate even with these limitations in mind. The point of this paper is not to introduce a completely new system to replace the CA algorithm, but to suggest changes and additional methods that could improve its performance.

It is also important to understand the characteristics of this image environment, as many techniques assume certain qualities to remain accurate. The camera is in a stable environment and is affected by little to no outside noise. The images taken does not have a high resolution and the image quality is low. Illumination on the objects of importance (the suit and value) is even and has a strong contrast with the background (white to black/red). While brightness is also even, the intensity varies on a frame per frame basis.
1.2 Hardware Specification

Experiments are performed on a local Linux environment, but not on the actual card dealing machine. The algorithm is written in C++ and uses the OpenCV library for many of its functions.

<table>
<thead>
<tr>
<th>Architecture:</th>
<th>x86_64</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU op-mode(s):</td>
<td>32-bit, 64-bit</td>
</tr>
<tr>
<td>CPU(s):</td>
<td>4</td>
</tr>
<tr>
<td>Thread(s) per core:</td>
<td>2</td>
</tr>
<tr>
<td>Model name:</td>
<td>Intel(R) Core(TM) i5-2520M CPU @ 2.50GHz</td>
</tr>
<tr>
<td>Stepping:</td>
<td>7</td>
</tr>
<tr>
<td>CPU MHz:</td>
<td>1218.492</td>
</tr>
<tr>
<td>CPU max MHz:</td>
<td>3200,0000</td>
</tr>
<tr>
<td>CPU min MHz:</td>
<td>800,0000</td>
</tr>
<tr>
<td>L1d cache:</td>
<td>32K</td>
</tr>
<tr>
<td>L1i cache:</td>
<td>32K</td>
</tr>
<tr>
<td>L2 cache:</td>
<td>256K</td>
</tr>
<tr>
<td>L3 cache:</td>
<td>3072K</td>
</tr>
<tr>
<td>RAM:</td>
<td>8GB</td>
</tr>
</tbody>
</table>

### Relevant Software

<table>
<thead>
<tr>
<th>Name</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>g++</td>
<td>7.5</td>
</tr>
<tr>
<td>Cmake</td>
<td>3.10.2</td>
</tr>
<tr>
<td>OpenCV</td>
<td>3.2</td>
</tr>
</tbody>
</table>

2 Motion Blur

The machine moves cards as it sorts, placing them in their correct place after recognition. Motion blur is caused by this movement and results in distorted frames. Since the cards are on top of each other some frames will have the next card partly covered by the moving card, which at worst can cover the suit and value, making the frame useless for further processing. Distorted frames should be discarded to avoid wasting time, as once the movement is done the camera will get a clear view of the card. To accomplish this a fast blur recognition method must be introduced.
2.1 Laplacian operators

There exist many techniques to measure the amount of blur an image has and much work has been done in this field. *Analysis of focus measure operators for shape-from-focus* [Pertuz et al., 2013] is a paper that compares and reviews around 36 different methods used for this purpose. The authors draws the conclusion that methods using Laplacian-based operators have the best overall performance at normal imaging conditions, which is suitable to this environment. These methods use a convolution kernel to approximate the second-order derivatives of the image to detect and highlight edges, resulting in an edge map as seen in Fig 2. The standard 3x3 kernel is used in this method to calculate the sum of two second order differences in both dimensions.

\[
L = \begin{pmatrix}
0 & -1 & 0 \\
-1 & 4 & -1 \\
0 & -1 & 0 \\
\end{pmatrix}
\]

The fastest way to make use of the resulting edge map is to calculate a single floating point value that can be used as a metric for sharpness. A common way to do so is to pool the pixel values and there are many different methods to do so. One of the proposed methods calculates the variance of the edge map, as a high variance indicates a high spread of responses which is representative of a normal image and vice versa [Pech-Pacheco et al., 2000]. The variance is computed by:

Figure 2: Edge detection using the Laplacian method, creating an image with the edges highlighted.
3 CONTOURS

\[ \text{LAP}_\text{VAR}(l) = \sum_{m}^{M} \sum_{n}^{N} [|L(m, n)| - \bar{L}]^2 \]

Where \( L(m, n) \) is the convolution of the original image \( l(m, n) \) with the kernel \( L \) and where \( \bar{L} \) is the mean of the absolute value given by:

\[ \bar{L} = 1/\text{NM} \sum_{m}^{M} \sum_{n}^{N} |L(m, n)| \]

Once the variance has been calculated it needs to be compared to a predetermined value. This value must be derived from the average variance that frames with no blur has against frames with heavy blur. The threshold needs to be set high enough to be effective but low enough to avoid mislabeling non-blurry frames as blurry. A concern with this method is that the machine could end up in a loop if the frames after the movement is done gets mislabeled as blurry. There are several methods that could be implemented if one wish to program defensively against this issue. An accumulator that counts the amount of discards in a row is a simple method that could work here, the idea being that if the accumulator gets large enough the algorithm analyses the frame regardless.

3 Contours

A limitation of CA is its sensitivity to damaged contours, damaged meaning image objects whose boundary has been distorted due to stains, dirt or colour loss. This cause 2 different types of contour damage: bleeding and broken contours. Bleeding contours refers to when different objects appear connected, making them seem as one object with a large boundary while broken contours is when gaps or holes occur. Both type of damage changes the contours connectivity and it only takes a small amount of damage to the suit and value to make the algorithm unreliable [Torgashov, 2014]. This often leads to the recognition failing or worse, false positives. The problem is to design and improve the algorithm to handle the different types of damage, which might require different approaches, while remaining fast and accurate.

3.1 Binarization

The work and research that has been done for the domain of scanning old historical documents can also be applicable here. Binarization methods used in that field need
to account for severe degradation in an evenly lit stable environment and while the degradation here is not as severe, the methods used can still be applicable. Most of the binarization processes use various forms of thresholding which classifies pixels as foreground or background to create a binary image. The many variants of thresholding can be grouped into 3 types; global, local or mixed. The difference is how they calculate the threshold value; global methods uses the entire image for every pixel while local uses different ones for different parts of the image and mixed methods uses a combination of both. They all have different properties; global being the fastest and most efficient and local being more precise [Kale et al., 2015].

The current algorithm uses the following methods to binarize each frame from the 30 FPS video stream:

1. Read the frame and convert it to grayscale.
2. Down- and up-sample the image to remove noise.
3. Create two binarized images using local thresholding and Canny edge detection.
4. Combine the two by using bit-wise OR

After these steps the contour comparison process starts which is based on the work of Pavel Torgashov. The underlying theory and techniques used are described in great detail in his article [Torgashov, 2014]. While the recognition methods are not the focus of this paper, it is important to understand a few of their properties. In this setting contours are represented and stored as vectors of complex numbers and the comparison function utilizes certain mathematical properties that this data representation has to ensure that scale, transposition and rotation are invariant to the process [Cardoso et al., 2015]. By representing contours as vectors and not 2D objects the complexity of these operations are reduced and allows CA to be used for real time applications, even if the hardware is limited. But while the recognition in CA is efficient, the current binarization process is not as it uses the combination of two local methods. The resulting binarized image also contains a lot of noise due to the poor quality of the camera and improving this step could save time.
Otsu’s method is a well established global thresholding algorithm. The method works by computing an optimal threshold based on the distribution of pixel values. It does so by creating a histogram and utilizing it to separate the pixels into two classes, foreground and background [Otsu, 1979]. This method is more general than a regular global threshold, as not all decks of cards have the same color saturation and does not require a predetermined threshold to be set. Empirical testing shows that this method produces good results in this type of environment compared to other methods [Sezgin and Sankur, 2004].

A limitation of this method is that it assumes a bi-modal distribution of gray values and it is sensitive if there is not a sharp contrast between objects [Kittler and Illingworth, 1985]. This is not an issue for most frames, as there is a high contrast between the objects of interest and the background (red and black figure on a white background). But while light is distributed evenly, the brightness intensity is often inconsistent between frames. Brightness is an issue as the distribution of pixel values gets spread out, making it difficult to set a global threshold. An increase or decrease of the brightness also causes the image to lose its contrast between the foreground and background. Applying Gaussian blur does help and give a smoother distribution of values but it does not help for extreme cases as seen in Fig 4, where part of the suit and value is considered background.
3.2 Morphology

While improving the binarization step helps to reduce noise and create binarized images where the foreground is highlighted, it does not fix the degradation issues. Connectivity problems are the focus here and as mentioned previously comes mainly from two sources; stains and gaps. Stains are only a problem if they are placed in an area that make separate objects appear connected. These type of stains are difficult to deal with, in particular if it is smeared out evenly with a dark colour.

Morphology is a group of different image processing techniques that deal with the shape of features in an image [Goyal, 2011]. The techniques are all based on erosion, dilation and combinations of the two. Dilation adds pixels to the boundaries of objects in an image, while erosion removes pixels. Closing is the act of applying erosion after dilation, which results in removal of gaps and small holes. This approach is sensitive to clusters of noise, as those clusters will connect and form a larger object. To avoid this they need to be handled prior to usage, which can be done by applying a blur filter on the image. A variant is to apply erosion prior to closing and then a dilation, but more operations risk finer details of the image to be lost. The operations uses a structuring element in the form of a convolution matrix, which can be of different shapes and sizes. The general rule of thumb is the smaller the matrix the less information is lost. There are no specific guidelines on how to apply these techniques as it depends on the environment. The shape, size and thickness of the object needs to be taken into consideration when choosing methods, which often requires testing using trial and error.

4 Colour

No part of the current algorithm checks or analyzes the colour of the card. This is information that could be applied to help identify cards as they are either black or red, depending on their suit. The ideal method would be one that not only is accurate, but
manages to save time compared to the current algorithm. The simplest method that exists today is to take each channel of the RGB image and threshold them. Depending on the threshold range, this method can identify various different colours. While operating on the RGB colour model reflects how the camera stores the data, it doesn’t correspond to how humans recognize colour [Huntsberger et al., 1985]. Another limitation with RGB is that the correlation between the channels is very high. This means that certain traits, such as intensity and brightness, impact all three channels accordingly [Cheng et al., 2001]. While RGB thresholding works fine in environments with high contrast and evenly coloured areas, it will not be reliable here due to inconsistent brightness between frames.

4.1 HSV

HSV is a colour model that has the following components:

1. Hue (Dominant wavelength)

2. Saturation (Purity of a colour)

3. Value (Brightness or intensity)

Hue represents a basic colour using a colour wheel where the values go from 0-360. Saturation describes the amount of white light that is mixed in with the hue. It goes between 0-100, which represents the percentage of white light. Value works in conjunction with the saturation and represents how much brightness a pixel has between 0-100, where 0 is completely dark and 100 got the most colourfulness. Since HSV separates the image intensity from the colour information it makes thresholding much more accurate, as brightness becomes an invariant. Regardless if the image plane is bright or dark, it will still represent the same part of the hue component contrary to RGB, where each channel get affected. This is very important in this environment, where brightness is often inconsistent albeit evenly distributed.
As seen in Fig 6, utilizing the properties of HSV enables finding red pixels regardless of brightness. There are however a few issues with this approach. Thresholding the entire image in search for red pixels is not efficient and require a lot of matrix operations that have a high complexity. A metric must also be derived to enable comparisons. Summing up the pixels would give a rough estimate that is invariant to intensity. But since certain deck of cards can introduce red in their design (in particular for Jack, Queen and King as they have small portraits) or be very minimalistic, a general metric will be difficult to set and will have a risk for false positives. Noise also occurs from light and plastic within the machine that is in the cameras view.

To make the most of the information, colour detection should happen prior to the comparison of found contours and templates, as filtering out those which are not of the correct colour will speed up the algorithm and raise its accuracy. Applying colour detection after this step could be done for defensive purposes and would only require analyzing a single contours intestines (the pixels within an enclosed contour). Depending on the probability of this occurring and its ability to catch false positives, as it can not differentiate between false positives of the same suit, this step would only marginally raise accuracy and not be worth the added run-time as extracting contour intestines is also a complex task.

Another method would be to only check the middle of the image for colours. This would remove the noise occurring outside of the card and be a faster method, as fewer pixels lowers the run-time. The important details tend to be in the center of the image and unlike detecting contours, checking for colour does not require that the entire suit and value to be within the area. To set the threshold of the amount of non-zero pixels which are required to consider the image red, a lot of data and testing is needed.
5 Discussion & Results

Testing of the various methods were done on the machine whose hardware specification can be found in Sec 1. As it is not tested on the card-dealing machine itself the measured timings do not accurately reflect the real run-time, but enable comparisons of different methods.

5.1 Blur detection

This experiment calculates the variance of the Laplacian as described in Sec 2. The design of a card impacts the variance of the Laplacian as decks with minimalistic design will naturally have a lower amount of edges. A general threshold needs to be set relatively low to account for this and early testing shows that 200 is accurate, but more testing is required to find the perfect threshold value. The result can be seen in Fig 7. This method excels at quickly discarding very distorted images with no clear foreground object. A limitation is its inability to discard all images with partially covered cards, as the part that is not covered will appear clear to the camera and may raise the variance. The accuracy of this method is hard to measure, but as seen in Fig 7, image 2:3 appears...
slightly distorted but still has a high enough value (233.76) to be considered for further processing.

<table>
<thead>
<tr>
<th>Row:Column</th>
<th>Normal algorithm</th>
<th>With Laplacian</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:1</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>1:2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>1:3</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>2:1</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>2:2</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>2:3</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>3:1</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>3:2</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>3:3</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

The implementation used for the run-time comparison does not utilize any sort of parallelization, which could speed up this algorithm drastically. It can be concluded with this early testing that it is efficient at discarding blurry images. But since distorted images lack distinguishable objects, the current algorithm will naturally detect fewer contours and thus finish relatively fast. More contours means more relatively complex operations (comparisons to the given template) so the time saved will be less than discarding normal images. The number of instructions performed by this method is approximately equal for images of the same size, which all in the table above are. Depending on the probability of distorted images, it is currently unclear whether or not adding this step would save time when sorting a complete deck.

Figure 8: An example of a false positive that could get discarded with this method

This method accurately removes distorted images and can be tuned to be more or less sensitive. It is invariant to illumination, brightness and damaged contours. It will lower the risk for false positives by removing distorted images as seen in Fig 8. The amount of time saved or lost using this method will depend on the probability of distorted images
occurring. This method is highly parallelizable and could be optimized to save more time if the underlying hardware supports it.

5.2 Efficient binarization

<table>
<thead>
<tr>
<th>Contour extraction average run-time after 100 runs</th>
<th>Original</th>
<th>Otsu’s</th>
</tr>
</thead>
<tbody>
<tr>
<td>404ms</td>
<td>101ms</td>
<td></td>
</tr>
</tbody>
</table>

Each run in the test above processed and extracted the contours from 12 images. The average run time per frame is 33 ms using the original method and 8 ms using Otsu’s method. But since Otsu’s method is sensitive to brightness, a combination of the two is proposed. By running the algorithm with Otsu’s method first and using the old method as a backup if it fails, time could be saved if these occurrences are rare. More data is required to calculate the expected value of the average run time using this implementation.

5.3 Improved contour extraction

Both types of contour damage (broken and bleeding contours) require different methods to handle them. But those methods are often specific for that type of damage and worsen the condition for other types of damage. Closing is very effective at closing gaps, but risks enlarging stains and connect them to other objects. Erosion makes uneven stains lose their connectivity, but also enlarge gaps. The solution is to extract contours from both variants of the binarized image. The proposed algorithm does the following steps:

1. Read the frame and binarize it

2. Apply closing and erosion separately, creating 2 binary images from the original one.

3. Extract contours from both and combine the results

4. Use the original CA methods to try and identify any contours with this combined list of contour vectors.

The point of doing this is to generalize the algorithm as much as possible to all types of degradation. The only case it can not handle is when an image has an object that suffers
from both types of degradation, these are however rare. While this does generate more contours, this process could be easily parallelized (as both outputs could extract and filter contours concurrently and join together once the process is done). The structuring elements are of the size 3x3 for the erosion to avoid removing anything but the outer layer and scattered noise while the closing is 7x7 to fill in the gaps properly. Such a big structuring element for closing would not be possible unless this general approach is used, as it would significantly worsen the quality of cards that are stained.

Figure 9: Erosion manages to remove enough pixels to separate the objects.
Figure 10: Closing here gives very good results, successfully removing the gap in the zero.

As seen in Fig 9, erosion does not remove the stain in full, it simply creates a small gap so the algorithm can distinguish the objects as separate. As previously mentioned, closing makes the problem worse by enlarging the stain by removing the small holes in it. In Fig 10 the gap is removed by closing. The bigger the gap is the bigger the structuring element needs to be, so while it works for this case its not guaranteed to work for very large gaps. The easiest gaps to fill are ones that are not completely white and still got some scattered pixels of colour remaining.

<table>
<thead>
<tr>
<th>Contour extraction average run-time after 100 runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
</tr>
<tr>
<td>404ms</td>
</tr>
</tbody>
</table>

Each run in the test above processed 12 images using the same data from the previous test. The average contour extraction time with morphological operations included is 15 ms using Otsu’s method and 46 ms using the current algorithm. This test does not account for the additional run time during the comparison, as it requires additional change to handle duplicates. As it will take additional time, the morphological methods can be introduced if the first attempt fails. Note that Otsu’s method in combination with the morphological operations only take around half of the time than the original method, making it a very fast and efficient method at handling degraded cards. These operations are also highly parallelizable and could be optimized further.
5.4 Colour recognition

Figure 11: Frames that are centered (1/2 of the image left) prior to thresholding

Testing is done with the two different methods to determine the time difference; Thresholding the entire image and basing the metric on the amount of red pixels found versus only thresholding the middle. The middle in this case will be considered 1/2 the image, which is achieved by splicing off 1/4 from every side, essentially removing everything that could be considered outside the card and focus only on where the suit and value tend to be. As seen in Fig 11, parts of the suit and value still remain in the image while a lot of noise is removed. Further testing is required to set the splicing to a large enough level to avoid misidentification. This test was run 100 times with 12 different images (seen in Fig 11) to get an average run-time.

<table>
<thead>
<tr>
<th>Run-time average after 100 runs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Full image</td>
<td>Center</td>
</tr>
<tr>
<td>274ms</td>
<td>75ms</td>
</tr>
</tbody>
</table>

By reducing the image by half, the run-time gets non-linearly shortened as the time complexity of the operations is $O(w \times h)$, where $w$ and $h$ stands for the width and height of the image. The center method is around 3.5 times faster with an average time of 6 ms per image versus 22 ms (as every run processed 12 images each). Early testing shows
that the threshold to determine whether or not an image is red would be around 2500
pixels, as it still needs to be high enough to filter out any type of noise occurring from
light reflecting on the cards white background.

6 Conclusion

This thesis has researched 3 potential areas of improvement for the card dealing machine.
The machine’s high demand for speed and accuracy requires that any proposed method
can justify its value for its run-time cost. The following research questions are derived
given these requirements:

- Would color recognition improve this system and where would it be implemented
to have the most impact?

- What types of methods could be added to a real-time OCR algorithm to handle
different types of degradation?

- Will the system be more accurate and faster by discarding distorted frames?

By analyzing each problem and researching about its related work, 4 methods has been
found which fulfill the high demands of the system. Testing was conducted to not only
showcase the methods values, but to find and propose efficient implementations in the
existing system that limits any potential downsides the methods might have
A metric based on the variance of the Laplacian can be used to measure an images sharp-
ness. It is a quick and efficient way that does not add a lot of time to the system. But
since distorted images naturally have less distinguishable contours, the current algorithm
processes them quite fast making the time saved fewer than expected. Further testing is
requires to establish whether or not the system would be faster, but the accuracy would
improve by removing low quality frames.
Morphological operations are efficient methods to deal with different types of degrada-
tion and the resulting contour damage. Closing is used to fix broken contours and erosion
is used to separate objects connected by stains. By implementing these two methods in
the preprocessing step, the system can be generalized and process frames it previously
could not. Otsu’s thresholding is a fast binarization method that early testing shows is 4
times faster than the old binarization method (Canny and adaptive thresholding). But
since a limitation is its inability to handle bright images, the proposed implementation
suggests using the old method as a back up if Otsu’s method fails. More data is needed
to establish if time is ultimately saved.
By converting the original RGB image to HSV, a quick metric can be devised based on
the number of red pixels found by thresholding the hue channel. But relative to the
current algorithm, this would be a costly method with little value other than checking
if the suit is correct. The proposed method attempts to make it faster by splicing the image and only checking the middles and implementing it prior to the contour comparison step, to speed it up by a factor of four by limiting the possible suits by half. While faster, the impact of such a method requires further testing as the probability of false positives of the opposite colour is unknown.
References


