Reconstructing shredded documents

Razvan Ranca

4th Year Project Report
Artificial Intelligence and Computer Science
School of Informatics
University of Edinburgh

2013

Abstract

Table of Contents

Chapter 1

Introduction

Chapter 2

Literature Review

Chapter 3

Probabilistic Score

This paper proposes a departure from the previous cost function definitions, looking instead at using a probabilistic model to directly estimate the likelihood of two edges matching.

3.1 Motivation

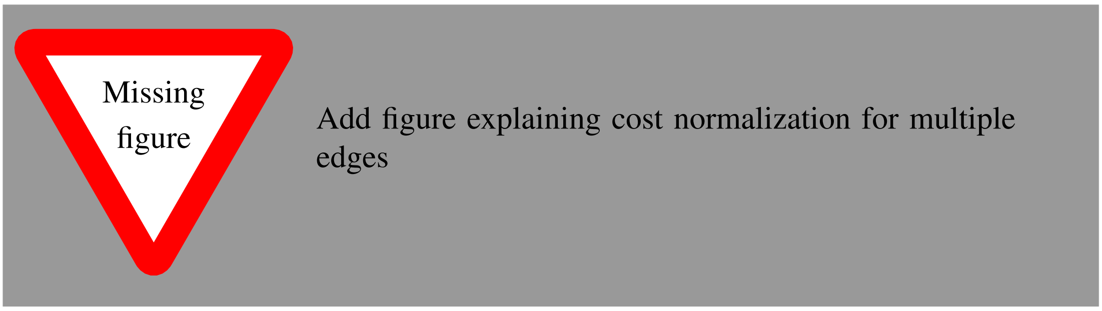
A pervasive problem with the cost functions discussed above is that their design is ad hoc and relies on hand-picked values based on the authors' empirical observations. For instance, in [6] the authors have to decide on the size of the Gaussian window they employ, on the weights to assign to the pixels that fall within that window and on a suitable value for their threshold function. In [9] the authors not only face all of the above problems, but also have to decide on a threshold value for their row comparison and on another threshold regarding the minimum amount of black pixels that an edge must have in order for it's matches to be considered relevant.

Explain these cost methods in the lit. review

This ad-hoc formulation suffer from several impediments which the probabilistic method manages to avoid or at least ameliorate:

Cost function	Probabilistic score function
Relies on the values the authors hand-picked for their particular dataset	Learns a new document's pixel distribution by analysing the shreds it is given
Cannot be easily combined with a different similarity function, as it lacks modularity	Can be easily composed with any similarity function that produces a probability. Several such functions already exist, such as the optical character recognition based system proposed in [4])

Cost function	Probabilistic score function
Difficult to evaluate results of the function since the numbers outputted are meaningless, only the order of the results matters	The scores returned are the estimated probabilities of matching, so evaluation is natural, by comparing estimated and observed probabilities.
Cost is additive and must therefore be normalized relative to the sum of lengths of the matching edges	All scores are normalized probabilities, so length of edges or number of edges matching is irrelevant (see Figure x)



Add figure explaining cost normalization for multiple edges

Lastly, the effectiveness of probabilistic models, when applied to text data, has been repeatedly shown. Pixel prediction goes back to systems such as the 1981 JBIG lossless compression ISO standard(which tried to predict the value of a pixel using several features including 6 of it's neighbours [2]) and is still employed in today's state of the art encoders [1].

3.2 Description

The central idea of the probabilistic function is, when looking at a proposed match, to estimate the probability that a candidate pixel is correct given several of its neighbour pixels(i.e. the candidate pixel's "context", see Figure 3.1). We refer to this conditional probability as $\Pr(p \mid C(p, E_x))$, where $C(p, E_x)$ is the context of pixel p when placed next to edge E_x. Ideally we'd want to learn these conditional probabilities by analysing the original document, which we obviously don't have access to. However, relatively few pixels are destroyed when shedding a document and we can assume that the ones that are destroyed are uniformly distributed over the space of contexts. Therefore the cumulative distribution of the shreds will, in general, be a close approximation to the distribution in the original document, which means we can get a good estimate of the needed probabilities by analysing each individual shred and averaging the results.

3.2.1 Edge likelihood

Once we have estimated the conditional probabilities, the total probability of two edges matching can be calculated by sliding the context down the proposed edge and multiplying all the individual candidate pixel probabilities. That is to say, if we have a

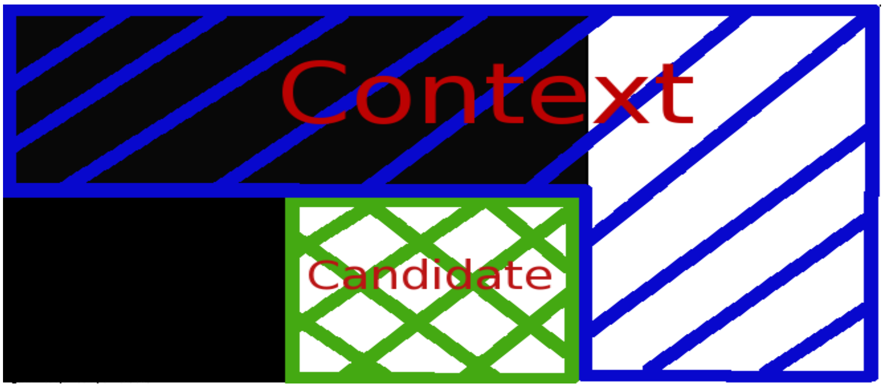


Figure 3.1: This shows a proposed match between the top 3 and the bottom 3 pixels. The model estimates the probability that the candidate pixel is white based on the four context pixels

proposed matching between edge E_1 and edge E_2, and E_1^x represents the xth pixel on edge 1, then we would estimate:

$$\Pr(E_1, E_2) = \prod_{i=1}^{len(E_2)} \Pr(E_2^i \mid C(E_2^i, E_1))$$

A problem arises though, because the shape of the context prohibits us from taking the conditional probability for the first and last few pixels (depending on the length of the context). For these the best we can do is assign them the prior probability, $\Pr(p)$, which is also extracted from the data simply by counting the proportion of white and black pixels. Therefore, when using the context of size 4 shown in Figure 3.1, the edge probability will actually be[1]

$$\Pr(E_1, E_2) = \Pr(E_2^1)\Pr(E_2^{len(E_2)}) \prod_{i=2}^{len(E_2)-1} \Pr(E_2^i \mid C(E_2^i, E_1))$$

3.2.2 Normalization

The above, while being a good measure of the likelihood of an edge, is a function that will tend to decrease as $len(E_2)$ increases. The function will therefore tend to prefer shorter edges with less probabilities to multiply. Luckily, using probabilities gives us one more piece of information here. In the original file, every edge of every shred has only one correct match, since no two shreds are superimposed. Therefore, we know that the sum of the probabilities of all matches along one edge should sum up to one. In technical terms, if S_E is the set of all edges, then:

$$\forall E_a \sum_{E_x \in S_E} \Pr(E_a, E_x) = 1$$

[1] There actually is a further problem with the presented formula, namely that it doesn't account for edges belonging to the same piece. If E_1 and E_2 both belong to the same shred then obviously the probability that they match is 0 since a shred can't be in 2 places at once. This extra complication is ignored in the mathematical formalism for reasons of clarity

In order to satisfy this constraint the probabilities are normalized along every edge. After the normalization, the preference for shorter edges is eliminated and the probabilities for different edges and pieces can be directly compared regardless of local features such as a particular edge length. Therefore, accounting for the normalization process, the final definition becomes:

$$\Pr(E_1,E_2)=\frac{\Pr(E_2^1)\Pr(E_2^{len(E_2)})\prod_{i=2}^{len(E_2)-1}\Pr(E_2^i\mid C(E_2^i,E_1))}{\sum_{E_x\in S_E}\Pr(E_x^1)\Pr(E_x^{len(E_x)})\prod_{i=2}^{len(E_x)-1}\Pr(E_x^i\mid C(E_x^i,E_1))}$$

The normalization discussed above has an additional property, which ends up mitigating the predisposition towards whitespace that some of the other cost functions suffer from. The property in question is that if E_1, E_2 and E_3 are very similar or identical edges then they will have less chance of being picked as the best match because the available probability mass of any match will be split equally among the 3 of them. Formally, if I_{E_x} is the set of edges identical to E_x then:

explain this predisposition in lit. review

$$\forall E_a \Pr(E_a,E_x)\leq\frac{1}{\mid I_{E_x}\mid}$$

Since there are usually several white edges floating around in the edge set at any time, the above property ends up naturally discounting them without the need for any ad hoc heuristics (see Table 3.1 for a typical situation). To further discount the white on white matches we can introduce dummy white pieces into the shreds pool.

	Raw probabilities			Normalized probabilities		
	Edge 3	Edge 4	Edge 5	Edge 3	Edge 4	Edge 5
Edge 1	0.90	0.90	0.20	0.45	0.45	0.10
Edge 2	0.10	0.10	0.50	0.14	0.14	0.71

Table 3.1: Here Edge 1, 3 and 4 are all white and as such have a very good raw score when matched together. After normalization however, the match between edge 2 and 5 is considered a better bet than any match containing edge 1, as edge 1's probability mass is distributed equally among the identical edges 3 and 4.

3.2.3 Learning

Given the above descriptions, the problem of calculating the probability of any two edges matching can be reduced to the problem of estimating $\Pr(p\mid C(p,E_x))$ for an arbitrary pixel an it's context. Depending on the size of the context various machine learning methods could be used to accomplish this task. However, if the context is of the form shown in Figure 3.1, then a simple exhaustive exploration of the context space becomes a possible solution. If the context is of size 4, as above, and we are working on black and white documents, then there are only $2^4=16$ possible contexts. Since the number of contexts we can sample is directly proportional to the pixel count of the source image, for an average one page document, the number of observations will exceed 1 million. In practice this means that all 16 possible contexts are well represented in the document(a document where any of the contexts had less than 250 observations hasn't yet been encountered).

Of course the fit to the data will be limited by such a small context, however care must be taken when extending the context as it can easily lead to over-fitting. For instance using a context of size 7 instead of 4 usually leads to having several contexts with a number of observations in the single digits. Such values can, of course, not be trusted. Therefore it seems necessary to use more sophisticated methods in conjunction with a larger context.

One such attempt was made by using neural networks. Different architectures were tried, with both contexts of size 4 and 7, however no significant difference was found when compared with the above direct estimation. Therefore the method was abandoned as the longer training time could not be justified.

compare NN(perhaps with and without hidden layer) output to some evaluation measure used below

3.3 Evaluation

The evaluation measures used here all work by comparing the edges predicted by the algorithm to the real edges. In order to obtain the predicted edges, for every edge we simply take the most likely pair edge. If $PredMatch(E_a)$ is a function that returns E_a's predicted match, then:

$$PredMatch(E_a) = \arg\max_{E_x} \Pr(E_a, E_x)$$

3.3.1 Comparison with Gaussian cost functions

In order to perform this comparison, a function $CorrMatch(E_a)$ is defined, which returns the correct match for edge E_a. With this function we can define the $score(E_a)$ function as:

$$score(E_a) = \begin{cases} 1 & \text{if } CorrMatch(E_a) = \arg\max_{E_x} \Pr(E_a, E_x) \\ 0 & \text{otherwise} \end{cases}$$

However, there's a mistake here. The problem is that $\arg\max_{E_x} \Pr(E_a, E_x)$ is not guaranteed to return a single element. What we really need to check for then is $CorrMatch(E_a) \in \arg\max_{E_x} \Pr(E_a, E_x)$.

This change allows for multiple maximum likelihood matches, but doesn't account for the increased probability that the correct edge is within the predicted set by chance. In the extreme case, if the probability score returned the same value for every edge, then this scoring function would judge any pairing as correct. The solution is to discount the score based on the size of the predicted set (which also makes intuitive sense, since when a search is actually performed, if there are multiple matches with the same probability, the search will simply have to pick one at random). The final score function is:

$$score(E_a) = \begin{cases} \frac{1}{|\arg\max_{E_x} \Pr(E_a,E_x)|} & \text{if } CorrMatch(E_a) \in \arg\max_{E_x} \Pr(E_a,E_x) \\ 0 & \text{otherwise} \end{cases}$$

Therefore the proportion of correctly predicted edges (where S_E is again the set of all edges) is:

$$predictedCorrect = \frac{\sum_{E_x \in S_E} score(E_x)}{\mid \{E_a \mid CorrMatch(E_a) \in S_E\} \mid}$$

Here the denominator is just counting the number of edges that actually have matches. This is necessary because using $\mid S_E \mid$ instead would also count the outer edges which have no correct match.

Calculating analogous measures for the cost functions defined in [6] and [9] allows us to compare the three methods and shows the probabilistic score as having a consistent advantage over the other functions(see Figure 3.2).

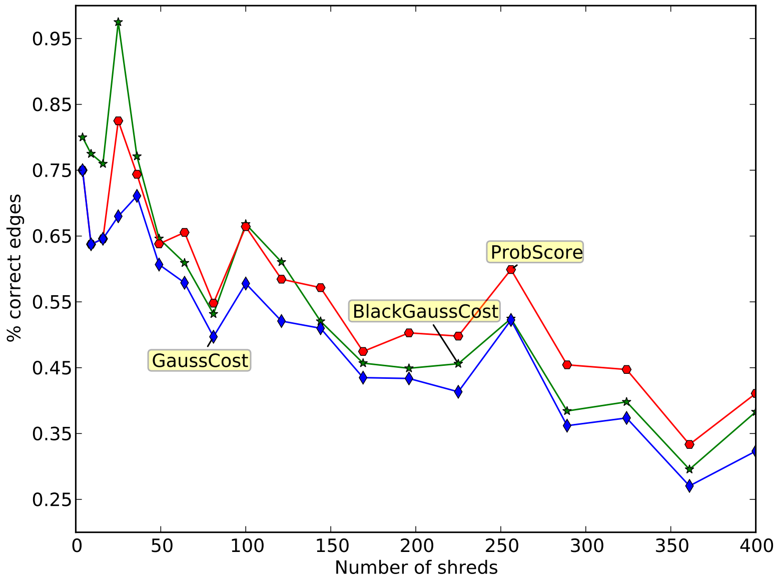


Figure 3.2: The probabilistic score has better results on medium and large instances than the cost functions presented in [6] (GaussCost) and [9] (BlackGaussCost)

3.3.2 Validity of predicted probabilities

As mentioned in the Motivation, there is another(perhaps more natural) evaluation that we can perform. Namely comparing the predicted and observed probabilities for our predicted edge matches. The observed probabilities are calculated exactly as in the previous section, and the predicted probabilities are recorded for every predicted match.

The formula $bucket = \lfloor prob * 10 + 0.5 \rfloor$ is used to place the predicted probabilities into one out of a total of 10 buckets. The observed and predicted probabilities are then averaged within each bucket, and the process is repeated for different number of shreds. The resulting graph is Figure 3.3.

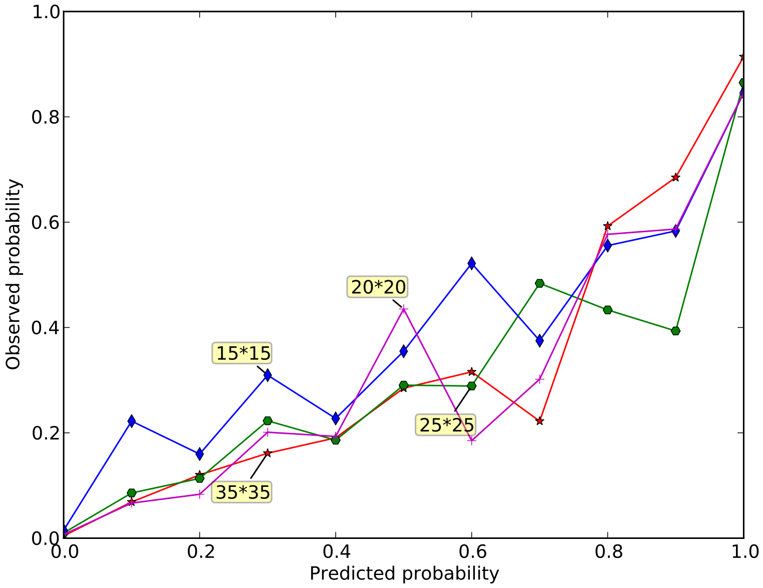
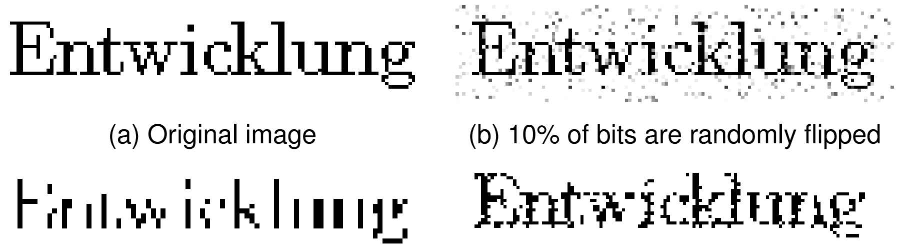


Figure 3.3: The labels show the number of cross-cur shreds. As the number of shreds increases the correlation between the predicted and observed probability diminishes. However the overall shape of the curve is still generally increasing, so a higher predicted probability will usually translate into a higher observed probability and for large predicted probabilities the method is quite accurate

3.3.3 Robustness

Another important aspect worth evaluating is the robustness of the method. When used in real life situations it is unlikely that documents will always be high-resolution, perfectly cut into shreds and completely smudge-free. In order to test this, the results obtained on an image are compared to those obtained when various types of noise are added to the image. The types of noise analysed are: downsampling, flipping random pixels and shuffling random pixels around their original position (see Figure 3.4).



(a) Original image

(b) 10% of bits are randomly flipped

(c) The image is downsampled by a factor of 1.5

(d) Pixels are randomly shuffled to one of their immediate neighbours

Figure 3.4: How one word of the image is modified by the various types of noise

The score evaluation is run on the original and modified documents, and the results are shown in Figure 3.5. As expected some performance degradation is observed in all cases. It is interesting to note that the largest reduction in performance is caused by the random pixel flipping. This can be explained by the fact that the other types of noise are restricted by the existing structure of the document, if you have a section of white pixels, neither downsampling nor shuffling can introduce a black pixel into it. The algorithm seems to suffer more from the unrestrained randomness exhibited by the flipping. The good news is that in real life scenarios most noise should not be completely random and we'd expect it to be more similar to the downsampling and shuffling types of noise than to the pixel flipping.

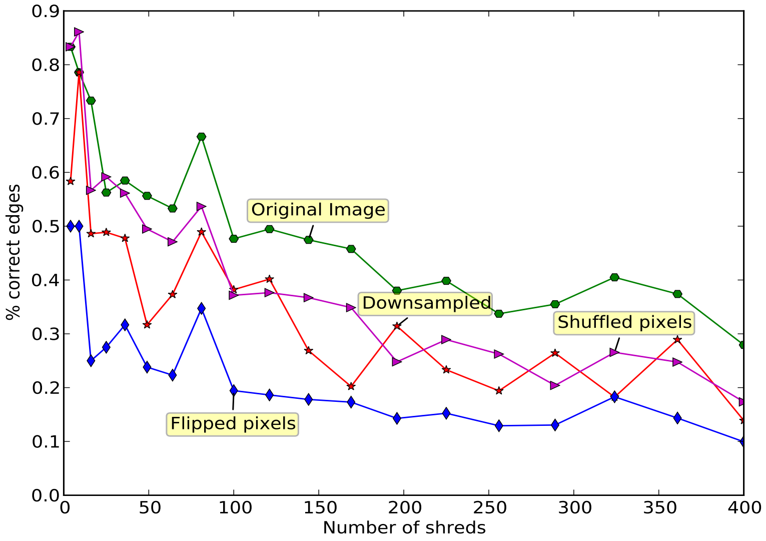


Figure 3.5: Degradation of performance between original image and 3 noisy images

3.4 Drawbacks

Finally, we'll look at some of the flaws in the probabilistic scoring function and at some possible solutions.

3.4.1 Uniformity assumption

Learning a single set of conditional values of the form $\Pr(p_i \mid C(p_i, E_x)$ implicitly makes the assumption that these values are relatively uniform over the whole document. This is usually a reasonable assumption to make if the documents we're interested in are all text. However this assumption is certainly not always justified. Consider the document from Figure 3.6. Here we can clearly see that there are two distinct regions, the text region and the table region. An algorithm which tries to fit a single valued model to this document cannot achieve very good results, because this document cannot be described with a single set of $\Pr(p_i \mid C(p_i, E_x)$ values. The problem will only be compounded if we're looking at multiple shredded pages from possibly different documents.

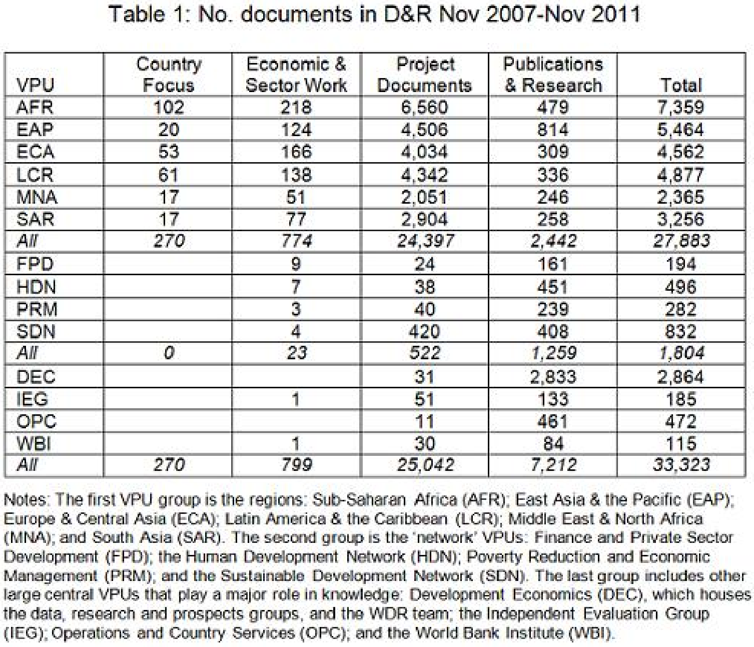


Table 1: No. documents in D&R Nov 2007-Nov 2011

VPU	Country Focus	Economic & Sector Work	Project Documents	Publications & Research	Total
AFR	102	218	6,560	479	7,359
EAP	20	124	4,506	814	5,464
ECA	53	166	4,034	309	4,562
LCR	61	138	4,342	336	4,877
MNA	17	51	2,051	246	2,365
SAR	17	77	2,904	258	3,256
All	*270*	*774*	*24,397*	*2,442*	*27,883*
FPD		9	24	161	194
HDN		7	38	451	496
PRM		3	40	239	282
SDN		4	420	408	832
All	*0*	*23*	*522*	*1,259*	*1,804*
DEC			31	2,833	2,864
IEG		1	51	133	185
OPC			11	461	472
WBI		1	30	84	115
All	*270*	*799*	*25,042*	*7,212*	*33,323*

Notes: The first VPU group is the regions: Sub-Saharan Africa (AFR); East Asia & the Pacific (EAP); Europe & Central Asia (ECA); Latin America & the Caribbean (LCR); Middle East & North Africa (MNA); and South Asia (SAR). The second group is the 'network' VPUs: Finance and Private Sector Development (FPD); the Human Development Network (HDN); Poverty Reduction and Economic Management (PRM); and the Sustainable Development Network (SDN). The last group includes other large central VPUs that play a major role in knowledge: Development Economics (DEC), which houses the data, research and prospects groups, and the WDR team; the Independent Evaluation Group (IEG); Operations and Country Services (OPC); and the World Bank Institute (WBI).

Figure 3.6: A document in which the uniformity assumption is wrong.

The obvious solution here is to make the probabilistic model complex enough to remove the uniformity assumption. In theory this could also help reduce the problem size, by separating the documents into smaller uniform regions. However, avoiding overfitting might be difficult in this scenario. For instance, we wouldn't want the model to decide that every boldfaced piece of text belongs together in a separate uniform region.

3.4.2 Lack of symmetry

If E_x and E_y are two edges and E_x^{rot} and E_y^{rot} are the previous edges rotated by 180° then we might reasonably expect that

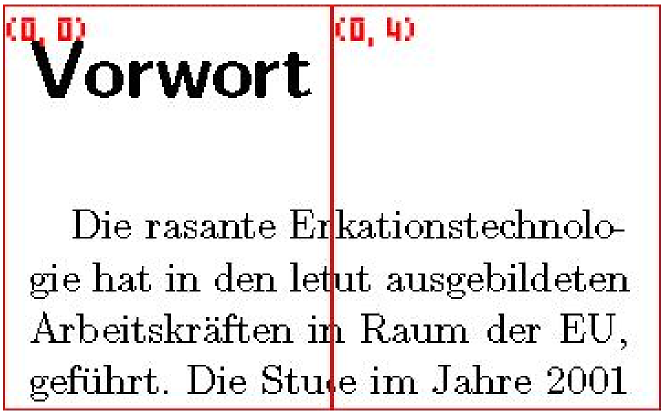
$$\forall E_x \forall E_y \Pr(E_x, E_y) = \Pr(E_y^{rot}, E_x^{rot})$$

This would be a good property to have, since we are essentially comparing the exact same edge matching in both situations. However, the current probabilistic model offers no such guarantee. In particular the probabilistic model cannot guarantee that $P(E_x, E_y) = P(E_y^{rot}, E_x^{rot})$ because it is not assured (and indeed quite unlikely) that enough data was present for these two conditional probabilities to have converged to the same value.

Depending on the situation this deficiency could be ignored. It is possible, however, that the search function being employed will assume the score to be symmetric. In that case, the best solution seems to be to calculate both $\Pr(E_x, E_y)$ and $\Pr(E_y^{rot}, E_x^{rot})$ and re-assign both probabilities to some number in between the 2 original probabilities (arguments could be made for either *min*, *max* or *avg*). Care should be taken that this re-assignment is done before the normalization step, otherwise the values will no longer add up to 1.

3.4.3 Ignoring non-edge information

This problem is common to both the probabilistic score and all the cost functions presented above. It is also the hardest to fix. Figure 3.7 shows an error made on a 5x5 cross-cut document. It is interesting to notice how even in a document cut into only 25 pieces such a convincing false match can occur. As the size of the shreds and therefore the amount of information available to our scoring function decreases further we can expect a huge number of such mistakes, making reconstruction of any non-trivial cross-cut document problematic.



Vorwort

Die rasante Erkationstechnologie hat in den letut ausgebildeten Arbeitskräften im Raum der EU, geführt. Die Stue im Jahre 2001

Figure 3.7: An incorrect match that would be very difficult to detect by a cost/score function which only looks at the edge pixels

The solution here likely involves combining multiple scoring functions looking at different sets of features, both higher and lower level. Luckily, as mentioned above, the probabilistic score can easily accommodate any such scoring functions as long as they can express their result as a probability. A few possible higher level scoring functions are discussed *somewhere*.

link to that part

Chapter 4

Heuristic Search

4.1 Motivation

As discussed in the literature review, many complex methods have been applied to the problem of searching for the best arrangement of shreds. These methods take varied approaches such as reduction to a travelling salesman problem and making use of industrial solvers([6]), variable neighbourhood search([6, 7, 8]),genetic algorithms([8]),ant colony optimization([7]) and reformulation as an integer linear programming problem([5]).

add more detail about all of these in lit. review

However, at least in the first instance, we decide to focus upon relatively simple greedy heuristics. Some reasons for this decision are:

- The heuristics run significantly faster than the aforementioned methods. This allows for ease of prototyping and the ability to easily experiment with many variations on both the search and cost functions

- The inner-workings of the heuristics are transparent. This transparency allows for ease in diagnosing problems with either the heuristics themselves or with the cost functions. The transparency also allows for easy visualisation of every individual search step which enables us to observe exactly how errors occur and are how they are propagated. In contrast, using some of the previous methods would make it difficult to diagnose, for instance, whether an error is caused by the search or the cost function.

- Any advanced search methods need to be compared against a solid baseline. These heuristics provide such a baseline.

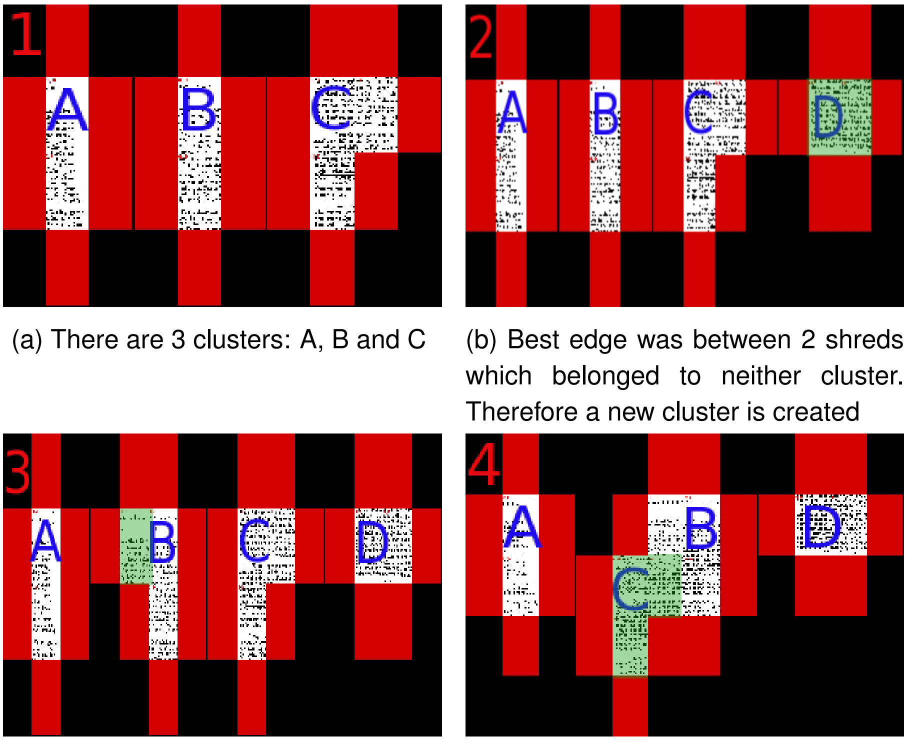
- Surprisingly good performance has been obtained using greedy heuristics. For instance, [9] reports that their simple heuristic search managed to outperform both a variable neighbourhood search and an ant colony optimization method.

4.2 Description

Despite the fact that most of the previous work has focused on complex search algorithms, a few greedy heuristics have been explored before. We first re-implement 3 of these heuristics which, in growing order of complexity, are: the ”Row building” heuristic([7]), the ”Prim based” heuristic([7]) and the ”ReconstructShreds” heuristic([9]). More details about these techniques are available in the literature review section. Additionally, we implement a fourth, novel, heuristic which is an extension of the ”ReconstructShreds” version.

add this to lit. review.

The central idea behind the ”ReconstructShreds” heuristic is that, at each search step, the best available edge should be added to the partial solution. A few steps of the execution of this algorithm are examined in Figure 4.1, where all groups of shreds of size 2 or larger are shown (i.e. the individual pieces, which are groups of size 1, are omitted from the figure). The algorithm will continue to add the best possible edge to the partial solution and thus enlarge and merge clusters until only 1 cluster is left. This final cluster is the algorithm’s final solution.



(a) There are 3 clusters: A, B and C

(b) Best edge was between 2 shreds which belonged to neither cluster. Therefore a new cluster is created

(c) Best edge was between a shred belonging to cluster B and one belonging to neither cluster. Therefore cluster B is enlarged.

(d) Best edge was between a shred belonging to cluster B and one belonging to cluster C. Therefore the two clusters are merged.

Figure 4.1: Four steps from the middle of a ”ReconstructShreds” search are shown. The clusters are called A, B, C and D, the green pieces highlight the changes that occur in every step and the red pieces show the positions that are considered for the insertion of new shreds.

In order to explain the reasoning behind our new algorithm, first ”ReconstructShreds” must be more closely looked at.

4.2.1 Analysis of ”ReconstructShreds”

In order to achieve the desired behaviour, this algorithm calculates all the edge probabilities and then simply goes through the list in descending order. Whenever it encounters a valid edge, it adds it to the solution set (see Algorithm 1 for the pseudocode [1]).

Algorithm 1 The ”ReconstructShreds” heuristic

```
                                  ▷ Takes the set of edges and the set of shreds as input
procedure RECONSTRUCTSHREDS($S_{edges}$, $S_{shreds}$)
    $probs \leftarrow []$               ▷ Initialize 2 empty arrays for the probabilities and edges
    $edges \leftarrow []$
    for all $E_x \in S_{edges}$ do
        for all $E_y \in S_{edges}$ do
            $probs[(E_x, E_y)] \leftarrow \Pr(E_x, E_y)$ ▷ Calculate and store all the probabilities
        end for
    end for

    $setsLeft \leftarrow |S_{shreds}|$       ▷ Initially every shred is it’s own set, initialize these
    for all $S_x \in S_{shreds}$ do
        $InitSet(S_x)$
    end for

    while $setsLeft > 1$ do                ▷ Get the edges with the max probability
        $(E_x, E_y) \leftarrow \arg\max_{(E_x, E_y)} probs[(E_x, E_y)]$
        $S_x \leftarrow GetSet(E_x)$                    ▷ Retrieve the sets of these 2 edges
        $S_y \leftarrow GetSet(E_y)$
        if $S_x \neq S_y$ & $mergePossible(E_x, E_y)$ then
            $S_x \leftarrow Union(S_x, S_y)$          ▷ If the edge is valid, merge the two sets
            $S_y \leftarrow Union(S_x, S_y)$
            $edges.append((E_x, E_y))$
            $setsLeft \leftarrow setsLeft - 1$
        end if
        $probs[(E_x, E_y)] \leftarrow 0$       ▷ make sure the processed edge isn’t picked again
    end while

    return $edges$          ▷ The set of returned edges describes a complete solution
end procedure
```

[1] The pseudocode in this chapter assumes the existence of a disjoint-set data structure which can perform operations such as $InitSet(x)$(create a set with the element x in it), $GetSet(x)$(return the set to which x belongs) and various set operations such as $Union(S_x, S_y)$ and $Intersect(S_x, S_y)$

The problem with this approach is that the probabilities used are static and therefore the algorithm is completely ignorant of the formed partial solution. It's only interaction with the current state of the solution occurs via the $mergePossible(E_x, E_y)$ function which just tells the algorithm if the current solution allows for that particular merge. In particular, the algorithm takes no account of the fact that when merging 2 sets of pieces, several new edges may form. Therefore the "ReconstructShreds" heuristic would be happy to merge two large sets of pieces based on 1 good edge resulting from the merge even if several other terrible edges also result from that same match (see Figure x).

Missing figure

add figure demonstrating the mentioned bad scenario for RS

Our algorithm is designed to address this shortcoming.

4.2.2 Kruskal based heuristic[2]

One approach towards resolving the problem observed in "ReconstructShreds" is to recalculate the probabilities of two edges matching at every iteration. By doing so we can take into account all the additional matches that would result when the sets corresponding to the 2 edges are merged(see Algorithm 2).

The differences between the "Kruskal" and the "ReconstructShreds" algorithms are:

- The probability calculation has been moved inside the *whileloop*, and thus our calculated probabilities need not be static any more.
- Rather than simply taking the pre-calculated probability $\Pr(E_x, E_y)$ as the final measure of the likelihood of a match between E_x and E_y, the helper function $getProb(E_x, E_y)$ is now called instead. This new function checks the proposed merge and identifies all the new edges that would be formed if this merge is selected (for convenience we assume the set of edges S_x has a function $S_x.neighbours()$ which returns all the pairs of edges that are neighbours under this merge). The final probability is then the given by multiplying the individual probabilities for every newly created edge(see Algorithm 3).

[2] This method is called a "Kruskal based heuristic" because the main goal of the method, namely that of always adding the best available edge to the solution, is analogous to the goal of the minimum spanning tree algorithm, "Kruskal"[3]. Therefore the general Kruskal method can be extended to this specific problem, with the only additional difficulty of having to reject potential matches which would result in 2 shreds overlapping. Indeed "ReconstructShreds" is already an extension of the Kruskal method, though the authors do not identify it as such

Algorithm 2 The Kruskal based heuristic

```
procedure KRUSKAL($S_{edges}$, $S_{shreds}$)
    $edges \leftarrow []$
    $setsLeft \leftarrow |S_{shreds}|$
    for all $S_x \in S_{shreds}$ do
        $InitSet(S_x)$
    end for

    while $setsLeft > 1$ do
        $probs \leftarrow []$                ▷ Probability calculation is done for every iteration
        for all $E_x \in S_{edges}$ do
            for all $E_y \in S_{edges}$ do
                $probs[(E_x, E_y)] \leftarrow getProb(E_x, E_y)$        ▷ Helper function is called
            end for
            $normalize(probs, E_x, S_{edges})$    ▷ An extra normalization step is needed
        end for
        $(E_x, E_y) \leftarrow \arg\max_{(E_x, E_y)} probs[(E_x, E_y)]$
        $S_x \leftarrow GetSet(E_x)$
        $S_y \leftarrow GetSet(E_y)$
        if $S_x \neq S_y$ & $mergePossible(E_x, E_y)$ then
            $S_x \leftarrow Union(S_x, S_y)$
            $S_y \leftarrow Union(S_x, S_y)$
            $edges.append((E_x, E_y))$
            $setsLeft \leftarrow setsLeft - 1$
        end if
        $probs[(E_x, E_y)] \leftarrow 0$
    end while

    return $edges$
end procedure
```

- A normalization step is added. This is necessary because, by potentially multiplying the probabilities of several edges together to get the probability of our match, the sum of all probabilities is no longer guaranteed to be 1. Formally, after the normalization step taken in the calculation of the $\Pr(E_x, E_y)$ values(see Section 3.2.2) we had the following assurance:

  $$\forall E_a \sum_{E_x \in S_E} \Pr(E_a, E_x) = 1$$

  We would like to have the same assurance regarding $getProb(E_x, E_y)$, which is why the additional normalization step is required. As before, the normalization step takes the form:

  $$normProb(E_x, E_y) = \frac{getProb(E_x, E_y)}{\sum_{E_a \in S_E} getProb(E_x, E_a)}$$

Algorithm 3 The getProb helper function

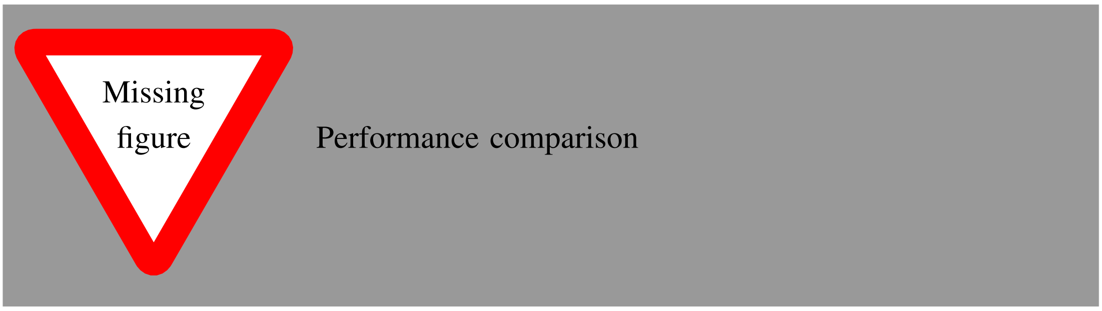
```
procedure GETPROB($E_x$, $E_y$)
    $prob \leftarrow 1.0$
    $S_x \leftarrow GetSet(E_x)$
    $S_y \leftarrow GetSet(E_y)$                    ▷ Get the set of the proposed match
    $merged \leftarrow Union(S_x, S_y)$
    for all $E_a \in S_x$ do    ▷ Multiply probs of new neighbours created by the match
        for all $E_b \in S_y$ do
            if $(E_a, E_b) \in merged.neighbours()$ then
                $prob \leftarrow prob * \Pr(E_a, E_b)$
            else if $(E_b, E_a) \in merged.neighbours()$ then
                $prob \leftarrow prob * \Pr(E_b, E_a)$
            end if
        end for
    end for
    return $prob$
end procedure
```

4.2.3 Evaluating the heuristics

This is a stub, don't have the figures yet

Finally we turn to evaluating the strength and weaknesses of the algorithms discussed above. Towards this purpose we run two sets of tests.

Firstly, look at the performance offered by the various search functions. This is accomplished by running all the heuristics on the same input document, using the same cost function and then comparing the number of correct edges observed in the output. The results can be seen in Figure xx



Performance comparison

Secondly, we analyse the scalability of the algorithms. In real world scenarios an unshredder would potentially have to work with thousands, or even tens of thousands of pieces, which makes scalability an important factor to consider. For comparison purposes, the runtime of some of the more complex search functions are also shown.

check which

Missing figure

Scalability comparison

explain why only Prim and Kruskal are used from this point forward

4.3 Cascading

All of the greedy methods presented thus far have in common an inability to correct errors. This makes them prone to a cascading effect through which an error made early in the search process can have a major effect on the result. Since there is no means through which to move the wrongly placed shred, the search will instead try to find additional pieces which match well with said shred, and these pieces will have a significant probability of also being wrong.

In order to quantify the magnitude of this issue it is helpful to plot the function: $Error_{search}(x)$, where $x = Error_{cost}$. This function shows the proportion of errors that the search function commits when given a cost function with a certain error rate. The results of this experiment(see Figure 4.2) are telling.

explain how this is done, especially that it's with artificial data

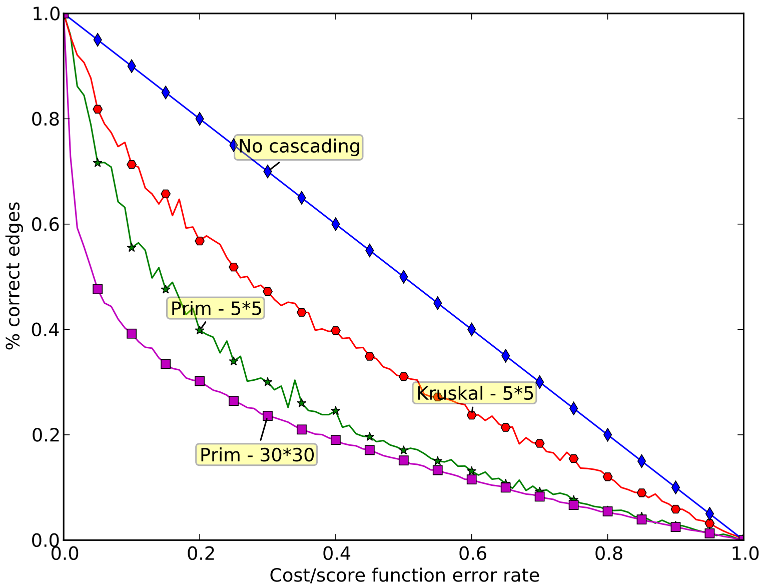


Figure 4.2: The effect the error in cost/score has on the final error of the method for both search heuristics on 5*5 shreds and for Prim on 30*30 shreds

Even on a tiny 5 by 5 instance, in order for the Kruskal heuristic to achieve 80% accuracy the scoring function must be 95% accurate. As can be seen, the problem only gets worse as the number of shreds increases. On a 30 by 30 instance Prim requires the same 95% cost function accuracy rate in order to achieve a final accuracy of only 50%.

In order to address this problem, several error correcting mechanisms were analysed.

4.3.1 Making all shreds movable

The simplest approach is to consider pieces that have already been placed as movable and treat them the same as the pieces which havent been placed. Using this new formulation in the above cascading experiment yields very interesting results(see Figure 4.3).

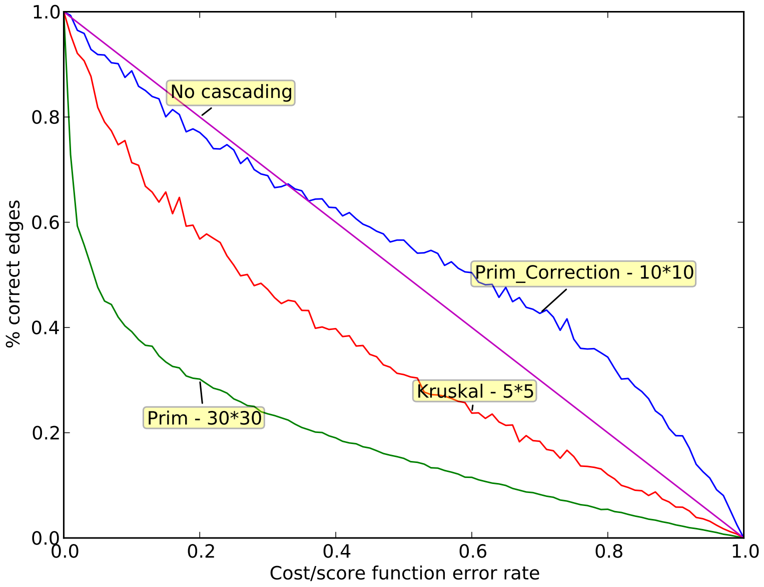


Figure 4.3: The effect the error in cost/score has on the final error of the method for the old heuristics and the new error correcting one

explain "S" shape of curve

In theory, this approach should solve the cascading problem. However, in practice, this basic approach can lead to an infinite cycle of moves if it happens that one piece is the best match for two different edges, as the greedy algorithm will continue to switch it between the two. This specific problem can be solved by giving the greedy correction algorithm a lookahead of size one, by only moving a piece if this would increase its probabilistic score. This solution however will only eliminate cycles of length 2, in order to eliminate all cycles in this way would require a complete examination of the search tree from that point onwards, which quickly becomes intractable.

explain locality of prob score which allows for infinite cycles to occur

In order to eliminate cycles while using a fixed size lookahead, all previous states that the search has passed through can be remembered and therefore cycles can be detected. Once a cycle is detected, the algorithm can choose to revert to the best state encountered within the cycle and then pick a move that breaks the cycle.

4.3.2 Adding a post-processing step

Another problem that plagues the heuristic search methods presented here is that while the search is in progress it cannot be known which pieces will end up on an outer edge and which will be in the centre. This means we cannot take account of the score of an edge piece adjacent to white space, which in turn places no incentive on the search to find compact shapes that have less pieces on an outer edge .

This problem can be ameliorated by doing a post-processing step to the search. When all the pieces have been placed we finally know what the proposed outside edges are, so we can now keep count of the external whitespace score and apply the same search and cycle detecting process described above. Since the search we perform is still greedy, in order for the method to find correct solutions, it will require a large lookahead (see Figure 4.4). This requirement again becomes quickly intractable. However, since all previous states are recorded, this post-processing step is guaranteed to obtain a final state thats either better or at least equivalent to the one it started with. This guarantee cannot be made for the previous correction heuristic.

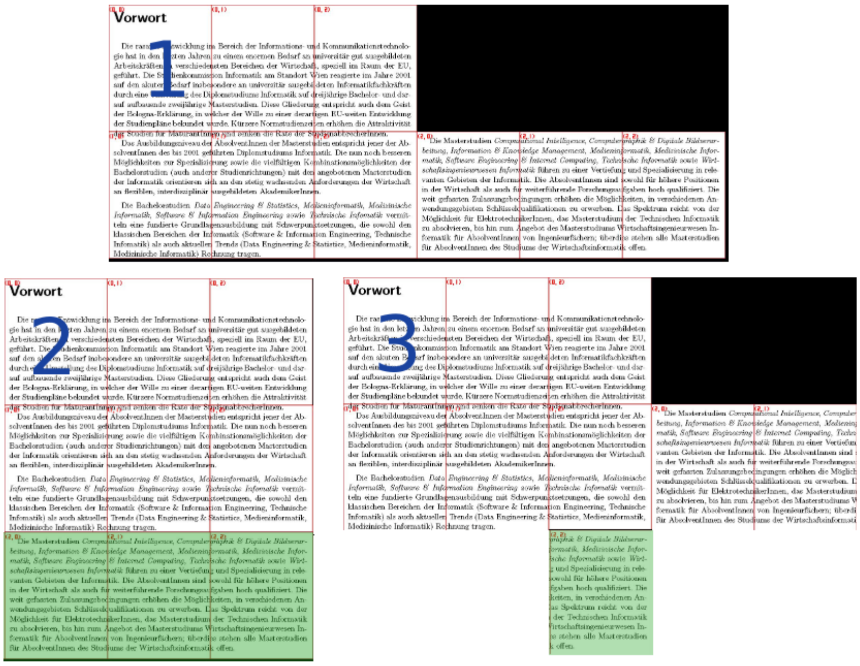


Figure 4.4: Since external whitespace is not counted towards the score, solutions 1 and 2 have the same score even though 1 has 16 external edges and 2 has only 12. In order to move from 1 to 2 the greedy search would have to pass through 3 which also has 16 external edges. In this case the transition to the correct result will only be made if the search can see at least 3 moves ahead

4.3.3 Evaluating the error-correcting methods

Both of the above error-correcting mechanism slow down the run-time of the search significantly. The problem is that, even though infinite cycles are detected, the algorithm can still make a large number of moves before it returns to a previous state where a cycle can be stopped.

Missing figure

add scalability figure with error-correcting methods

The performance hit shown above limits the size of the lookahead that can be used and the size of the lookahead can severely limit the performance boost obtained. With a lookahead of 1, the corrections done during the running of the search seem prone to noise and may actually hurt the result. The post-processing step however provides a small but consistent boost in performance (see Figure 4.5)

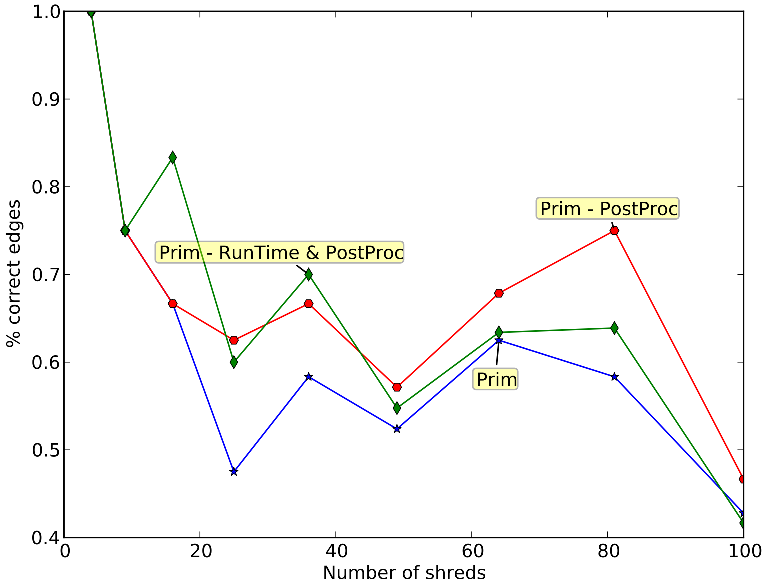


Figure 4.5: Comparison between the basic Prim algorithm and the enhanced versions either with just the post-processing step or with both the run-time corrections and the post-processing. The run-time corrections are not very consistent when using a small lookahead

Bibliography

[1] Bottou L. Howard P. Haffner, P. and LeCun Y. Djvu: Analyzing and compressing scanned documents for internet distribution. In *Document Analysis and Recognition*. Springer, 1999. 10

[2] Arps R. Chamzas C. Dellert D. Duttweiler D. Endoh T. Equitz E. Ono F. Pasco R. Sebestyen I. Starkey C. Urban S. Yamazaki Y. Hampel, H. and T. Yoshida. Technical features of the jbig standard for progressive bi-level image compression. In *Signal Processing: Image Communication*. Elsevier, 1992. 10

[3] J. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. In *Proceedings of the American Mathematical Society*. American Mathematical Society, 1956. 22

[4] Diem M. Kleber F. Perl, J. and R. Sablatnig. Strip shredded document reconstruction using optical character recognition imaging for crime detection and prevention. In *Imaging for Crime Detection and Prevention*. Springer, 2011. 9

[5] M. Prandtstetter. Two approaches for computing lower bounds on the reconstruction of strip shredded text documents. Technical Report TR18610901, Technishe Universitat Wien, Institut fr Computergraphik und Algorithmen, 2009. 19

[6] M. Prandtstetter and G. Raidl. Combining forces to reconstruct strip shredded text documents. In *Hybrid metaheuristics*. Springer, 2008. 9, 14, 19

[7] M. Prandtstetter and G. Raidl. Meta-heuristics for reconstructing cross cut shredded text documents. In *Genetic and Evolutionary Computation*. ACM Press, 2009. 19, 20

[8] Prandtstetter M. Schauer, C. and G. Raidl. A memetic algorithm for reconstructing cross-cut shredded text documents. In *Hybrid Metaheuristics*. Springer, 2010. 19

[9] Massad Y. Sleit, A. and Musaddaq M. An alternative clustering approach for reconstructing cross cut shredded text documents. In *Telecommunication Systems*. Springer, 2011. 9, 14, 19, 20