Grammatical inference of dashed lines

Arnold Jonk & Rein van den Boomgaard & Arnold Smeulders

University of Amsterdam

Faculty of Mathematics, Computer Science, Physics, and Astronomy

Kruislaan 403

1098 SJ Amsterdam

The Netherlands

`jonk@wins.uva.nl`

`tel.: ++31 20 525 7580`

`fax: ++31 20 525 7490`

running head: GIODL

May 25, 1998

Abstract

Dashed lines are a common element in line drawings. This paper deals with the inference of a dashed lines' grammar, given a stream of graphical symbols. A method is presented, based on syntactical pattern recognition, capable of inferring arbitrary grammars without a priori knowledge. A detailed complexity-analysis of the developed algorithms is presented, as well as experiments demonstrating the usefullness of our method.

List of symbols

- α
- i,j,k,x,y,h,t
- $G[]$,$p()$,$K[]$,$L[]$,$f()$
- w_i^j
- $\mathcal{C}$, Ω

1 Introduction

In the automated conversion of line drawings ([12], [18]), such as cartographic maps, engineering drawings and dress patterns, dashed lines are a common element. An automated system for interpreting line drawings needs to detect both the direction and location of the curve as well as its composition of graphical symbols.

A dashed line is defined by the center line and a grammar. The grammar defines the repetition of the pattern along the line. The grammar ranges from quite simple to more complex patterns. The center line of the grammar is the line following the center points of the objects (for example line-segments or crosses).

This paper deals with inferring the pattern of a dashed line given a stream of graphical symbols. Thus in this paper we tackle the problem of finding the underlying pattern of a string of symbols. This problem also appears in as diverse applications like beat-induction in music [3] and the recognition of birdsongs [17].

Detecting the location of the dashed line is treated as a separate problem. There are several ways to tackle this problem, for example by tracking as demonstrated in the experiments. Whatever approach is chosen to detect the centerline of a dashed line, the problem of inferring the grammar needs to be solved.

In literature, a dashed line is sometimes viewed as a discontinuous curved or straight line with, not necessarily regularly placed, gaps. The pattern of the dashed line is not explicitly reconstructed. An example of this is described in [1]. With the use of tube-directional morphological operations, line-segments are connected. The standard line-detecting procedures find the lines in the image. A drawback of this procedure is that the grammar of the dashed line is lost in the process. In addition, different collinear lines are merged. The authors point at the sensitivity of the procedure to noisy data. In [4] a system merging short line-segments into longer lines is described. The grammar of this line is derived on the basis of heuristics, but plays no explicit role in the dashed line detection. The algorithm performs well on the narrow class of admitted grammars. In [19] and [16] general systems for interpreting line-drawings are presented of which dashed lines are part. Both systems rely heavily on specific drawing criteria for selected line-types. In conclusion, these methods fail to make an explicit reconstruction of the dashed line. Therefore, these methods can not be generalized to detect more complex dashed lines.

Methods that reconstruct the pattern of a dashed line include [15]. The algorithm is capable of segmenting a string of symbols into a set of dashed lines. The match between a set of symbols and a grammar is based on stretch, substitution and ommision operations. The set of operations is not robust against frequent object fragmentation. The system is capable of checking the observed linear texture against a list of given grammars, and no other one. In [12] and [9] similar approaches are described, although the class of grammars is restricted even further. The above mentioned algorithms have in common that the class of admitted grammars is small and predefined. In contrast it is our goal to develop an algorithm capable of detecting arbitrary grammars.

In our approach, the detection of the grammar is based on methods from the field of syntactic pattern recognition. In syntactic pattern recognition, a pattern in a class is described by a string of symbols, which is generated by a grammar. Parsing algorithms are used as recognition procedures. The grammar which accepts the string identifies the pattern. The parse of the string provides structural information. Parsing a string becomes difficult when errors are introduced in the string. In general there are several ways of dealing with errors, for example the use of stochastic grammars [6], and the use of error-correcting parsing [7]. Grammatical inference, and especially error-correcting parsing, is a natural approach to tackle the problem of recovering the grammar of a dashed line.

The paper is organized as follows. In section 2 the definition of a dashed line is given. In section 3 the detection and classification of graphical symbols is described, a neccesary step before grammatical inference. In section 5 methods to generate possible grammars, given a string of literals, is presented. Section 4 describes the algorithm that decides which of the tested hypotheses is correct. Finally, in section 6 systematic experiments are described, and some conclusions are drawn.

2 The definition of a dashed line

In defining a dashed line, the following definitions are used:

- Literal. A symbol or a gap.
- String. A sequence of literals.

- Symbol. An element of the alphabet of detectable symbols. A symbol can be, for example, a line-segment, a cross or a dot. Each symbol is parametrized. For example line-segments have length, position, width and orientation as their parameters. The orientation of a symbol is measured in relation to the center line, as is demonstrated in figure 3.

- Gap. Distance between consecutive symbols, as measured along the center line (see figure 3). Gaps are denoted with S_j. The parameters of a gap are its location.

- Cyclic group. A sequence of literals.

A dashed line is characterized by a cyclic group and a center line. Several concatenations of the cyclic group form the stream of literals of the dashed line. The center line then localizes the stream of literals in the image.

3 Object detection

In this section the object detection step is described, which is a necessary step preceeding the dashed line detection. In principal, the range of objects is unbounded. But, to be able to detect objects, an alphabet of objects must be determined. This alphabet is, of course, application dependent. Extending the alphabet will increase the generality of the detection-procedure.

The desired output of the object detection step is a description of the image in terms of object-occurrences. Each occurrence of an object needs to be assigned a certainty. A set of pixels may occur in different object occurrences. This is a consequence of both the uncertainties in detection and a consequence of the fact that detection occurs before interpretation. Consider for example figure 4. The circled object is identical in both images, but its interpretation depends on the context. Both interpretations must be allowed by the object detection step.

4 Matching a grammar against a string

In this section, the method of finding the cyclic group best describing a sequence of literals (from now on called the string) is described. This method consists of two steps. First, a list of hypotheses for the cyclic group is generated. Then each hypothesis is evaluated. The hypothesis resulting in the best fit (referred to as the shortest distance) is selected as the most likely cyclic group. In the next section, the hypothesis-generation is discussed. In this section, the calculation of the distance between an hypothesis and the string is discussed. In figure 5 an overview of the grammatical inference procedure is presented. There are three different methods of generating hypotheses, to be detailed later.

4.1 Cyclic graph matching

In recovering the cyclic group of a string, it is assumed that because of drawing and detection errors, there is no perfect match between the grammar and the string. For example, the string *abcabcabd* might be generated by the cyclic group *abc*, with the last element erroneously detected. In general, there are three types of errors. An insert (extra literal in the string), a delete (missing literal) and a substitute (replaced (group of) literal(s)). Note that any substitution can be written as a sequence of inserts and deletes.

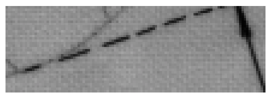


Figure 1: *Fragment of a simple dashed line in an engineering drawing.*

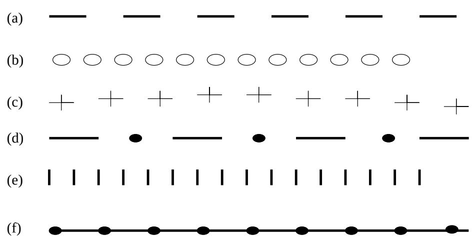


Figure 2: *Dashed lines consist of a wide range of objects.*

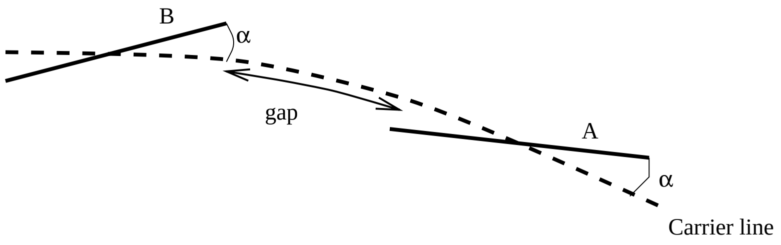


Figure 3: *Objects on a center line, with parameters.*

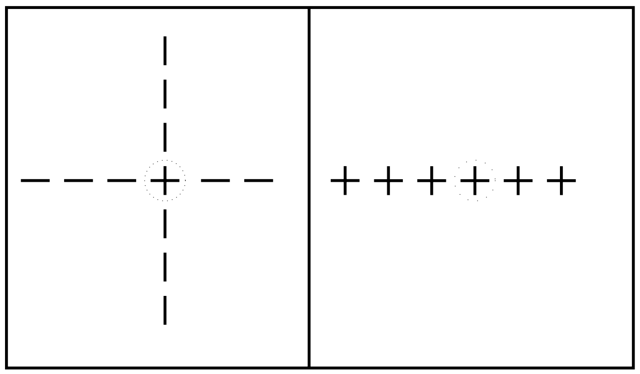


Figure 4: *Example of context dependant interpretation of graphical symbols.*

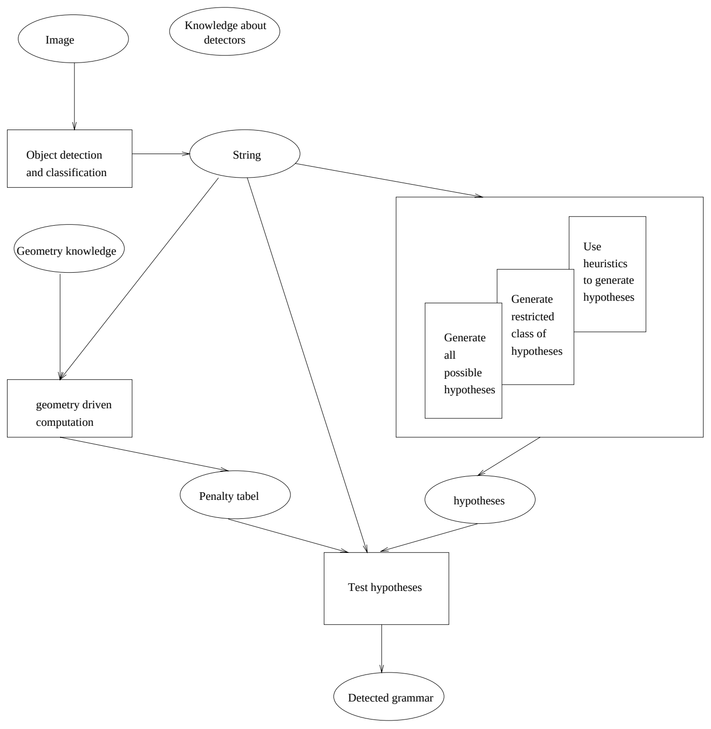


Image
Knowledge about detectors
Object detection and classification
String
Geometry knowledge
Use heuristics to generate hypotheses
Generate restricted class of hypotheses
Generate all possible hypotheses
geometry driven computation
Penalty tabel
hypotheses
Test hypotheses
Detected grammar

Figure 5: *Overview of the grammatical inference procedure. There are three alternatives for the hypothesis-generation, of which one needs to be selected.*

We define the distance between a cyclic group and a string as the minimal cost of a sequence of inserts, substitutes and deletes necessary to map the string on concatenations of the cyclic group. See figure 6 for an example of calculating the distance. This distance is an extension of the Levenshtein distance ([14], which is a distance measure between two finite strings, whereas the string generated by concatenating cyclic groups is infinite. Computation of the Levenshtein-distance was shown to be solvable with dynamic programming in [17]. The problem of matching a cyclic string with a string is explored [8] and enhanced in [9]. We will present a variation on these methods suitable for extension to multi-literal substitutions. Our main contribution stems from rewriting the string matching problem to finding the shortest path in a directed graph. This provides for more flexibility without increasing the computational load.

The graph shown in figure 7 is constructed from n by m nodes, with n the number of literals, and m the size of the cyclic group. Each edge in the graph represents an operation of the type insert (p_i), delete (p_d) or match/substitute ($p_{m/s}$). Each node is denoted as $k[x, y]$ where x and y are the positions in the string and cyclic group respectively. In figure 7, because of the basic insert, delete and match/substitute operations, only adjacent nodes are connected. In contrast to the matching methods referred to earlier, we also allow multiple object substitution. For example, the literals abc might be replaced with a single d.

We define a penalty table P, a matrix with sets of symbols on both axes. The value associated with two sets of symbols is the cost of substituting one set

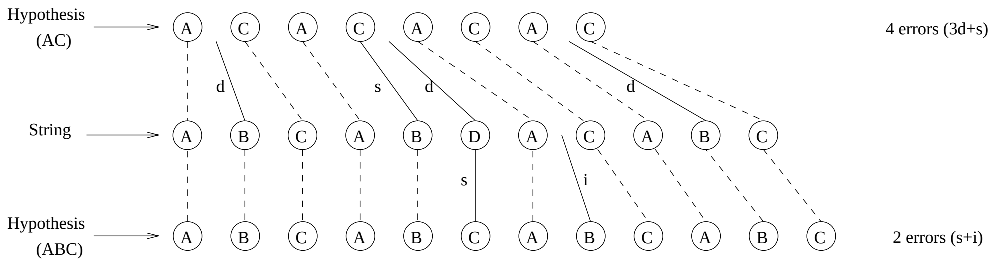


Figure 6: *Mapping two cyclic groups on a string, with examples of all three error-types. The cyclic group (AC) results in four errors, three deletes and a substitute. The cyclic group (ABC) results in only two errors. It is decided as the most likely cylcic group to have generated the string.*

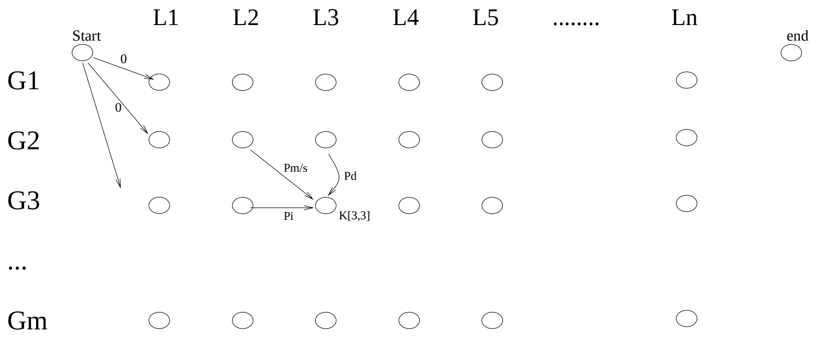


Figure 7: *The matching graph with the three incoming edges of one node, corresponding to the simple m/s, i, and d operations.*

with the other. The value $p(x,i,y,j)$, which is the weight of the edge connecting the nodes $k[x-i,y-j]$ and $k[x,y]$, is the cost of matching $\{L[x-i],\cdots,\mathrm{L}[x-1]\}$ with $\{G[y-j],\cdots,G[y-1]\}$. As defined, these values $p(x,i,y,j)$ are tabulated in P. In figure 8 is it demonstrated that this definition allows for the three basic operations as well as multiple object substitution.

To find the cheapest way to arrive at a node, we define:

$$k[x,y] = \begin{cases} 0 & : \quad x=0 \\ \min_{0\leq i\leq x, 0<j\leq y} k[x-i, y-j \bmod m] + p(x,i,y,j) & : x>0 \end{cases} . (1)$$

Finding the minimal number of errors in the match now reverts to finding the shortest path in the graph. See figure 9 for an example. In this example a simple penalty-table is used. So, $p(x,1,y,0) = p(x,1,y,1) = p(x,0,y,1) = 1$ for any combination of x and y. All other edges have weight ∞.

4.2 Cyclic group matching applied to dashed line detection

Using literal-specific knowledge, we can now generate substitution rules. In this section, an example of this generation is given for straight line-segments.

The entries in the penalty table P must be higher when the substituted literals differ more. Consider for example line-segments. The criteria to compare line-segments include width, angle and length. So, the more two line segments differ in these parameters, the costlier the subsitution.

Multiple literal substitution is depicted in figure 10.(b). Here two line-segments and a gap are replaced by one long line-segment. The criteria for single segment substitution also apply here.

Observe figure 11, in which substitution plays an important role. Here an

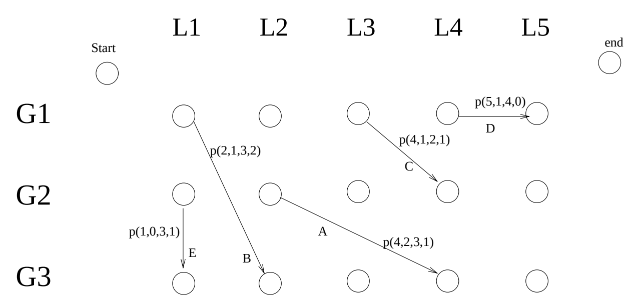


Figure 8: *Edge A substitutes string elements L_2 and L_3 into cyclic-group element G_2. Edge B substitutes string element L_1 into cyclic-group elements G_1 and G_2. Edge C is a match/substitute, edge D a delete and E an insert.*

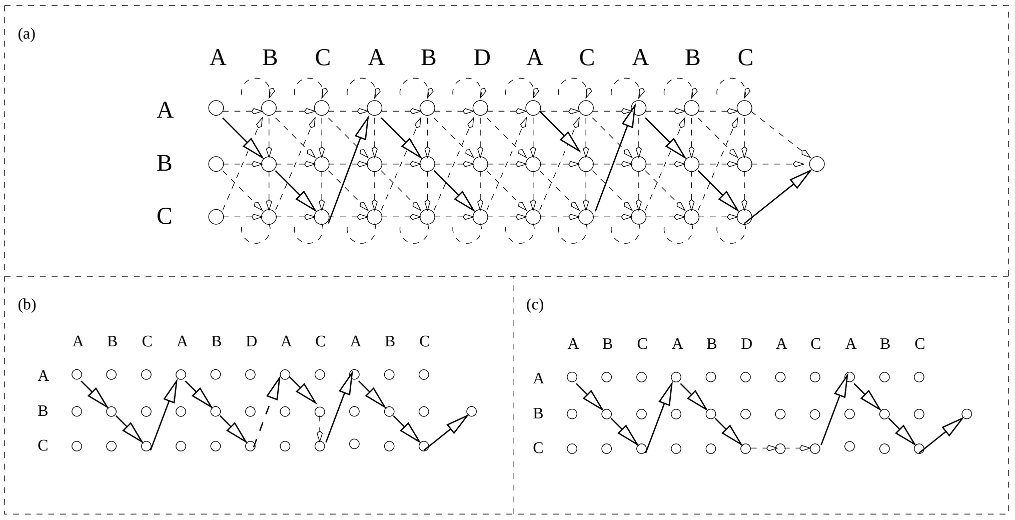


Figure 9: *In graph (a), the dotted line indicates a penalty 1, the solid lines penalty 0. The shortest path in the graph between the start and term node has penalty 2, which is the distance between the string abcabdacabc and the cyclic group abc. Note that there are two walks between the start and end-node with penalty 2. The first walk (b) substitutes the D for a C and inserts a B. The second walk (c) deletes DA.*

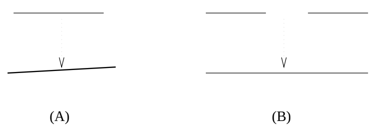


Figure 10: *Substitution examples. (a) Valuation of substitution must be based on width, angle and length. (b) The same holds for multi-literal substitution.*

example image is presented with the detected line-segments. In figure 11.c the resulting gaps and line-segments with their assigned class are shown. In figure 11.d, the interpretation that follows from the match with the cyclic group $xayaya$ is presented. Multiple literal substitution is used twice to interpret a long line-segment and short gap as a middle-sized line-segment and gap. It is clear the cyclic group $xayaya$ results in a very good match.

The function used to generate substitution rules is derived from the clustering function described in [10]. Several substitution rules were generated, shown in figure 12. Here gaps are denoted by dashed boxes. In example 11, only rule (b) was used. Other rules might have been used if the match was performed on a different cyclic group. Applying these rules in a match is very cheap, because these rules retain the length of the dashed line. In other words, the rules explain missing or superfluous ink, they do not stretch or shrink the observed dashed line.

4.3 Complexity

The complexity, measured in the number of considered edit-operations, of evaluating a single hypothesis is $n \times m \times s$. Here, n is the number of literals in the string, m the size of the cyclic group, and s the average number of substitution rules and edit operations that can be applied at a node in the matching graph. Observe that s is proportional to the size of the alphabet, and typically very small, but at least 3.

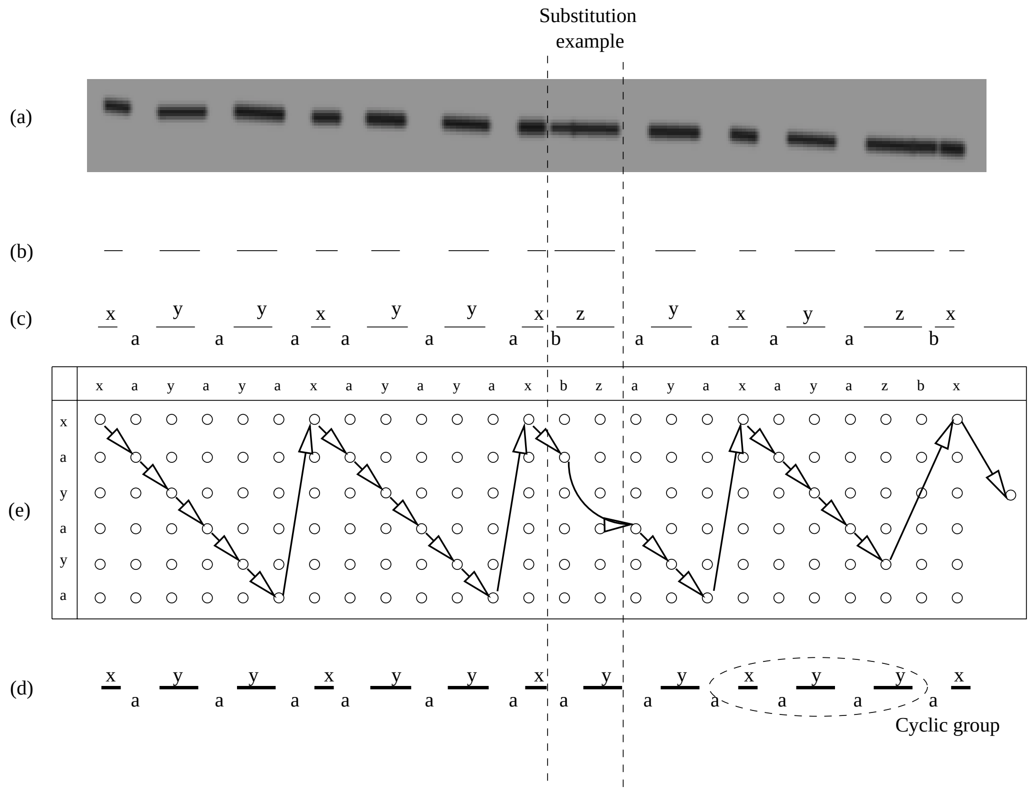


Figure 11: *Example of matching. (a) Input image. (b) detected line-segments. (c) classified gaps (a,b) and line-segments (x,y,z). (d) The match with a, regularly spaced, cyclic group xayaya.*

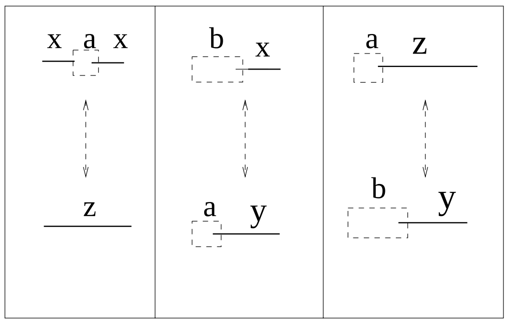


Figure 12: *Example of three generated rules, with low cost. The low cost is due to comparable sizes of the (sequence of) literals on the top and bttom part of the substitution rules.*

5 Finding the optimal cyclic group

Given the alphabet and the number of literals in the list, the number of possible cyclic groups is bounded. It is limited to all possible permutations of the alphabet within the possible grammar sizes. To consider this complete list of all permutations is, however, very costly. If the number of different symbols and gaps is denoted as w_s and w_g respectively, then $w_s^{m/2} w_g^{m/2}$ is the number of possible cyclic groups of size m. As the maximum length of the cyclic group is half the length of the string, we arrive at $O(\sum_{m=1}^{n/2}(w_s^{m/2} w_g^{m/2} nm))$ for the complexity of the algorithm evaluating all possible cyclic groups. We can evaluate this equation as $O(n^2 w_l^{n/2} w_g^{n/2})$. Figure 13 gives a plot.

One way to reduce the amount of tested hypotheses is to impose constraints on the allowed grammars. For example, dashed lines in standard engineering drawings do not consist of cyclic groups with length greater than five. Grammars used in engineering drawings and cartographic maps are of a distinct type of grammar. With A and B line-segments C a gap, these grammars can be described by $(AC(BC)^m)$, with m usually between 0 and 4.

These assumptions on the size and semantics of the cyclic group lead to a different computation of the complexity. The amount of possible hypotheses now equals $w_l w_g + 4w_l^2 w_g$. Combining this with the $O(nm)$ complexity of the graph matching, we get a complexity of $O((w_l w_g + 4w_l^2 w_g)\sum_{i=0}^{4}(ni)) = O(n)$.

5.1 Heuristics for general solutions

While in most practical applications assumptions on the grammar reduce the computational load, this might not always be possible. In this section we investigate methods that use heuristics to arrive at an algorithm with an acceptable complexity.

We observe that literals often occur after one another in the string. As a consequence, they should also be neighbors in the cyclic group. In figure 14, a matrix is presented. In the matrix the number of times one literal is followed by another is tabulated. The matrix is visualized by a weighted directed graph, as shown in figure 14.(b).

A cyclic group can be, as was shown for a string, written as a directed graph. The derived graph is Hamiltonian[1] if we add an edge between the last and the first element in the cyclic group. Starting from the string's neighbor-graph G_s, we determine classes of feasible hypotheses in an attempt to reduce the number of hypotheses.

We define the set of cyclic group-graphs, denoted by $\mathcal{C}$. Each element $C \in \mathcal{C}$ is a graph with the same vertices as G_s, and its edges form a closed walk in G_s. C is a weighted directed multigraph, with the sum of weights of the edges between two vertices $v1$ and $v2$ equal to the weight of the same vertices in G_s. The edges of the graph C are denoted as V_c. $\mathcal{C}^n$ is defined as the set of all the closed walks in G_s with length $\leq n$. For an illustration see figure 15.

Each graph $C \in \mathcal{C}^n$ can be written as a cyclic group by noting the order in which the vertices are travelled in a Hamilton-walk. The derived cyclic group is not unique. The example of figure 16 shows the smallest [2] graph yielding two

[1] A Hamilton-walk is a closed walk in the graph that travels all the edges exactly once and has equal start point and end point. A graph is Hamiltonian if such a walk exists.

[2] In order to obtain two rotational variant Hamilton walks, at least six edges are needed.

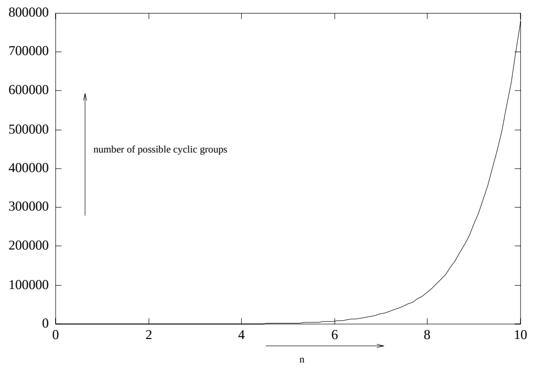
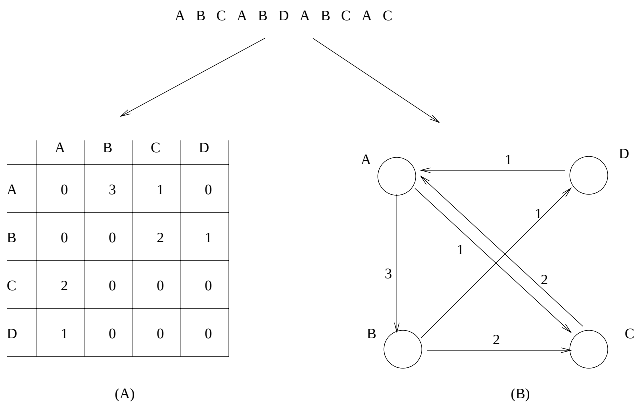


Figure 13: *The number of possible cyclic groups with varying stringsize.* $w_g = w_l = 3$.



	A	B	C	D
A	0	3	1	0
B	0	0	2	1
C	2	0	0	0
D	1	0	0	0

Figure 14: *(a). The neighbors matrix of an example string. (b) The directed graph.*

different not rotationally symmetric cyclic groups. A graph with six edges can lead to two different cyclic groups, but the number of different Hamilton walks grows rapidly in relation to the amount of edges.

For each graph $C \in \mathcal{C}^n$, $f(C, G_s)$ measures the extend to what C explains G_s. The weight of an edge is denoted as $|v|$, $\Phi(G_s)$ is the sum of the weights in G_c and $\Omega(C)$ is the number of (non-zero weight) edges in C. $\Omega(C)$ thus calculates the length of the cyclic group. We now define the graph penalty function:

$$f(C, G_s) = \sum_{v \in V_c} (|v| - (\Phi(G_s)/\Omega(C)))^2. \tag{2}$$

$f(C, G_s)$ calculates the sum of squared differences between the edge weights in C and the relative length of the string.

In figure 15, the right hand graph of length 3 leads to a cyclic group of (ABD). If the original string ($ABCABDABCAC$), was produced by this cyclic group, a value of $10/3 = 3.3$, using the penalty function, would be expected at each edge. The values in the graph are much lower than 3.3, so the cyclic group ABD does not seem to be likely. In figure 17 the graphs from figure 15 are presented, along with their graph penalty. Note that in both measures, the same ordering is only preserved for all graphs of the same length. This is due to the fact that the heuristic favors graphs with many edges (large cyclic groups) because the square term awards a low expected weight. In general the heuristic does not answer the question which groupsize was most likely used to generate the string.

Given the measure $f(C, G_s)$ we can construct an algorithm that, given a

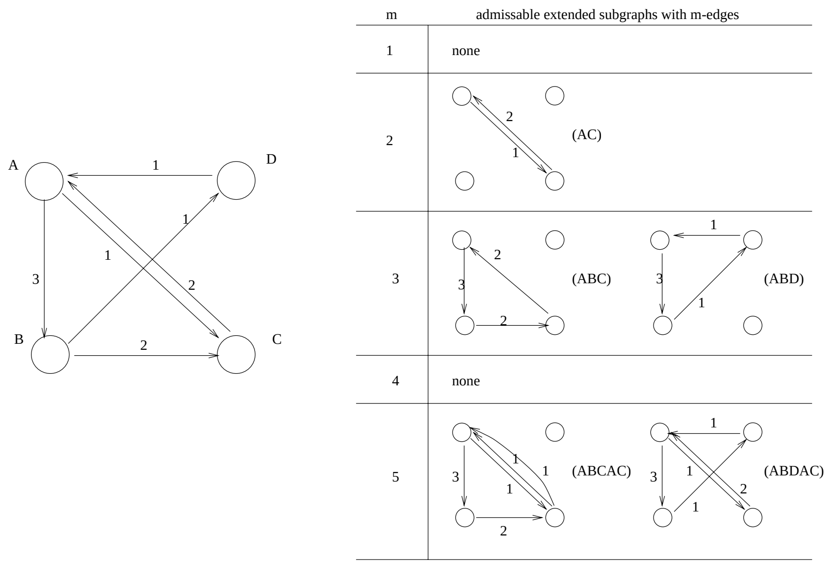


Figure 15: *Starting from the directed graph on the left, the graphs on the right can be constructed. The resulting graphs are ordered by size. Each graph is accompanied by an associated cyclic group.*

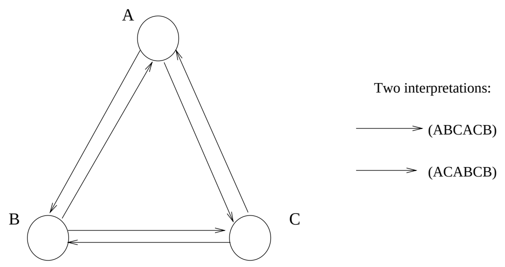


Figure 16: *Graph with different Hamilton walks.*

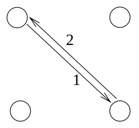
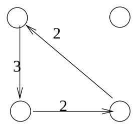
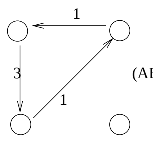
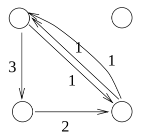
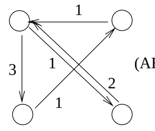
			Squared difference	Graph matching penalty
2		(AC)	25.0	4
3		(ABC)	3.65	2
		(ABD)	11.0	3
5		(ABCAC)	4.0	2
		(ABDAC)	4.0	1

Figure 17: *The two penalty measures applied to an example. In the Graph matching, an insert, delete and substitute operation have penalty 1.*

string, quickly computes the subgraphs with lowest penalty. This algorithm generates all Hamiltonian subgraphs of G_s and computes the graph penalty function for each of the generated graphs. The number of admissable subgraphs is large. It is proportional to the number of subgraphs which is 2^n ([3]). Because the algorithm only needs to generate the cheapest (in terms of graph penalty function) Hamiltonian graphs of a fixed length, this algorithm need not have a high average case complexity.

In practice, the Hamiltonian graphs are generated in a recursive process. The algorithm is described in pseudo-code in figure 5.1. The procedure $SelectNextEdge$ selects the edge with highest multiplicity in the original graph G not yet part of subgraph SG. In the procedure $OrderMultiplicity$ the multiplicity of the selected edges are ordered according to the minimal contribution to the subgraph's penalty. Only subgraphs which can result in a Hamiltonian subgraph of size $size$ and a lower penalty than found previously are being considered. The algorithm considers all possible subgraphs in principle. But it does so in a manner that limits the amount of completely considered subgraphs considerably.

An analysis of this algorithm in more detail is beyond the scope of this paper. Figure 19 gives an indication of the performance of the algorithm.

[3]We arrive at 2^n because there are n edges, and a subgraph is formed by swithing edges on and off.

```
procedure  FindHamiltonian(Graph G, Graph SG, Integer size, Integer current)

if  current = size
  Best ← SG
  return
endif

Edge ← SelectNextEdge(G,SG)
MultList ← OrderMultiplicity(G,Edge,current)

for  Multiplicity ∈ MultList
  NSG ← AddEdge (SG,Edge,Multiplicity)
  if  Hamiltonian(NSG,size) = TRUE
    Eval ← Evaluate(NSG,size,current+Multiplicity)
    if  Eval < BestEval
      FindHamiltonian(G,NSG,size,current+Multiplicity)
    endif
  endif
endfor

return
```

Figure 18: Pseudo-code of the optimal subgraph finder

5.2 Substrings

A reasonable assumption is that the cyclic group occurs at least once in the string. Using this assumption, the number of cyclic groups to check reduces even further. In figure 20 the probability is shown that a cyclic group is a substring of the string it has generated. In appendix A it is shown how to calculate this graph.

Even though the cyclic group may not be a substring of the generated string, using substrings may still be optimal. This is due to the fact that the cyclic group not necessarily has the lowest penalty of all possible hypotheses. In figure 21 this effect is demonstrated. In this figure, the probability that the optimal hypothesis is excluded by the employed heuristics is presented. This figure is derived by repeated testing (every combination of m and n was repeated 50 times). Therefore, it appears not as smooth as figure 20. As can be seen in figure 21, the algorithm based on heuristics occasionally makes an error. But this does not occur too often, and mainly with high $cyclicgroupsize/stringsize$ ratios. Given the fact that generating all possible hypotheses is not feasible in practical applications, it is concluded that the heuristics perform well.

5.3 Complexity analysis

Please recall that all substrings of the string are taken as potential cyclic groups. There are $n-m$ possible substrings of length m in a string of length n. Because evaluating one hypothesis has a complexity of $O(nms)$, where s si again the average number of applicable substitution rules, evaluating all substrings of a string as hypotheses has a complexity of $O(\sum_{m=1}^{n/2}(n-m)nms)$. We evaluate

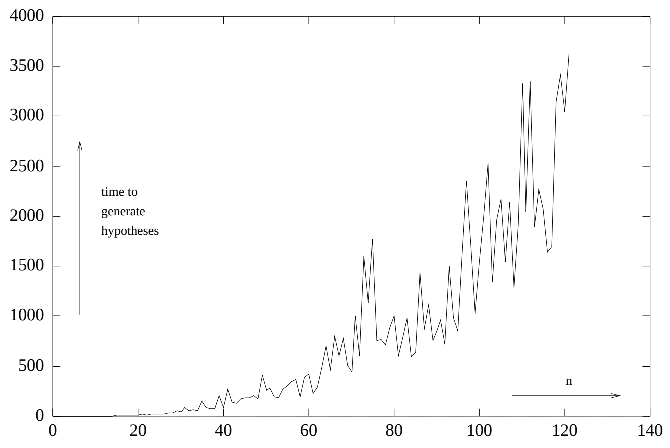


Figure 19: *Time (in ms) to compute optimal set of hypotheses according to the subgraph heuristic. Durations are averaged over series of 50 random generated strings based on cyclic groups with random length. The chance of an error in generating a string from a cyclic group was set to* 0.2 *per symbol.*

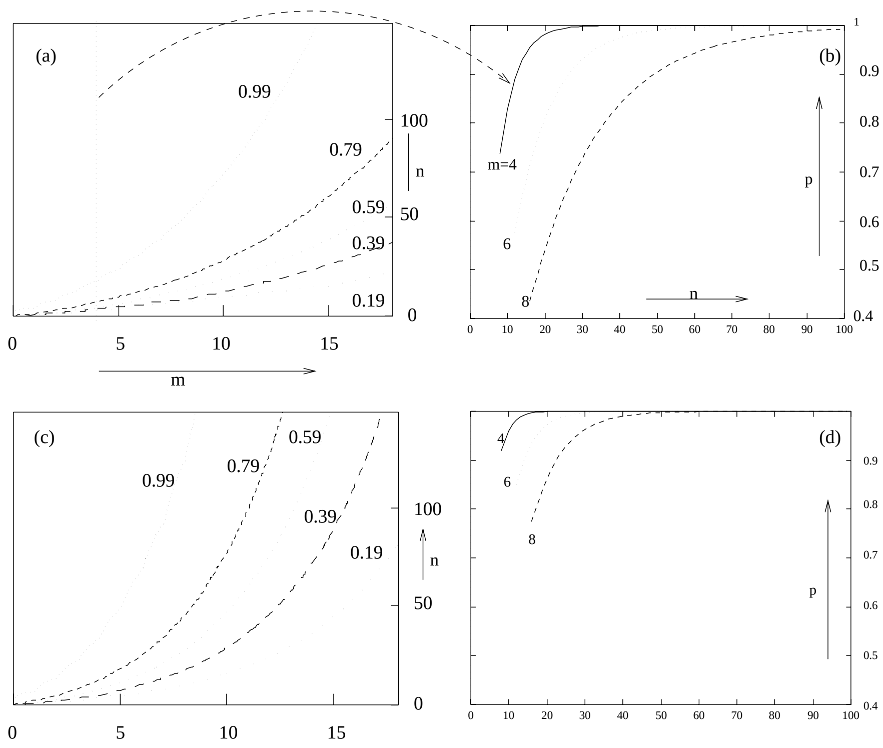


Figure 20: *(a) Iso-probability lines that the cyclic group is a substring of the generated string. (Probability of any error set to* 0.2 *in figure (a), and* 0.1 *in figure (c)).*

this, using summation rules, as

$$O(s(5n^4/24 - n^3/24 - n^2/12)), \tag{3}$$

which is equivalent to $O(n^4)$.

First it is noted that the order of n^4 is an upper limit, and only valid in the pathological case. For example, it does not take into account the many redundancies in substrings. How this workes out is demonstrated in figure 22.

The order-analysis assumes that every hypothesis is tested in full. This is not necessary in general, for the algorithm is only interested in the best fit. Only a rather naive algorithm would first evaluate all the hypotheses completely, and then select the best fit. A smarter algorithm, incrementing the evaluation of every hypothesis, leads to an algorithm with an acceptable average-case complexity. This does not change the worst-case however, for it will always be possible to construct an example where every hypothesis will need to be evaluated in full.

When the method of using substrings is combined with the neighbor-graph approach (only hypotheses that pass both criteria are tested), the number of hypotheses is reduced again. In practice we adopt this approach. The results of this approach are outlined in section 6.

5.4 Imposing a maximum on the size of the cyclic group

In practice, there is an upper limit for the length of the cyclic groups used in the drawing. When the analysis is limited to cyclic groups of a fixed length of

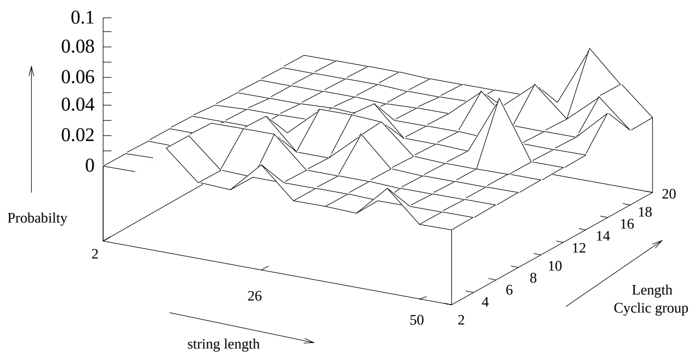


Figure 21: *Probability that the heuristics incorrectly exclude optimal cyclic group. (Chance of error set to* 0.2*)*

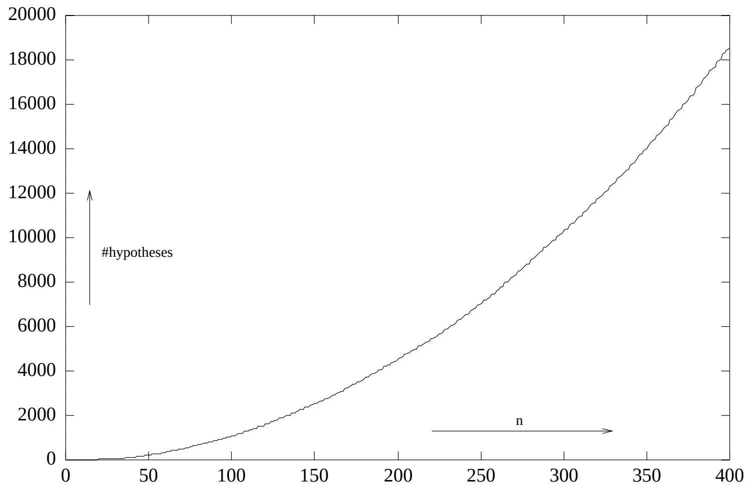


Figure 22: *Number of hypotheses in relation to the size of the string. The graph was obtained by repeatedly generating strings based on a cyclic group of length 6 and determining the number of different substrings. The probability of an error was set to* 0.2.

$[1,\ldots,m_{max}]$, equation 3 changes to

$$t(n) = \sum_{m=1}^{m_{max}} (n-m)nms, \qquad (4)$$

which can be rewritten to $t(n) = (\frac{1}{2}nm_{max}^2 + \frac{1}{3}m_{max}^3)ns$. This has a worst-case order of n^2, a considerably improvement over the complexity in the case where no constraints were placed on the grammar. In practise the amount of hypotheses to be tested, captured in the first term, is sublinear due to the redundancy-factor described earlier. Observe figure 23 for an illustration of this sublinearity. So, in effect a nearly linear result is achieved.

6 Experiments

In figure 24.(a)-(e) a dashed line is shown with increasing degradation. The ground truth in these images is the same, but in each successive image more gaps and short line-segments were added to the image. The images are synthetic, and generated by an algorithm described in [2].

The dashed lines in these experiment consist of straight line segments and gaps. The line segments are classified by a simple procedure segmenting the length histogram. Significant peaks are detected by iteratively convolving the histogram with a Gaussian 1D-kernel until the number of peaks does not change in succesive steps. Each line segment is then classified to the nearest peak in the length histogram. The initial size and increment of the Gaussian kernel is based on experience. Experience shows that a small initial size and increment gives good results. It is stressed that in this paper we concentrate on the grammatical inference. The object detection and classification procedures can be freely chosen.

A simple method for detecting the dashed line and classifying the objects is employed before the grammatical inference is invoked. The dashed line detector is invoked by given an initial estimation of the dashed line (represented by an object and a direction). The dashed line is incrementally extended by added the object in the image that is aligned to the dashed line. If there are more than one object available, the object leading to the grammar with lowest penalty is selected.

Figure 24.(a) displays an undisturbed images. Among the dashed lines in figure 24, only the last one did not result in a correct classification of the grammar.

In figure 25, the relationship between the amount of degradation and recognition-succes is depicted. The undisturbed image consists of 29 line segments in a regular pattern. The image is disturbed by adding random gaps and lines. The figure shows that the recognition breaks down when around 6 lines and gaps are added. In effect it means that recognition stops when less than 60 percent of the correct line segments remain. In those cases, human perception also breaks down. See for example figure 26 for an image that was generated with 6 additional gaps en line segments. It is clear that this type of image causes problems for the human eye.

In figure 27 the time to infer the grammar in relation to the size of the dashed line is presented. The tested image is 24.(c).

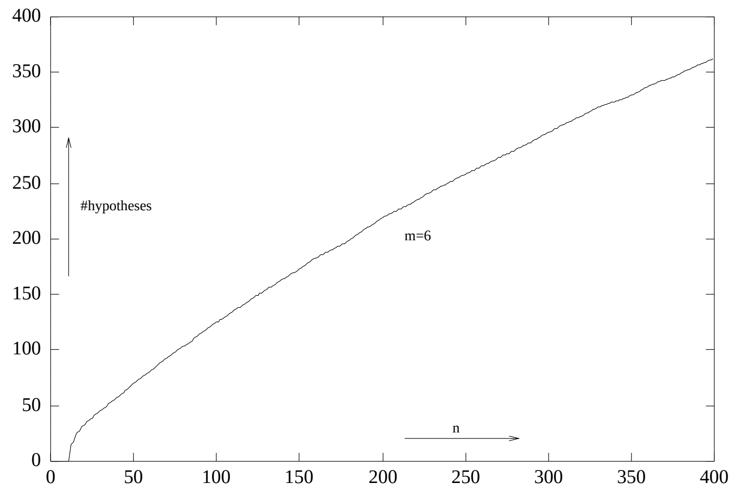


Figure 23: *The number of hypotheses to be tested when the maximum size of the cyclic group is set to 10. This graph was generated by repeatedly generating strings based on a cyclic group of length 6, and the chance of an error set to* $0,2$.

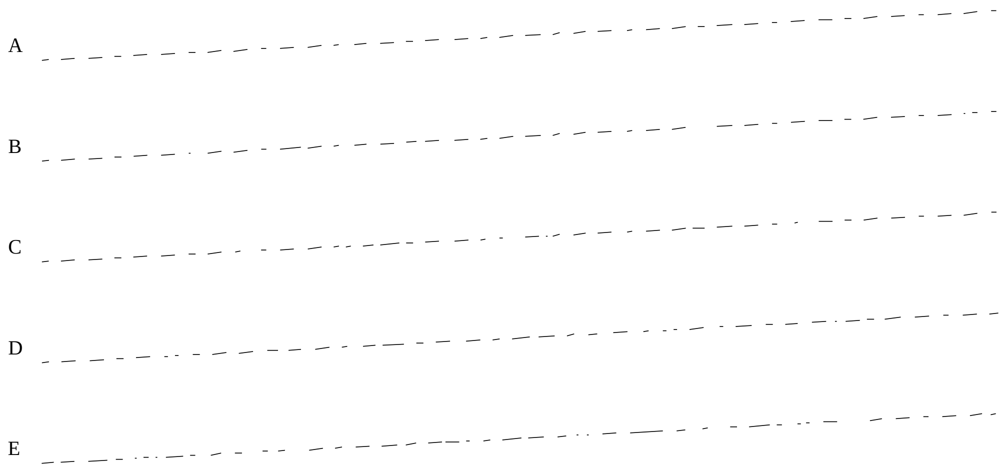


Figure 24: *Dashed line with increasing degradation. (a)-(d) were correctly classified.*

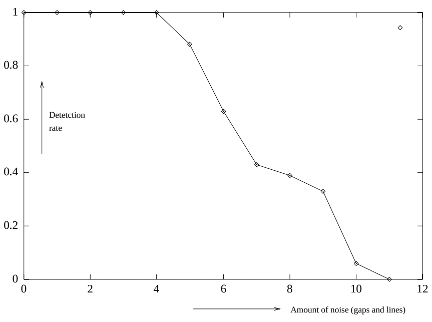


Figure 25: *A dashed line (consisting of 29 line segments) is degraded with a number of gaps and additional lines (the x-axis) before the dashed line detection is employed. In the graph, the amount of disturbance is depicted againt the detection rate. A dashed line is correctly classified if both the grammar, and the parameters of the line-segments and gaps are correctly extracted. The pattern to be detected consists of one short and two long straight line segments separated by small gaps. Thirty images were generated for every degradation level.*

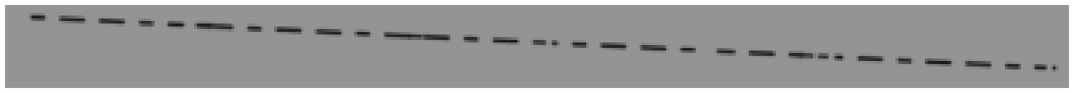


Figure 26: *Example dashed line, consisting of 29 line segments in a regular pattern and dewgraded by added 6 random line segments and gaps.*

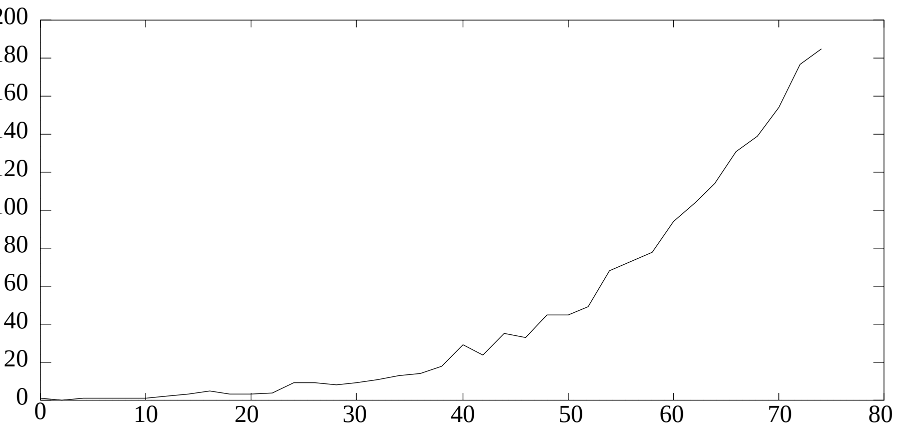


Figure 27: *Performance of the grammatical inference. On the horizontal axis, the number of literals is displayed. The vertical axis shows the time in $\frac{seconds}{100}$. Our algorithms are implemented on a Sun sparc station, 50 Mhz.*

7 Conclusions

In this paper a method was developed to infer the grammar of a string of symbols. First the graphical symbols present in the image need to be detected and the centerline of the dashed line must be determined. These two steps are necessary prerequisite before the grammar of the dashed line can be inferred.

We showed, by extending known techniques, how the matching distance between a grammar and a string of graphical symbols can be determined using dynamic programming. The complexity if the matching algorithm is $O(nm)$ with n and m the length of the string and the grammar respectively.

Methods were investigated to generate the set of possible grammars that could have generated the string. It was shown that an exhaustive search of all possible grammars was implausible. There are two ways to overcome this problem. First, restrictions can be imposed on the size and type of the grammer. These restrictions can often be derived from domain knowledge about the line-drawings under study, and lead to an algorithm with acceptable performance. When the size of the grammar is restricted, the order of the grammatical inference-algorithm is $O(n^2)$.

The second way to reduce the computational load of the grammatical inference is by introducing heuristics. We observed that symbols often occur after one another in the string. As a consequence, they probably are also be neighbors in the cyclic group. Based on this, a small subset of all possible grammars can be generated. The resulting algorithm has a complexity of $O(n^4)$, which is acceptable for strings of practical length.

In the experiments we have shown that the algorithm is capable of recognizing common dashed lines without a priori knowledge about the type and length of the grammar. The recognition breaks down when the dashed line is degraded beyond human recognition capability.

The work presented in this paper is extendable to other applications where patterns in a linear stream of symbols must be inferred. Examples of such applications are as diverse as the detection of birdsong, musical beat-induction and inference of heart-rhythms. Extendability to two-dimensional applications like texture matching although interesting in itself, is not straight forward.

A Substring probabilities

The probability of a cyclic group occurring as a substring of the generated string depends on the probability of an error, and the length of the cyclic group and generated string. This problem has a natural analogy to a well known coin tossing problem. Namely, the chance of a run of m heads in a sequence of n trials, where $n \geq m$. This problem is explored in [5].

Calculating these probabilities can be done by approximation, as in the reference, but also exact. This is done by constructing a graph, as shown in figure 28 for calculating the probability of a run of 3 heads in a sequence of 8 tries. In this graph, every node has two outgoing edges, one for a tail (t) and one for a head (h). The nodes are denoted by coordinates i, j, where i refers to the number of previous tries, and i the current run of h's. A t-edge exiting from an i, j node leads to the $i+1, 0$ node. This is not true for t-edges exiting from an i, m-node, which insteads leads to the $i+1, m$-node.

The value in a node (i, j), denoted by $C[i, j]$ is computed by summing the products of the weights of incoming edges with the value of the node they exited from. The value of node $0, 0$ is set to 1. Now, $C[i, m]$ represents the probability of a sequence of m $h's$ in i tries.

This graph can be represented by the following (with m the size of the cyclic group, n the length of the string, and $1-p$ the probability of an error) recurrence-relations:

$$C[i,j] = \begin{cases} 1 & : \quad i=0, j=0 \\ \sum_{k=0}^{m-1}(1-p)C[i-1,k] & : \quad i>0, j=0 \\ pC[i-1,j-1] & : \quad i>0, 1\leq j<m \\ C[i-1,j]+pC[i-1,j-1] & : \quad i>0, j=m \end{cases}$$

With $1 \leq k \leq n$, and $C[n,m]$ being the desired probability. There is no known closed-form formula for $C[n,m]$. To simplify, the $C[i,m]$ nodes could also be expressed in each other:

$$C[j,m] = \begin{cases} 0 & : \quad j<m \\ p^m & : \quad j=m \\ C[j-1,m]+(1-p)p^m(1-C[j-m-1,m]) & : \quad j>m \end{cases} \tag{5}$$

Because we are only interested in $C[n,m]$, equation 5 allows for fast computation of the desired probability.

References

[1] Gady Adam & Huizhu Luo & Its'hak Dinstein . Morphological approach for dashed lines detection. 1996

[2] C. De Boer & A. Smeulders. Towards the performance analysis of grey-value dashed lines. GREC'97

[3] P. Desain & H. Honing. Rule-based models of initial beat induction and an analysis of their behaviour. Proceedings of the 1994 International Computer Music Conference. 80-82.

[4] Dov Dori & Liu Wenyin & Mor Peleg. How to Win a Dashed Line Detection Contest. 1995

[5] W. Feller. An Introduction to Probability Theory and Its Applications. 1965

[6] K.S. Fu. On Syntactic Pattern Recognition and Stochastic Languages. Frontiers of Pattern Recognition ed. by S. Watanabe, Academic Press, 1972

[7] K.S. Fu. Error-Correcting Parsing For Syntactic Pattern Recognition. in Digital Patter Recognition, Springer, Berlin, 1976.

[8] J. Gregor & M.G. Thomason. Dynamic programming alignment of sequences representing cyclic patterns. IEEE Trans. PAMI, 15: 129-135, 1993.

[9] J. Gregor & M.G. Thomason. Efficient dynamic programming alignment of cyclic strings by shift elimination. Pattern Recognition 29(7):1179-1185

[10] A. Jonk & A. Smeulders. An axiomatic approach to clustering line segments. ICDAR 95

[11] S.H. Joseph & T.P. Pridmore. Knowledge-directed interpretation of mechanical engineering drawings. IEEE PAMI, vol. 14 (9), 1992

[12] T. Kasvand. Linear texture in Line Drawing. 8th ICPR, 1986

[13] C.P. Lai & R. Kasturi. Detection of dashed lines in Engineering drawings and maps. Proceedings ICDO, Saint-Malo, France, 507-514, 1991

[14] V.I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. Cybernetics and control theory 10(8):707-710, 1966

[15] R. Mariani et al. Linear texture segmentation using elastic model matching. Application for geographic map understanding. GREC 1997, pp. 201-208

[16] R. Kasturi & S.T Bow et al. A system for interpretation of line drawings. IEEE PAMI, vol. 12 (10), 1990

[17] D. Sankoff & J.B. Kruskal. Time warps, string edits, and macromolecules: the theory and practice of sequence comparison. Addison-Wesley, 1983

[18] Systems for paper map interpretation: methods engineering. A.W.M. Smelders & T. ten Kate. IWGR 1995, pp. 110-118

[19] P. Vaxiviere & K. Tombre. Celesstin: CAD conversion of mechanical drawings. IEEE computer, 1992

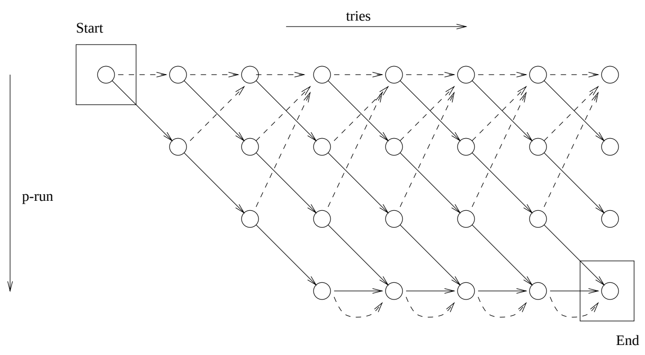


Figure 28: *Graph used to calculate probability of a run of 3 h's in a sequence of 7 tries. A solid line denotes a head (h), with $P(h) = p$. A dashed line denotes a tail (t), with $P(t) = 1 - P(h)$.*