Helping SMEs Automate like Corporations: A Constraint Satisfaction Problem for Automatic Invoice Field Extraction

Guido Teunissen
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
g.teunissen@student.utwente.nl

ABSTRACT
Invoice feature extraction has been a topic of research for many years. Current methods are, however, mainly focused on the visual and positional layout of an invoice, essentially disregarding the information in the field contents. This paper regards this research topic as a constraint satisfaction problem. It therefore first gives a comprehensive view on what field variables are commonly present on Dutch invoices, and which structured set of data variables are practically relevant as output. After variable identification, logical relationships among these variables are composed. This, unlike previously presented methods, allows for the verification of the structured output, because this method is based on the logic among field contents. The accuracy of the tool, tested on a dataset of 241 invoices, is 85.1% for the overall system, and 89.1% only regarding system errors. The method shows its full potential in the false positive rate of only 2.9%. This research sets the basis to develop a practically relevant tool for invoice field extraction and analysis in a real-world Dutch SME setting.

KEYWORDS
Invoice Analysis, Document Processing, Feature Extraction, Constraint Satisfaction Problem.

1. INTRODUCTION
Processing inbound invoices is an administrative strain companies would rather eliminate. However, a study among Belgian companies showed that in 2013 only 6% of the invoices sent among businesses were digital. Others were either still sent via paper (67%) or via email (27%) [14]. This shows that an overwhelming majority of the invoices sent will still need to be handled manually or via a (semi-)automated processing system. Automating the process of handling paper or emailed invoices could allow for cost savings and better accuracy. Provided the majority of the inbound invoices are still in paper or image format, automation can aid in the processing by extracting the pieces of data - or ‘fields’ - on those invoices that were previously typed over manually.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 2017, University of Twente, Faculty of Electrical Engineering, Mathematics and Computer Science.

Dutch small and medium businesses (SMEs), which this paper focuses on, must periodically submit their VAT tax filings of, among others, their received purchase order invoices [3,7]. Administration, via e.g. accounting software, helps in processing these filings, structuring this data efficiently by storing only the relevant and required fields of each invoice. Field extraction is identifying those relevant fields from the invoice document, and giving a structured output.

Automatic analysis of invoices by extracting relevant data parts has been a topic of research for many years. Different methods have been presented to solve this problem. Some previously proposed methods use visual cues, like separator lines and enclosing boxes, for identifying data fields [12]. Others use a morphologic approach with keywords in invoice tables [2]. For non-standardly designed invoices, this will be of little use. Another approach is classifying invoices with the same layout and assuming the data fields are positioned similarly. Classification can be based upon the use of image features, structural features, and/or textual features [1]. This approach, however, still requires a learning set per invoice class, reintroducing manual processing. [9] and [10] improved on this by separating previously known structures from new ones, and processing the new ones automatically. However, this method yields better results with more processed invoices, still giving larger corporations with more invoices an advantage.

This paper views invoice field extraction, unlike previous work, as a constraint satisfaction problem (CSP) [8,13]. A CSP, in its basic form, tries to solve a given problem by filling in every combination of input data to a set of variables, and evaluating every combination to be valid by an identified set of rules. The paper therefore identifies local logical relationships among the fields of Dutch invoices, to develop a logical understanding about the common structure of those invoices. These relationships serve as the base intelligence of the CSP. Previous research mainly disregarded the actual field content information, utilizing only a limited view of an invoice. Because our approach is content-based, it can validate potential answers based on the relations – also called constraints - that are common for invoices.

In addition, the CSP approach will have an advantage in processing creatively designed invoices, and in smaller datasets. Because our method can be applied to all Dutch tax-compliant invoices, there is no classification step needed, nor is a large database of previously processed invoices needed to yield better results. The accuracy of our method is, in contrast, not related to the number of previously processed invoices. This provides a method that averts the preliminary step of classification, the positional and graphical assumptions about
the layout, and the use of a morphological method, like in previously used invoice field extraction methods. Therefore, our method can be applied where others are limited; invoices with unconventional positioning of elements, creative graphical design, or non-standard usage of keywords can still be processed.

2. BACKGROUND

The method of doing field extraction for this research is via the CSP approach. A CSP as defined by [13] is applicable to our case. It can be described as follows. We are given a set of variables, with for each variable a finite domain of possible values that can be assigned to that variable. All possible combinations of variable-value assignments are reviewed. Constraints are a composition of the set of variables. These constraints limit the combination of values the variables can take, such that one or multiple combinations of assignments of values to those variables satisfies all constraints. This combination of variables can then be regarded as a solution to the problem. A constraint satisfaction algorithm (CSA) is then developed to solve the CSP.

An invoice is a type of document that can be represented by its set of fields. The fields of an invoice are elements that contain a type of data. Fields in CSP terminology can be viewed as variables. Figure 1 gives an example of an invoice where all financial fields are marked. In this example, the ‘Total Tax Amount’ field has ‘€1,05’ as its field value.

Invoices can contain a diverse set of fields, but for CSP purposes it is necessary to find those fields - or variables - that are commonly present on invoices. Invoices in The Netherlands are obliged by law to include a certain set of fields. These requirements are stated by the Dutch tax authority, the ‘Belastingdienst’ [6]. Invoices whose set of fields include all the specified required fields are tax-compliant. A dataset of invoices of a Dutch SME is used to verify if the tax-compliance view of variable identification is adequate for real-world usage or if deviations are found.

This paper aims to identify any logical relationships among fields, that constitute an invoice, ultimately trying to incorporate these logical relationships in a uniform system of equations, or in CSP terminology, a set of constraints. Such a relationship could be, for example, a linear relationship between the ‘Subtotal’ field, €’5,00’, the ‘Total Tax’ field of €’1,05’, and the ‘Total Amount’ field of €’6,05’.

Figure 2 gives an example of extracted fields values from the invoice in Figure 1. The scanned invoice has several types of fields that are extracted from the image. Some of those values are noise; they are not financial values, such as dates, names, and other text values. Other fields such as the Total Amount field of €’6,05’, a Total Tax field of €’1,05’, and a Subtotal field of €’5,00’ are supposed to be filled in in the variables, but are at this stage still unstructured. The CSA then tries every possible combination of values by filling them in in the variables and evaluate if an identified constraint, such as ‘Total Amount = Subtotal + Total Tax’, amounts to true. If this is the case, the CSA presents this combination of values to variables as structured output.

![Figure 1. Example of an Invoice with Marked Financial Fields.](image1)

![Figure 2. Example of Field Values Extracted from the Invoice](image2)

3. RESEARCH AIMS

R1. The paper aims to identify a set of variables, both commonly present on Dutch invoices, and practically relevant to Dutch SMEs for administration purposes. These variables will be based upon: (i) Dutch tax authority compliance, (ii) cases commonly found in a real-world setting, (iii) practical relevancy for Dutch SMEs, so it composes a uniform, comprehensive, and practical view of fields in Dutch invoices.

R2. Next, a set of constraints composed of the identified variables will be formulated. These constraints will be identified by analyzing the local logical relationships among invoice fields through dataset observation and tax law examination.

R3. These identified variables and constraints will set the basis for developing a practically relevant constraint satisfaction algorithm.

R4. The CSA will serve as the foundation to develop a practically relevant invoice extraction and analysis tool, that will be applied to and tested on a dataset of real-world invoices of a local Dutch SME.

4. RELATED WORK

Automated invoice field extraction has been a field of study reaching back at least two decades [12]. Field extraction is often the outcome of a series of steps, incorporating a range of different methods.

An early method proposed by [12] is structure analysis. This, however, assumes the presence of graphical lines and enclosing boxes, usage of certain keywords, and spatial relationships between the graphical and textual components, limiting this method to invoices that include those graphical features.
[2] proposes an approach by using the morphological information of the words in an invoice. For example, a column containing the lexical ‘total’ or ‘tot’, will be considered as the Total Amount of the invoice. An accuracy of 91.02% was achieved. However, this paper only considers extracting information about the article line items of an invoice, and assumes the presence of a structured table.

A repeatedly used approach is to first include a classification step, and extract fields once the class is known [15]. Classes in this context roughly correspond to the layout of the invoice and are often invoices from the same company, or made with the same accounting software. This classification step allows for automatic extraction of fields [15]. Classification techniques include the use of image features, structural features, and/or textual features of the invoice [1]. Correct classification rates, even with only a few documents in the learning set, have shown to be very accurate: above 90% [1].

[9] and [10] propose a system of methods for field extraction that are based upon structural similarities with previously processed invoices, either by using the structure of other very similar documents, or by composing a new structure class for the newly identified ones. This, however, requires a large database with previously analysed invoices for better results. In contrast, a survey shows that only 20% of large companies’ creditors send invoices frequently [11], so for most creditors the learning base of invoices will be small in practice. An accuracy rate of 76% was achieved for keyword structures, similar to this research’s output, in the unknown class case.

5. SYSTEM OVERVIEW

A system for automatic field extraction of invoices is built as described in the last research aim (R4). The system is comprised of several components, both existing and new. The input of the system is a Dutch invoice in image format. The system gives structured data about the financial information of the invoice as output. This output is verified by the identified constraints. The individual components that translate the input to the output will be described next. See Figure 3 for the system overview as a graphical representation.

The first component of the system is an optical character recognition (OCR) tool. The invoice in image format is processed by the OCR system, and outputs an array of textual, numeric, and financial field values present on the invoice. These fields are the input to be readily used by the next system component. There exist a wide variety of well-tested, robust, current solutions that do character recognition. In addition, character recognition is an extensive research topic on itself, and is not the core problem of this research, but a mere convenience feature to provide a more complete system from input to output. Therefore, an existing implementation of OCR is used that takes care of this component. In this case, the Google Vision API is used.

The second component takes care of data pre-processing. The array of fields, the output from the OCR tool, are filtered, cleaned, and verified to create a valid input for the algorithm in the next system component. As the OCR tool outputs all field values that are present on the invoice, but the algorithm is only concerned about the financial data, all textual and non-financial numeric fields are filtered out. The remaining financial fields are verified and when necessary processed to have a uniform financial format.

The third component is the CSA, the main part of the system, where the unstructured array of financial fields is structured in accordance to the identified variables and constraints. The CSA makes use of these constraints to test each possible combination of the input values to its variables. The constraints dictate if a possible solution is viable. The CSA outputs structured and verified data about the financial component of the invoice. As the number of possible combinations of variables can be large, backtracking is used to make execution time faster, but not lose any potentially viable results, this will be explained further in section 7.2.

In the next few sections a more detailed description about the specifics will be given. As the variables and the constraints are at the basis of the CSA, those will be identified first. Next, the specifics of the CSA are discussed. Lastly, results regarding the accuracy of the system are given.

6. CONSTRAINTS IDENTIFICATION

The constraints must be identified before writing the algorithm. First, we will regard tax law requirements for potential variables. Next, a real-world dataset of invoices is described and used to compare the identified variables to a real-world case. Lastly, these variables are captured in constraints, which will be used for the CSA in the next section.

6.1 Invoice Tax Law Requirements

A starting point for identifying variables lies at the Dutch tax office. A set of required fields is specified that Dutch invoices must adhere to by law [6]. All requirements that are not related to financial information, such as ‘company name’, ‘address’, and ‘date’, are disregarded. This gives the following set of rules. A tax-compliant invoice must state:

- the amount of product delivered;
- the total invoice amount excluding tax, ‘Subtotal’ (S);
- the tax rates used (i%n’s). The tax rates in The Netherlands are limited to three rates: 0%, 6%, and 21% [5]. These will therefore be the rates that are used for the tax brackets;
- the amount of total tax invoiced, ‘Total Tax’ (TT);
- if multiple tax brackets are used, for each tax bracket the amount of tax invoiced, ‘Tax Amount for i% bracket’ (Ti%), and the amount this is based on...
excluding tax, ‘Tax Basis for i% bracket’ (B_{i\%}) should be disclosed.

As our system is also built with practical relevancy in mind, the set of variables should also reflect a useful output structure of invoice data for Dutch SMEs. On the input side, this is done by examining the dataset of invoices for common variables and constraints. For the output, Dutch tax law is examined for required fields to be administrated. As in the previous case, non-financial fields will be disregarded, as the CSA is only concerned with linear constraints. The Dutch tax office states that the following financial fields should be saved in an administration system [7]:

- total invoice amount excluding tax (subtotal), specified per tax bracket if applicable;
- tax amount, also specified per tax bracket if applicable.

These fields are already included in our earlier set of identified variables.

### 6.2 Dataset Collection, Annotation

A dataset of invoices in image format is collected from a local Dutch SME. The purpose of obtaining and annotating this dataset is to firstly identify fields and constraints that occur in a real-world setting. This gives a more comprehensive view into the actual structure of invoices in practice, than merely looking at the tax law requirements itself. In addition, it is later used as a ground truth to assess and improve the accuracy of the tool.

The dataset, after selection and cleaning, consists of 241 Dutch purchase order invoices in image format, from the recent past three years, either sent to the company by email as an image or pdf, or scanned from paper invoices. The dataset is characterized by a total of 47 distinct vendors. The largest number of invoices from the same vendor is 23, the smallest is one. Only invoices with non-negative amounts were selected, so credit notes are not included in the dataset. For a more comprehensive view, invoices from different years with unusual combinations of fields were included too.

All 241 invoices were annotated by hand. For each invoice, the vendor of the invoice and all its financial fields were collected. These financial fields consist of the subtotal of the invoice, the total tax of the invoice, the total amount of the invoice, an optional added costs field, and for each included tax bracket the tax basis - the amount of which the tax amount is based - and the tax amount (VAT).

### 6.3 Dataset Considerations on Tax Requirements

Next, the annotated dataset will be compared with the tax requirements, to identify if the set of identified fields from the tax requirements should be adapted, expanded, or restricted based on real-world observation. This gives the following results.

Additions of fields were categorized as follows. One notable field that is not included in the tax requirements, but is present on all invoices in the dataset is the ‘Total Amount’ (TA) of the invoice. Because this field is always present in the dataset, it will be empirically assumed to be present on other real-world cases of invoices as well. Next, the addition of a field that can be categorized as ‘Added Costs’ (AC) is identified. It typically comprises of administration costs or shipping costs, and is excluded from the tax brackets. It is present on 5 of the invoices.

Omissions of tax law required fields were the following. In the case of an invoice with multiple tax brackets present (7 occurrences), the Total Tax field was missing in all cases. Although, each used tax bracket’s Tax Basis and Tax Amount were present, the sum of the Tax Amounts, the Total Tax, was not included explicitly. Next, in the case of only one tax bracket used, the Tax Basis field for that bracket and the Total Tax field were not stated explicitly. The explanation for this could be a layout space saving one, as in the case of only one tax bracket the Subtotal and the Tax Basis are the same amount. This is also the case for the Tax Amount for that bracket and the Total Tax. For SMEs, it could be viewed as redundant to duplicate these fields.

After considering all perspectives as described in the first research aim (R1), a set of variables is identified. These variables can be found in Table 1. Only AC is omitted as will be described in section 7.3.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TA</td>
<td>Total Amount of the invoice</td>
</tr>
<tr>
<td>S</td>
<td>Subtotal of the invoice</td>
</tr>
<tr>
<td>TT</td>
<td>Total Tax of the invoice</td>
</tr>
<tr>
<td>T_{i%}</td>
<td>Tax Amount for the i% tax bracket.</td>
</tr>
<tr>
<td></td>
<td>i = 0,6,21%</td>
</tr>
<tr>
<td>B_{i%}</td>
<td>Tax Basis which the Ti% is based on.</td>
</tr>
<tr>
<td></td>
<td>i = 0,6,21%</td>
</tr>
</tbody>
</table>

### 6.4 Linear Constraints: A System of Equations

After the variable identification, a set of logical equations composed of these variables can be deduced. These constraints are built in a data driven way; the logic between the fields in the annotated dataset can be deduced by composing equations between those variables and testing them by the ground truth values.

The following system of equations has been constructed, as was the intent of the second research aim (R2). See Table 2 for the results. In case of the CSP, these equations are regarded to be the set of constraints the algorithm must satisfy. For purposes of accuracy in the algorithm, the AC variable is omitted from the first formula, this will be explained in section 7.3.

### 7. CONSTRAINT SATISFACTION PROBLEM

In this section, the CSA will be described in detail. The CSA is composed by its variables and constraints. Specifies on variable and constraint identification is described in section 6. The resulting variables can be found in Table 1. The resulting constraints can be found in Table 2.

A variable in our case is an invoice field, such as TA or S. The domain of possible values is the set of processed field values read from the invoice image. The domain of possible values could differ per variable, for example, if a field value is already assigned to TA, this value will be excluded from the domain for S, as it is not duplicated on the original invoice also. The constraints are the logical relationships among the invoice fields, such as $TA = S + TT$. Each possible combination of field values of the variables is tested to satisfy all constraints. If a combination of field values satisfies all constraints it is regarded as a possible solution.
Table 2. Identified System of Equations of Dutch Invoices

<table>
<thead>
<tr>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4.1</td>
</tr>
<tr>
<td>4.2</td>
</tr>
<tr>
<td>4.3</td>
</tr>
</tbody>
</table>

7.1 Setting Variable Domains: Field Values

The field values read by the OCR tool from the invoice function as the domain of possible values a variable can take. This is because each invoice is regarded to have the information, all needed field values, present to satisfy the constraints. As the constraints are linear, and contain only financial data, the textual and numeric fields are filtered out. The financial fields are identified by format; each field value that does not comply to this format is regarded as a textual or numeric field, and is filtered out.

Looking at the OCR output, the financial fields can be categorized as either European financial style, with a comma as decimal mark and a dot as thousand separator, or American financial style. The format verifier, technically implemented as a regular expression, was checked to correctly identify only financial fields in accordance to the annotated ground truth. In addition, for practical purposes, all verified financial fields were translated to a uniform financial style, clearing all whitespace and currency symbols. Errors by the OCR tool were disregarded in the verification of the format verifier. These errors included not reading dots, commas, and ‘1’s, confusing a ‘4’ as an ‘A’, and a ‘3’ as an ‘S’, and splitting up a financial value in two due to whitespace.

A complication, however, arises when setting the variable domains to only the read field values from the invoice. Looking at the dataset, two categories of complications can be identified. The first occurs when an invoice only has one tax bracket, such as 21%, present on the invoice, as already mentioned in section 6.3. In this case, the fields TT and \( T_{21\%} \), and S and \( B_{21\%} \), have identical values by definition, and in practice have only one value instance per pair present on the invoice. The second case is one of missing zero instances. If a tax bracket is unused, the corresponding \( T_{ni} \)'s and \( B_{ni} \)'s have in all but one case no zero instances present on the invoice. For performance and implementation purposes, these two complications are dealt with by the algorithm, and not in the pre-processing component of the system.

7.2 Backtracking: Derived Constraints for Improved Execution Time

CSPs can be solved by using the generate-and-test paradigm. In this method, every possible combination of the values in the domain is assigned to the variables and tested for constraint satisfaction. However, this method is rather inefficient; the number of combinations possible is the Cartesian product of the individual variable domains. [13]

A more efficient method uses backtracking. This method instantiates the variables in sequence. This allows for interim verification of the partial variable assignment. If a partial assignment already violates a stated constraint, the algorithm can skip the remainder of this part of the execution, and continue with a new combination. [13]

The identified constraints in section 6.4 serve as the preemptive branch-cutters in the algorithm’s execution. For clarity, the identified constraints of Table 2 are rewritten as distinctive rules which evaluate only two variables at once. The constraints are linear and the possible domain values only include positive real numbers. These rewritten constraints will be called derived constraints and can be found in Table 3.

These derived constraints can improve the execution time as follows. For example, for the equation \( TA = S + TT \) (Table 2.1), if a value of ‘54.00’ is assigned to TA, and ‘95.00’ for S, it is of no use to evaluate further any assignment to TT. As there exist no negative values, the constraint cannot possibly be satisfied with this assignment. It violates the derived constraint \( TA \geq S \). In this way, the program can skip further assignment of all possible combination of variables for the remaining variables, but not skip any potential correct combination.

Table 3. Derived Constraints for Backtracking

<table>
<thead>
<tr>
<th>Base Constraint</th>
<th>Derived Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( TA \geq S )</td>
</tr>
<tr>
<td>2</td>
<td>( TA \geq TT )</td>
</tr>
<tr>
<td>3</td>
<td>( S \geq B_i ) for ( i = 0.6, 21% )</td>
</tr>
<tr>
<td>4</td>
<td>( B_i \geq T_i ) for ( i = 0.6, 21% )</td>
</tr>
</tbody>
</table>

7.3 Constraint Satisfaction Algorithm

In this section, the CSA is discussed, as mentioned in research aim 3 (R3). To allow for backtracking, the variables are assigned in sequence. After each assignment, the corresponding (derived) constraint (Table 2 and 3) is checked. If this assignment violates a constraint the execution skips the remainder of this assignment. To accomplish this, in practice, the value domains for each variable are filtered to only contain valid, non-constraint violating values.

The number of field values is uncertain, as it is read from a real-world invoice. It can be the case that more field values are present than the number of variables. It could be possible to have multiple value combinations that satisfy all constraints. So, the CSA should allow for multiple results. During the testing stage, it was observed that invoices with multiple zero ‘noise’ field values gave multiple results instead of one. The highest accuracy was achieved when a result was chosen that had no zero instances in the tax amount variables.

The CSA also accounts for the complications in the variable domains as mentioned in section 7.1. To solve potential missing zero-values for unused tax brackets, a zero-value instance is added to the domains of the variables \( T_{ni} \)'s and \( B_{ni} \)'s if no zero-instance is present. Next, the problem of invoices only displaying a single instance of the values for the variables TT and \( T_{6\%} \), and S and \( B_{6\%} \) for a single tax bracket invoice is solved in the following way. The original domain set for the second occurring variable in the algorithm’s execution (in this case the variables \( T_{6\%} \) and \( B_{6\%} \)) is extended to have a duplicate included of the earlier assigned values of the other variable (the assignments to TT and S). These two solutions...
are only used if no solution has been found in the original domain set of values. This is to first try to find a solution with the original field values, not cluttering it with potentially erroneous combinations.

During the implementation and testing stage, the first constraint in Table 2, TA = S + TT + AC, was altered. The system proved to be more accurate and had faster execution time when the variable AC was excluded. The worse accuracy was due to only having 4 invoices that included the AC variable. For all other invoices, this variable was essentially a wildcard, because unlike TT and S no other constraints defined this variable further. This resulted in the algorithm giving multiple erroneous results for invoices without an AC field.

Another issue was identified during the implementation and testing stage of the algorithm. Because companies have some degree of freedom to round their tax amounts [4], the constraints 4.2 and 4.3 (Table 2) cannot be checked for exact equality; a tolerance should be used. A low tolerance is very strict on the rounding and can potentially identify a correct solution as false. A high tolerance, on the other hand, allows for more values to fill the variables, as the constraint also allows for more freedom in assignment. A tolerance of 0.03 Euros eliminated all identified rounding error results, while still maintaining a high accuracy.

See Pseudocode 1 for the high-level representation of the CSA. The input of the algorithm is an array of financial field values, $D_{\text{full}}$, which are uniquely identifiable by instance, after data pre-processing. The output is a Fields object, implemented as a dictionary type, which structures all variable-value pairs.

Because the solution is found via two separate algorithms and they only differ by a few lines, they are combined. Line numbers marked with an asterisk (*) are only used in the second ‘missing-instance-case’ algorithm, which accounts for missing instances for the variables TT and T$_{\text{t21}}$ and S and B$_{\text{b0}}$. In addition, the missing-instance case CSA is run secondly, only if no result is found in the first case CSA.

In this code, the domain values for each variable are denoted as $D_{v}$, where v is the variable this domain corresponds to. The method ‘addZero’ adds a zero instance (an element with ‘0.00’ as its contents) if no other zero instance is present. Line numbers made italic represent the backtracking rules. The ‘Fields’ variable that is passed along the methods is a custom dictionary-like object that stores intermediary field-value candidate pairs. The ‘FieldsSet’ is an array of found Fields that satisfy all constraints.

---

**Pseudocode 1. Combined CSA**

```plaintext
[global] FieldsSet // Array of 'Fields'
[global] tolerance = 0.03
getResult($D_{\text{full}}$) -> returns Fields
1:  guessTA($D_{\text{full}}$) // mutating function
2:  firstFields = FieldsSet.first
3:  for each Fields in FieldsSet
4:      if Fields["tt"] > 0
5:          return Fields
6:  return firstFields

guessTA($D_{v}$)
1:  for each ta in $D_{v}$
2:      // Removes instance of ta from set $D_{v}$
3:      $D_{v} = D_{v} - \{ ta \}$
4:      Fields["ta"] = ta
5:  guessS($D_{v}$, Fields)
```

---

8. RESULTS AND EVALUATION

The dataset of 241 invoice images was used to test the accuracy of the system. A correct result in this case would be if the system’s assigned values to all variables are all equal to the values in the manually annotated ground truth of the dataset.
An incorrect solution would be if one or more variables (per invoice) do not have the same values, or if a result is empty. In the case of an empty result, the system could not find a combination of the field values that satisfied all constraints. All errors were investigated and categorized.

The results are as follows, as summarised in Table 4. The overall system did not correctly solve 36 out of 241 invoices, this gives an accuracy of 85.1%. Out of these 36 errors, 11 were due to the incorrect reading of values by the OCR tool. The overall OCR error rate is therefore 4.6%. The remaining 25 incorrect answers are system errors; they are caused by the algorithm’s shortcomings, excluding the OCR errors. The system error rate is therefore only 10.9%. In addition, out of the 36 total errors, the system did not give a result in 29 of those cases. This indicates a low ‘false positive’ rate, only in 2.9% of the cases the system explicitly gave a false answer.

<table>
<thead>
<tr>
<th>Overall Accuracy</th>
<th>OCR Error Rate</th>
<th>System Accuracy</th>
<th>False Positive Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>85.1%</td>
<td>4.6%</td>
<td>89.1%</td>
<td>2.9%</td>
</tr>
</tbody>
</table>

The system errors can be categorized as:

(i) an error in rounding of one cent of the formula TA = S + TT, has 3 occurrences, all from invoices of the same vendor, apparently an error in the vendor’s administration program;
(ii) a missing TT field on the invoice itself, in this case the invoice had multiple tax brackets, but did not state the aggregate total tax of each tax brackets tax amount, has 7 occurrences;
(iii) the invoice stated extra costs (AC) that were not taxed or included in the subtotal, as mentioned before this field was excluded from the constraints due to worse performance, has 4 occurrences; and
(iv) other not clearly identified reasons, has 11 occurrences.

The runtimes for processing each invoice, the system stages after OCR, are measured on the dataset. The time measurements were taken on the same 3.3GHz processor, and the program was implemented in the compiled programming language Swift. It took the system on average 0.25s to process one invoice, with a minimum of 0.01s, a maximum of 20.8s, and a standard deviation of 0.36s.

The runtimes without any form of backtracking are also measured. Because processing times can be rather high with the standard generate-and-test paradigm, only a few invoices handpicked by number of field values were used for testing. While invoices up until 5 fields still processed in less than a second, these processing times quickly soared as the number of fields increased. An invoice with 11 fields already took 35 minutes to compute, and 12 fields accounted for 110 minutes of computing time. One can image the amount of time the invoice with the highest number of fields (40) will take to process.

9. DISCUSSION AND FUTURE WORK

The results look promising. Accuracy rates of 76.4% up until rates of about 91% were achieved in related work. However, this paper’s definition of accuracy is stricter. Related work defines it on individual field basis, where this research considers the entire invoice: every field on the invoice should be identified correctly for a correct case. In addition, only [9], with an accuracy rate of 76% works with unknown class invoices and a similar output structure, which is more comparable to this research’s approach. The results of this research can therefore be regarded as comparable overall, to better in the most similar case.

False positive rates were not mentioned in related work. However, the achieved rate of only 2.9% can be regarded as promising. This shows the usefulness of the identified constraints, which contain the logic about invoices, to allow for verification of the answers. The low false positive rate could prove useful in a business environment.

Looking at the processing times, the usefulness of backtracking becomes clear. Considering the low average and the fact that in only 4 of the 241 cases the runtime was higher than 1 second, it would be very acceptable from a business’s perspective in time wise. Without backtracking the processing times quickly soared into the order of hours. These are not acceptable waiting times; manual processing would be faster.

Although the results are satisfying, there are several ways the system can be improved, either by optimizing the algorithm, improving the data pre-processing, or increasing OCR performance.

Even though the computing time per invoice is very satisfying overall, and acceptable in even the extreme cases, there may exist invoices with a lot of financial field values. In those cases, the computing time could be exponentially higher. A better format identifier in the pre-processing step for the financial field values could be implemented. One that is better at selecting field values to more accurately limit the number of values the algorithm has to compute.

The added cost (AC) constraint was omitted due to performance and accuracy issues. Although a low number of invoices in the dataset had added costs, it can prove to be useful to find a way to adapt the algorithm to include added cost but not worsen the results of invoices without the added costs, as was the case now.

In all 7 invoices that had multiple tax brackets, the total tax (TT) field was missing. Although this field should by law be explicitly stated, in practice it is apparently not adhered to, and should be accounted for.

The OCR tool accounted for some errors. Either not recognizing a field value at all, or misreading a value, for example omitting a comma, reading a 3 as an 8, separating one field value in two, etc. For improved results, a better OCR tool could be selected, or the OCR output could be post-processed to gain better results, such as identifying missing decimal separators, or joining two unjustly separated field values together.

The tool is now limited to invoices in The Netherlands. With the same research approach constraints and variables could be identified to allow for the system to be implemented in other geographical areas as well.

As the system is only tested on a relatively moderately sized dataset from one company, it is highly recommended to test and adapt the system on more invoice datasets. One can imagine that datasets of (recent) invoices can be relatively hard to obtain. Detailed financial information is not something a company is comfortably willing to share. Preferably, a dataset of a company with more unusual cases, such as invoices that have multiple tax brackets, or more invoices not in the 21% tax bracket is used. This dataset was most prevalent on single tax bracket (of 21%) invoices. As the 21% tax bracket is the standard rate for products and services [5] it is not surprising.

It would also be advised to combine the previously stated methods that use graphical, structural, positional information
for invoice field extraction with the newly proposed method. As this method only concerns financial information and not information such as addresses, company names, which the other methods do, a combination would give a more complete system. In addition, our method could serve as a verification step for the extracted financial information in the other methods, as it is based on identified constraints.

10. CONCLUSIONS
In this paper, an implementation of the CSP method was used to build a field extraction system for Dutch invoices. In building the CSA, the variables and constraints were identified first.

The system was set up to have a comprehensive, but also practical view. Therefore, the identified variables were composed with multiple perspectives in mind. The combination of tax law compliance, real-world dataset observation, and practical application relevance resulted in an identified set of variables, as can be seen in Table 1.

Next, constraints composed of these variables were identified by deducing logical relationships among these variables in the ground truth of the dataset, which can be found in Table 2.

These constraints and variables set the basis for developing the CSA, whose pseudocode can be found in Section 7.3. To improve execution time the backtracking paradigm was implemented. To provide a complete system, an (existing) OCR tool and a (new) data pre-processing tool were also implemented.

The system gave an overall accuracy of correctly identified invoices of 85.1%. A system accuracy (without OCR-errors) of 89.1% was achieved. In addition, due to the method being logic-based, had a low false positive rate of 2.9%. This could serve beneficial to combine this method with related work. Among the errors, a few clear categories were identified, which could lower system errors in future work. In addition, it is advised to test the system on other, more diverse, datasets as well, and adapt the system to be used in other geographical areas as well.

11. REFERENCES
DOI= https://doi.org/10.1109/ICPR.2004.1334166.
DOI=https://doi.org/10.1016/0004-3702(87)90002-6
DOI=https://doi.org/10.1109/ICIP.2007.4379838
DOI=https://doi.org/10.1109/ICDAR.2007.4378726
DOI=http://doi.org/10.1007/978-3-540-28640-0_43
DOI=http://doi.org/10.1145/1860559.1860599