9 Template Filling

9.1 Tasks of a TE-Agent

After name extracting, extracted instances are fed into the TE-agent. In the phase of template filling the TE-agent tries to assign the right instances into the template. As discussed in Section 5.1.3, there are at least three tasks of a TE-agent:

1. Extracted instances have to be assigned to the elements in template. If the relationship between instance and corresponding element is one to one, the assignment is trivial, otherwise some intelligent sorting and configuration is needed.

   Actually, even if NE-Agent produces 100% precision and recall at NE-level, it can cause very poor performance of whole template if the instances are false assigned. For instance, given defined template as address book, the NE-agent extracted two names. The first one is “Mayer”, which has address “Address 1”, and another name is “Mueller” with address “Address 2”. All the instances are right. That is, NE-agent performs perfect result. In the template filling, however, the TE-agent assign the first name with the second address. Then, the final result becomes worse. This template instance becomes immediately unusable. Hence, with this example, TE-Agent must be able to assign the name element together with his own phone number and address to a template instance.

2. Normally NE-Agents can not extract information perfectly, that is, with 100% precision and recall. On the other hand, for structured and semi-structured documents, some spatial relationships between template elements can provide useful information to enhance the extracting performance. Hence, TE-agent can be used as post-processing to improve the extracting performance using some spatial relationships. For instance, if there are some wrong instances extracted by NE-agents, the TE-agent has to solve conflicts carried out by such wrong hits. In this case, some wrong instances have to be dropped.

3. In the real life applications, the template model can be much more complicated than the laborious sample one with a few flat elements. Template filling with multi-instance and various cardinality of several elements must be supported in the practical domains. For example, if the document has multi-instance, TE-agent has to duplicate the template automatically and fill the right content instance into template.
The working steps in the phase template filling is described as following: as input the TE-Agent receives a list of instances extracted by NE-Agent. Then, TE-Agent get one instance from the head of the list and tries to fill it into the template. If there is no conflict by template filling, the TE-Agent continues to process next instance stored in the list. If conflict occurs (for example, more than one instance candidate for a template element), TE-Agent hat to solve the conflict, make decision and select the right instance for the element. If TE-Agent detects that the document has multi-slot template property, TE-Agent duplicates the template and fills the future instances into the new template.

In the template filling phase, solving conflicts is one of the most important steps. In this section we will introduce a TE-Agent using a new approach, which is used to perform results with high precision and recall. This new approach is designed with respect to spatial relationship under elements and for solving conflicts. The new approach is explained and the advantage is discussed in this section.

9.2 The Spatial Model

In many practical applications the IE system has to face two problems in template filling, namely the conflicts and the multi-instance. For the first problem, system has to drop the false instances and keep the right one. To solve this problem, we observed that the interrelationship under elements can provide useful information. For instance, the spatial relationship can be used to filter the bad instances, which do not conform to spatial constraints. For the second problem, even if NE-agent extracted instances perfectly (with 100% precision and recall), the final filling result could be very bad, because if elements instances are assigned in false templates, the total template can be considered as false. Furthermore, spatial relationship can also be used to border the multi-instances. Spatial model is therefore developed for template filling with respect to spatial and ordering of elements in template. This section described the contribution of this model.

For the class of documents that is of interest here (structured and semi-structured), significant spatial relationships are expected between the individual features of a given template – in other words, the order in which the feature instances occur is not completely random. Furthermore, the capture of these spatial relationships is a task that can be automated and achieved in a non-intrusive manner – i.e. based on observing the actions of a user carrying out the task of manually transferring information from text into the template7.

---

7 This assumes that the relevant text position is known to the computer at the time that a user enters a value for the feature of the template. This assumption is valid for the test environment of this work where values are entered either via Drag&Drop
The model of the spatial relationships between the feature instances corresponds to the profile of our TE-Agent. Note that the user who, in effect, trains the TE-Agent need not be an expert user with respect to the information extraction technology; they merely must be capable of reliably carrying out the template filling task manually.

Several types of possible special relationship are captured here (examples):

- **DirectBefore**\((A,B)\) – i.e. an instance of content B comes directly before content A
- **DirectAfter**\((A,B)\) – i.e. an instance of content B comes directly before content A
- **Before**\((A,B)\) – i.e. an instance of content B comes somewhere before content A
- **After**\((A,B)\) – i.e. an instance of content B comes somewhere before content A
- **Contain** \((A, B)\) – i.e. an instance of content B is only a part of content A

These relationships are created based on the statistical analysis of multiple manually filled templates (for the experiments described next section about evaluation, the TE profile is generated from 100 manually filled templates). The relationships are further sub-divided into:

- **Constraints** – i.e. spatial relationships that cannot be invalidated (they were observed always to hold)
- **(Weighted) Preferences** – relationships that are not always valid but have a statistically significant frequency. These relationships are typically represented as a numeric weight < 1.0.

An example of part of the TE-profile for one of the experimental tests is displayed below. We use the Job Advertisement (description about Job Advertisement see Section 4.3) to illustrate the model. Note that the After/Before/Containment relationships generally result in constraints where as the more specific Direct Before / Direct After relationships generally result in weighted preferences: this reflects a persistent overall structure for the

---

from text in the template or by highlighting a relevant text position and pressing an “enter value” button
description of each template instance although the detailed ordering of individual content instances is less well defined. This observation justifies our classification of the document type as being semi-structured.

| After Title: | Salary |
| After City:  | State  |
| ...         |        |
| Direct After Title: | Salary: 0.333333, |
|              | Req_EXP_Year: 0.259259 |
| Direct After City: | State: 0.815385 |
| ...         |        |
| Direct Before Req_EXP_Year: | Title: 0.350000 |

### 9.3 Template Filling with Greedy Voting Algorithm

This section introduces a local optimisation algorithm for template filling based on spatial model. Examples are given to explain the procedure.

The TE-Profile, described above can be used in order to filter the results communicated from a collection of NE-Agents – i.e. it is to be used to rule-out falsely identified feature instances while preserving the correctly identified feature instances. There are a number of possible ways in which the statistically derived information could be exploited. The algorithm tested here resolves conflicting feature instances based on a greedy voting algorithm.

The first stage is to identify conflicts. For example, for our TE-Agents, three types of conflict are possible (based on the assumption concerning document structure):

- *Multiple instances for a given feature have been found, although it is known in the template that the feature has cardinality 1*
-  
  - *The instances of two different features occur in an order that is contradictory with respect to known constraints of the TE Profile.*
-  
  - *A given part of the text is used as the basis for the value of more than one template feature.*

As an example, assume the following tuple reflects the order in which feature instances for a given template instance occur within the document (as communicated by the NE agents):

\[(\text{Salary}_1, \text{Title}_1, \text{Reg_EXP_Year}_1, \text{City}_1, \text{Salary}_2)\]
The conflicts that are generated (with respect to the previously given example TE-Profile) are:

- **Conflict1** (Salary_1, Title_1) -> Salary feature instance must come after Title feature instance
- **Conflict2** (Salary_1, Salary_2) -> Multiple instances of same feature

Conflicts can be resolved by eliminating one of the conflicting feature instances. This is achieved by allowing all feature instances outside of the conflict to vote for each of the conflicting instances. Namely:

\[
\text{Votes}(F_{\text{conf},i}) = \sum \text{Vote}(F_{\text{conf},i}, F_j), \text{ where } F_j \text{ is a feature instance not in the current conflict.}
\]

The values for Vote(F_{\text{conf},i}, F_j) are as follows:

- -1 if the spacial relationship between F_{\text{conf},i} and F_j violates a known constraint
- +1 if the spacial relationship between F_{\text{conf},i} and F_j conforms to a known constraint
- -W if the spacial relationship between F_{\text{conf},i} and F_j violates a known Preference of weight W
- +W if the spacial relationship between F_{\text{conf},i} and F_j conforms to a known Preference of weight W
- else 0.

For example, for Conflict1

\[
\text{Votes}(\text{Salary}_1) = \text{Vote}(\text{Salary}_1, \text{Reg}_\text{Exp}_\text{Year}_1) + \text{Vote}(\text{Salary}_1, \text{City}_1) + \text{Vote}(\text{Salary}_1, \text{Salary}_2) = (0.0) + (0.0) + (0.0) = 0.0
\]

\[
\text{Votes}(\text{Title}_1) = \text{Vote}(\text{Title}_1, \text{Reg}_\text{Exp}_\text{Year}_1) + \text{Vote}(\text{Title}_1, \text{City}_1) + \text{Vote}(\text{Title}_1, \text{Salary}_2) = (0.26 + 0.35) + (0.0) + (1.0 - 0.33) = 1.28
\]

The feature instance that receives least votes is eliminated from all conflicts. In other words, Salary_1 would be eliminated on the strength of Conflict1. As this also resolves Conflict2, no further action need be taken, i.e. the resultant feature instance set after the filtering by our TE-agent would be:

\[(\text{Title}_1, \text{Reg}_\text{Exp}_\text{Year}_1, \text{City}_1, \text{Salary}_2)\]
As shown in the above example, the elimination of a feature instance as a result of resolving one conflict is propagated to other conflicts. This means that the performance of the TE-Agent is sensitive to the order in which conflicts are resolved. This problem is partly solved by randomly selecting the conflict to be resolved, but there is still a significant danger of reaching a locally optimal solution. That is to say, the feature instance eliminated by resolving the first conflict is not necessarily the best feature instance to be eliminated from other untried conflicts. In this sense, we are applying a greedy optimisation algorithm. Future work will address this problem and provide a more globally optimal solution at the cost of increased computational effort.

9.4 Evaluation and discussion

In this section, the performance of the TE-Agent created by the above method is tested on two standard data sets in order to determine its effectiveness at compensating for low-performance NE-Agents: Job advertisements and Rental Ads. Both data sets are downloaded from Muslea's RISE Information Extraction Repository (http://www.isi.edu/~muslea/RISE/repository.html).

9.4.1 Job Advertisements

The data set for the experimentation is a set of 300 job advertisements (see Califf 1997 for details). Each job advert is sent as a separate email and from this basis text a template should be filled per email. Each template comprises 17 features such as Title, City, State, Salary, Required Years of Experience, etc. Typically, the value for a feature occurs exactly once in the text, although for any given email some values may be missing.

The data set is divided into three parts of 100 job adverts. The performance of the knowledge extraction system was evaluated using a cross-validation approach; i.e. each experiment was repeated three times, training on each of the 100 job subsets and the evaluation the performance on the remaining 200 jobs.

For the first experiment, the TE-Agent is trained automatically. The user manually performs the template filling task on the 100 job adverts used for training and the resulting filled templates are given to the previously described learning algorithm in order to create a TE-Profile representing the spatial dependencies between the different features.

The low quality NE-Profiles are derived automatically – i.e. from all examples of the text used to fill a given slot in the template, the set of keywords is extracted and used as the basis for an NE-Profile. These low quality NE-
Agents then perform a relatively crude keyword matching approach that typically yields a high degree of recall but poor precision.

The results of the experiment are summarised in the following table. To make comparison, the results of two other systems LearningPinocchio (Ciravegna 1999) and RAPIER (Califf 1997) are shown also in the table. The performance (precision and recall) is measured by first determining the total precision and recall for each feature independently and then taking the mean across all features.

<table>
<thead>
<tr>
<th>System</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>CapturePlus</td>
<td>86.9%</td>
<td>93.7%</td>
</tr>
<tr>
<td>RAPIER</td>
<td>65%</td>
<td>89%</td>
</tr>
<tr>
<td>LearningPinocchio</td>
<td>86%</td>
<td>82%</td>
</tr>
</tbody>
</table>

For a deep analysis the precision / recall results are measured both immediately after the NE-Agents have been applied and again after the TE-Agent has operated on the results of the NE-Agents.

<table>
<thead>
<tr>
<th>Experiment (Low Quality NE) – After NE-Agents</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment (Low Quality NE) – After TE-Agent</td>
<td>86.9%</td>
<td>93.7%</td>
</tr>
</tbody>
</table>

Here we can see that the automatically generated TE-Agent improves the overall precision at a minimal cost to the overall recall.

A more detailed analysis of the above results reveals that a majority of the feature values were extracted with near 100% precision / recall and that the sub-optimal performance can be attributed to the following 4 features:

<table>
<thead>
<tr>
<th>Feature (After TE_Agent)</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>72.2%</td>
<td>93.8%</td>
</tr>
<tr>
<td>City</td>
<td>74.8%</td>
<td>94.0%</td>
</tr>
<tr>
<td>Title</td>
<td>51.5%</td>
<td>91.2%</td>
</tr>
<tr>
<td>Des_Year (After TE_Agent)</td>
<td>34.0%</td>
<td>100.0%</td>
</tr>
</tbody>
</table>

This opens up a third possibility whereby the hand-crafted NE-Agents are used for these problematic features and the automatically generated NE-Agents are relied on for extracting the other features. With this third configuration for the experiments, the following set of results was achieved:

<table>
<thead>
<tr>
<th>Experiment 3 (Low Quality NE + 4 High Quality NE) – After NE-Agents</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>89.6%</td>
<td>93.4%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Experiment 3 (Low Quality NE + 4 High Quality NE) – After TE-Agent</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>96.6%</td>
<td>92.6%</td>
</tr>
</tbody>
</table>
In both experiments above, the automatically generated TE-Agent significantly improves the precision of the overall system without significantly damaging the recall.

### 9.4.2 Rental Ads

We used again Rental Ads to evaluate TE-Agent. For this experiment, both NE-Agent and TE-Agent are trained automatically. We used a token-based Rule Extractor to train and extract the NE values.

As for WHISK (Soderland 1999) we used a training set of 400 examples and a test set of a further 400 examples. In this experiment the recall and precision are improved with more examples. Recall begins at 88.5% from 50 examples and climbs to 95.0% when 400 examples. Precision starts at 87.0% from 50 examples and reaches to 97.7% when 400 examples. The results are show in comparison with WHISK as following:

<table>
<thead>
<tr>
<th></th>
<th>Examples</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHISK</td>
<td>50</td>
<td>94%</td>
<td>73%</td>
</tr>
<tr>
<td></td>
<td>400</td>
<td>98%</td>
<td>94%</td>
</tr>
<tr>
<td>CapturePlus</td>
<td>After NE-Agent</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>73.0%</td>
<td>91.4%</td>
</tr>
<tr>
<td></td>
<td>250</td>
<td>84.5%</td>
<td>95.8%</td>
</tr>
<tr>
<td></td>
<td>400</td>
<td>90.2%</td>
<td>97.0%</td>
</tr>
<tr>
<td>After TE-Agent</td>
<td>50</td>
<td>87.0%</td>
<td>88.4%</td>
</tr>
<tr>
<td></td>
<td>250</td>
<td>95.1%</td>
<td>94.7%</td>
</tr>
<tr>
<td></td>
<td>400</td>
<td>97.7%</td>
<td>95.0%</td>
</tr>
</tbody>
</table>

As same as the first experiment, there are good recall and poor precision after NE-Agent processing. After TE-Agent the precision is improved considerable with a slightly loss of recall. For example, in Rental Ads the system reaches 85.4% precision and 95.8% recall after NE-agent processing with 250 training examples. After TE-Agent, we get 95.1% precision and 94.7% recall. Figure 39 shows the total learning curve for the Rental Ad domain.
After the deep analyse of the results, we observed that the TE-Agent becomes unchanged after 100 examples. After analysing of the individual results, we found that with 100 examples, the precision of the elements neighborhoods and Price is already 90%, while the element number of bedrooms reaches only 34% precision. In the document, there is normally an acronym of bedrooms (such as "bd", "bdrm" etc.) after the to be extracted number. Because the Rule Extractor needs more examples to learn all acronyms as postcore, the extracted rules with 100 examples cover only a part of these acronyms.

9.4.3 Discussion

The spatial model based TE-agent is designed especially for semi-structured and structured documents. Within the scope of our agent-based model for the template filling task we have investigated the two possibilities of either positioning the precision early (i.e. in the NE-Agents) or late (i.e. in the TE-Agent(s)). However, our investigation has led to the interesting observation that, for an optimal performance at minimal cost information extraction system, a combination is required. Moreover, the determining in which NE-Agents it is worthwhile investing development effort can be guided by first testing the TE-Agent with crude, automatically generated NE-Agents.
These leads to a proposed methodology for the construction of a knowledge extraction system, at least for the sub-class of problems that involve template filling from semi-structured documents:

1. A non-expert manually performs the template filling task on a number of example documents
   - An automatic learning technique is used to construct a TE-Profile
   - A set of primitive NE-Agents are constructed based on the known values of template slots

2. The initial knowledge extraction system is applied to a further set of test documents
   - The performance W.R. each feature / NE-Agent is evaluated
   - The dependencies between different features in the TE-Profile are evaluated. The TE-Profile can learn captures dependencies between various NE-Agents. For instance, if NE-Agent X has a strong spatial relationship to NE-Agent Y - i.e. an instance of X can help correctly identify an instance of Y, it may be sufficient to only invest development effort in the X NE-Agent.

3. An expert user manually trains NE-Agents only for the selected critical features

Now we discuss about the precision and recall deeply. Many systems confuse these values in NE and TE-agent. Actually, in the most systems, the precision and recall for each template element are normally calculated independently. In fact, we have to define the precision and recall after template filling. If we consider the whole template as an atomic element, the value is calculated then more stringently. That is, only if all elements in template are well filled, this template is defined as right one, otherwise as false. Thus, the precision and recall after template filling has to gain total different value. The reason for such consideration is, that for some systems, the requirement of the right filling template is high. Moreover, the question is, how we can with these two numbers to judge the quality of IE-system. For some domains, the precision is highly required, while recall is not very important. For example, extracted information is used for statistical manner. The requirement in this case is, that all extracted information must be the right. However, it is unnecessary to extract all information in documents for statistical sampling. In this case, precision with 95% is not satisfactory. On the other hand, some domains need high recall result and use manual revision and correction to improve the precision. Then, the recall with 95% is considered as a bad result. Hence, alone with the number we can not judge the quality of system. More tailored scores should be introduced. For example, a weighted precision and recall scores with respect to expectation score.
In the current implementation the TE-Agent uses the greedy voting algorithm. The advantage of such algorithm is the linear processing performance. A weakness is that the greedy voting algorithm selects merely the "best" candidate. The algorithm is not able to decide whether all candidates are false. In this case, one of a false value will be assigned as the "best" result and filled into template. This is also a reason why the system can not reach 100% precision. We also tried to use Tversky-Similarity principle to model the order of contents as sequences. The shortcoming of this algorithm is that we have to test all possible combinations of different result to find the best combination with the most similarity with examples. The algorithm becomes practical unusable if there are more than five elements in template. The main reason why we use the local optimisation method (such as greedy voting) is that the system must perform a real time extracting to ensure that the user does not have to wait for a long time to get the result.

9.5 Related Work

Most of IE-systems do not have a explicit and stand alone TE-Agent. Actually, many IE-systems try to use one model to cover both NE and TE properties. However, in the algorithms used in the systems, some statistical models are used to enhance the performance with respect to the spatial relationships between template elements.

One well known statistical technique is Bayes. Bayes sums evidence provided by all tokens individually, including the tokens in a fragment's context. Freitag used this technique to extract information based on prediction of position and length (Freitag, 1998b ). In the evaluation of Freitag’s algorithm, system tries to identify the name of the speaker in a seminar announcement. This problem can be modelled as a collection of competing hypotheses, where each hypothesis represents that a particular fragment gives the speaker's name. In this case a hypothesis takes the form, “the text fragment starting at token position p and consisting of k tokens is the speaker.” (Call this hypothesis $H_{p;k}$.) In the case of a seminar announcement file, for example, $H_{309;2}$ might represent the expectation that a speaker field consists of the 2 tokens starting with the 309th token.

Freitag evaluated this Bayes model in seminar announcement. The size of documents is small and each document contains only one description of seminar announcement. Such restrictions enable the Bayes model to predict the position and length of tokens. According to the assumption of Freitag, in a typical information extraction problem field instances are short and do not vary much in length. Thus, this algorithm is only suitable for such semi-structured documents with short contents which have almost constant length. This restriction reduces the using of Bayes rules in IE. However, for such defined
document type, Bayes Rules seem to be very robust and guarantee a high performance result.

Hidden Markov models (HMMs) are a type of probabilistic finite state machine, and a well-developed probabilistic tool for modelling sequences of observations. Systems using HMMs for information extraction include those by Leek (1997), who extracts gene names and locations from scientific abstracts, and the Nymble system (Bikel et al. 1997) for named-entity extraction. These systems do not consider automatically determining model structure from data; they either use one state per class, or use hand-built models assembled by inspecting training examples. McCallum (1999) hand-build multiple HMMs, one for each field to be extracted, and focus on modelling the immediate prefix, suffix, and internal structure of each field; in contrast, Seymour (1999) focus on learning the structure of one HMM to extract all the relevant fields, taking into account field sequence.

In a sample system of Seymour (1999), HMM is used for information extraction from research paper headers by formulating a model in following way: each state is associated with a class to be extracted, such as title, author or affiliation. Each state emits words from a class-specific unigram distribution. The class-specific unigram distributions and the state transition probabilities can be learned from training data. In order to label a new header with classes, the words from the header have to be treated as observations and recover the most-likely state sequence with the Viterbi algorithm. The state that produces each word is the class tag for that word.

In order to build an HMM for information extraction, people must first decide how many states the model should contain, and what transitions between states should be allowed. Each word in the training data is assigned its own state, which transitions to the state of the word that follows it. Each state is associated with the class label of its word token. A transition is placed from the start state to the first state of each training instance, as well as between the last state of each training instance and the end state. Model structure can be learned automatically from data, starting with either a maximally-specific model, using a technique like Bayesian model merging.

However, this system must pre-process documents to format documents with some tokens (such as <ABSTRACT>, <EMAIL> etc.). Such tokens are described by regular expression manually. Actually, Seymour used HMM to model the order of template elements. In this sense, HMM acts as a stand-alone TE component to fill the elements into a defined template. The advantage of HMM for modelling order relation of TE-elements is that HMM is a global optimisation algorithm. That is, the selection of the best sub-sequence or combination of elements is calculated with respect of all relevant TE-elements. Consequently, this causes a problem of time performance. Evaluations showed satisfied results with very simple template, which
contains less than 5 elements. If there are more than 50 elements (it means that there are at least 50 states in HMM) in template, lots of examples are needed for training HMM in the training phase. In the extracting phase, the calculation with Viterbi algorithm can be very computational intensive.

Compared to HMM, the spatial-model and greedy-voting algorithm performs a local optimisation algorithm. The advantage of such local optimisation algorithm is the linear processing time. In many practical domains (such as product catalogue, online travel information), one template has normally more than 50 elements. In this case, using global optimization algorithm such as HMM becomes less reasonable. Because the time for training with amount of manually created examples and the time for extracting with computational intensive Viterbi algorithm expand quickly.

Most recently, Sarawagi (2002) uses similarity functions to find out the most similar instance sequence. Such similarity functions are for example edit distance, soundex, N-grams on text attributes, absolute difference on numeric attributes. To eliminate the duplication, system calculate at first all possible instance combinations with defined similarity functions. The output of this step is a list of relevance values generated by various similarity functions. The selecting algorithm is then able to classifier the most similar sequence after mapping this list of values to the trained example value list. For a defined domain, this approach can provide good performance. However, because the similarity functions are domain-specific, they have to be redefined when importing to new domains. For practical using, such importing and remodelling becomes too comprehensive to use.