CVAnalyzer: A Semantic Tool to Accelerate HR Recruitment Process

Rifat Jafrin
7003560
r_jafri@encs.concordia.ca
Dept. of Computer Science & Software Engineering
Concordia University

***Abstract*— Finding the right candidate for the right position in a timely manner from a large collection of CV (Curriculum Vitae) is a challenging issue in HR recruitment process. This paper presents *CVAnalyzer*, a semantic tool to provide related CVs and ranked CVs based on the job description written in OpenOffice. *CVAnalyzer* uses GATE (General Architecture for Text Engineering) for finding the relevant CVs and ranking them from a collection of CVs. It utilizes the *Semantic Assistants* to provide service in OpenOffice as a plugin. *CVAnalyzer* reduces the effort for matching and ranking CV significantly with compared to the manual process which is reflected in experimental results. It also reduces the human error involves in manual process. The concept presented in this paper is supported by necessary theoretical model and experimental evaluation.**

I. Introduction

Job Recruitment process is one of the most important task of Human resource management department which involve finding out the most relevant CVs when a job offer is posted. The overall performance and prosperity of an organization depend on selecting the right candidate . It is often necessary to go through each resume to see the skill, qualification etc and match them with respect to job post to find the right person who suits best for that job. However evaluating CVs and determining the most suitable CV for a certain job is a tedious and time consuming task. By enhancing an automated system that match the key features of job posting and CV can save more time, cost and effort of an organization.

Our proposed semantic tool *CVAnalyzer* finds the relevant CVs and sort them corresponding to a job description (we refer *JobPosting* to the 'job description' from now on). *CVAnalyzer* utilizes the *Semantic Assistants* framework to work with OpenOffice. It extends GATE [3] for semantic processing and create annotations from the CVs and *JobPosting*. We have developed a processing resource *cvRanker* which is used in GATE to compute the relevant CVs and sort them using the annotations, which also generates HTML report. The basic idea behind finding the relevant CVs is to find the CVs having one or more annotations which are also present in the *JobPosting*. We have defined a mapping which assigns score to each category used for annotation matching. *CVAnalyzer* sorts the CVs based on the computed scores using this mapping.

The main contribution of this work is to develop an automated tool to accelerate the HR recruitment process. In this paper, we focus on the IT related jobs. It is possible to extend *CVAnalyzer* for jobs in other domains.

The rest of this paper is organized as follows. Section II presents related works. The mechanism of *CVAnalyzer* is explained in Section III. Algorithm for finding relevant CVs and ranked CVs is presented in Section IV. In Section V, we present the performance evaluation of *CVAnalyzer*. Finally, we conclude and outline our future research goals in Section VI.

II. Related Works

Creating right annotations for both CV and *JobPosting* is the first and important task of job recruitment process using semantic technique. Several approaches are proposed by research community to improve the searching and finding related CV corresponding to a *JobPosting*. Some of the research works focused on developing ontology whereas others adopt on semantic annotation approach to enrich the CV and *JobPosting* documents. An Electronic Recruitment Ontology is introduced in [9] which is shared by both job seekers and recruiters to annotated the CVs and Job offers. The proposed semantic-matching algorithm finds the related CV using the annotations. Amdouni and Ben Abdessalem [2] used

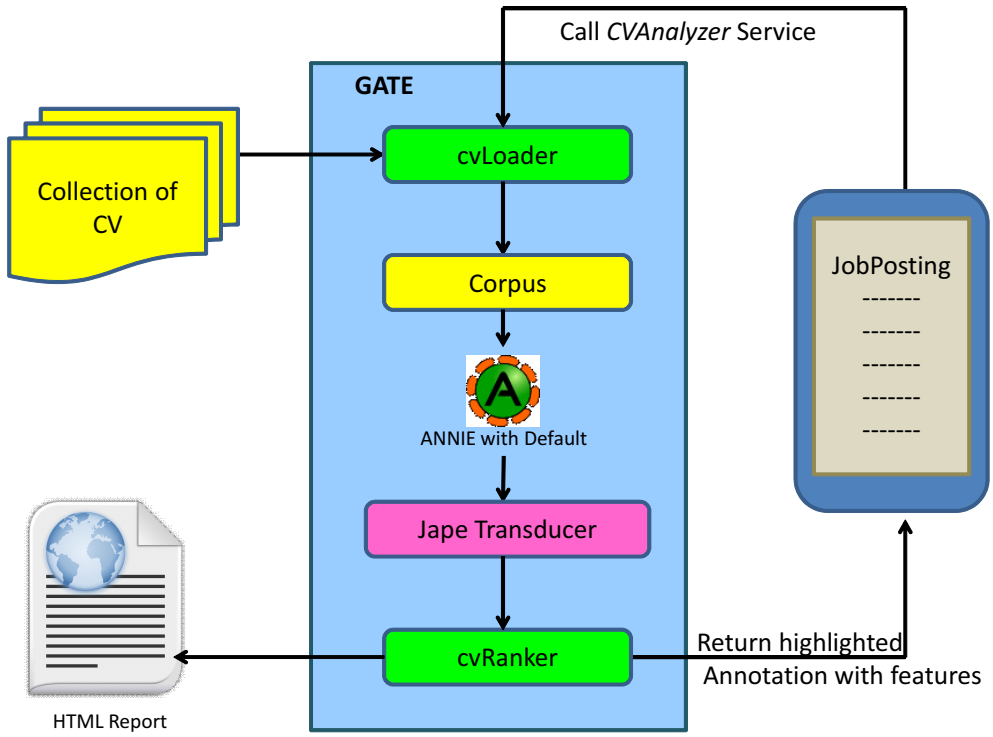

Fig. 1. *CVAnalyzer* with *Semantic Assistants*

GATE [3] to extract the key information of CV in French language. They extended GATE by adding some additional JAPE rules and Gazetteer lists to generate XML-Schemes by extracting information from the collection of CV. Karaa & Mhimdi studied the structure of 'EUROPASS CV' to identify some key terms of a CV such as 'personal information', 'work experience', 'technical skill', 'educational background' *etc*.and introduced an ontology which performed automatic annotations of the CVs [5]. Authors in [4] proposed a inferable model to determine the type of match between a CV and a job offer and then used similarity measure for that particular type of matching to rank the CVs with respect to the job post. Maniu and Ionela Maniu [6] developed an human resource ontology (HR-ontology) to improve the CV annotations. In this paper, we present *CVAnalyzer* which utilizes the *Semantic Assistants* framework to annotate *JobPosting* written in OpenOffice and CVs stored in server using the semantic software GATE [3].

III. Mechanism of *CVAnalyzer*

In this section, we present the mechanism of *CVAnalyzer* to find the relevant CVs from a collection of CV and rank them corresponding to a *JobPosting*.

CVAnalyzer is incorporated within the *Semantic Assistants* framework and used from the OpenOffice. GATE [3] is used to annotate the CV and *JobPosting*. The overall framework is presented in the Figure 1. The high level summary of the framework is as follows: *JobPosting* is written in an OpenOffice document and *CVAnalyzer* is selected as a *Semantic Assistants* service available in the OpenOffice having integration with *Semantic Assistants*. After the successful completion of the *CVAnalyzer* service, the *JobPosting* is annotated and contains the list of related and sorted CVs. The output of the *JobPosting* is shown in the Figure 2. A HTML report containing the sorted CV list and their key features is created in the user's home directory.

We now describe the components of the framework presented in the Figure 1.

A. *Semantic Assistant*

Semantic Assistants [8] is a framework that allow users to perform a number of natural Language Processing (NLP) services through some desktop or web applications. To establish connection between the end user and the NLP pipeline,

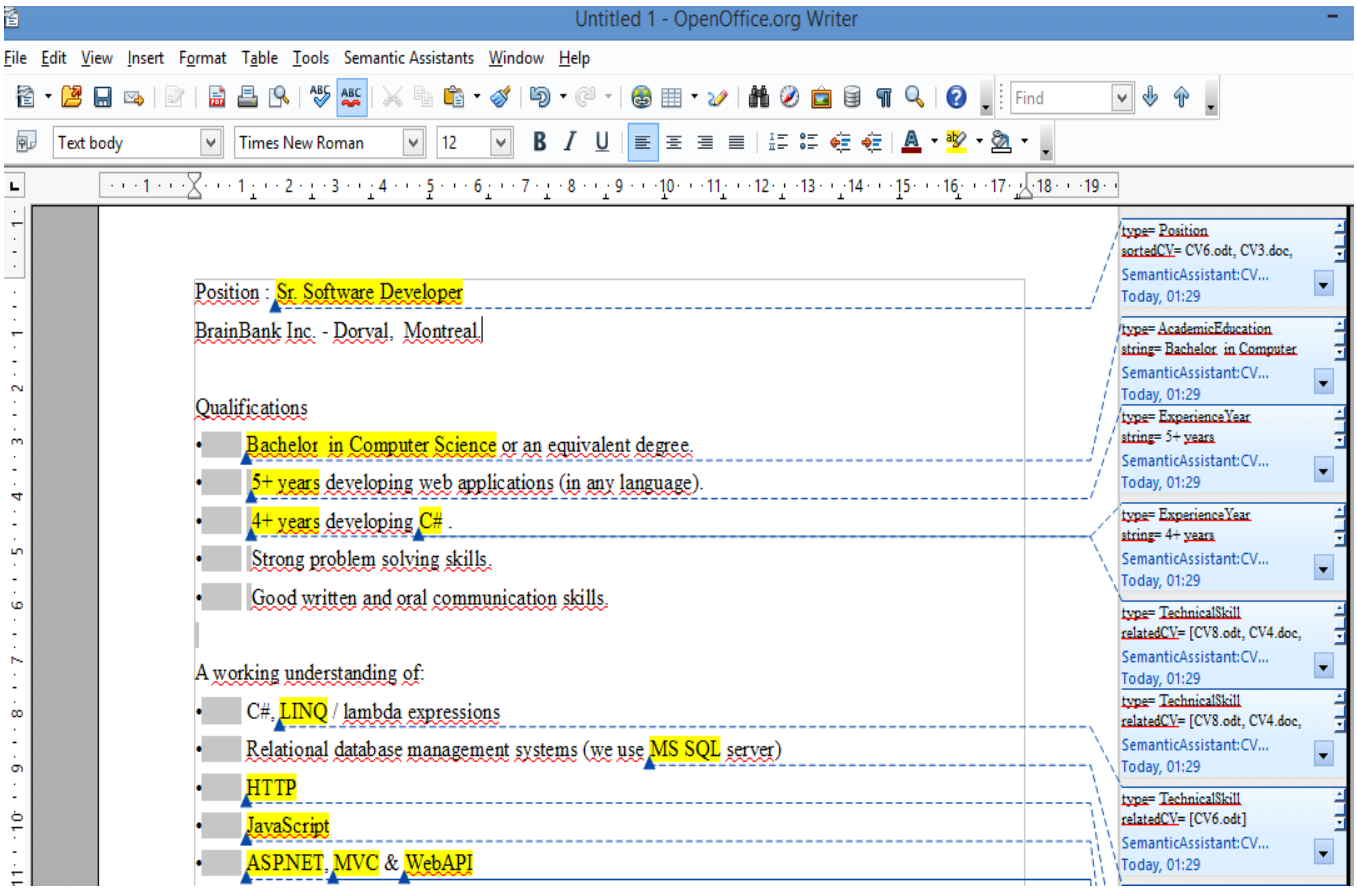

Fig. 2. *JobPosting* in OpenOffice with related and sorted CV list by *CVAnalyzer*

Semantic Assistants use OWL (Web Ontology Language) that describe the service, tasks and various artifacts. A java Plug-in has been developed for the OpenOfficece.org that enable users to access the available NLP service available in the *Semantic Assistants* framework. The plug-in creates a new menu *Semantic Assistants* with all the available NLP services. User can select any of the services and execute that.

A new NLP service named *CVAnalyzer* is created that enables the users not only to find the related CV when a job offer is posted but also rank those CVs based on some specific criteria. In this case, the client is Open Office document through which user have to post the job offer text. When *CV-Analyzer* service is called, a number of Processing Resources are executed over the *JobPosting* text and annotations with some highlighted texts are displayed to the user.

B. GATE

The proposed assistant *CVAnalyzer* is developed using GATE [3], which is a open source free software. GATE is a framework and development tool that enable user to perform various text processing task including information extraction, information retrieval, matching learning. GATE not only support multiple languages but also support input text from various format of file including doc, html, xml, pdf. It also have an evaluation tool that enable automatic performance measurement of the application developed.

We have used five components within GATE: *cvLoader*, Corpus, ANNIE, Jape Transducer, *cvRanker*.

C. Corpus

A collection of documents in GATE is called corpus. GATE processes all the documents in a selected corpus. We have used corpus for *JobPosting* and the collection of CVs.

Algorithm 1 *cvRanker* ($A(J)$, $\sum A(CV)$, ψ, δ)

Inputs:
  $A(J)$: Set of annotation from *JobPosting*
  $A(CV)$: Set of annotations for a CV
  ψ : Set of predefined criteria for matching CVs
  δ : Set of predefined category with score for ranking CVs
Helper Methods:
  $sort(\hbar)$: Sorts a HashMap containing CVs and their score in descending order
  $count(\oint_j(c_i), \oint_c v(c_i))$: Counts matching keywords in given summaries
  $printReport(\hbar)$: Prints a HTML report
  $isMatch(a_i, s_i)$: Returns true if the annotation's value is in the specified CV's summary
  $addFeature(a_i, \text{“}feature\text{”}, value)$: Add a new feature to the given annotations
Output:
  $A(J)$: Set of annotations in *JobPosting* with features including “relatedCV”, “rank”
begin
for each criteria c_i in ψ **do**
  $\gamma_j \leftarrow$ Find set of annotations with c_i from *JobPosting*
  $\oint_j \leftarrow$ Create summary of annotations having criteria c_i from *JobPosting*
  for each $A(cv_i)$ in $\sum A(CV)$ **do**
    $\sum \gamma_i \leftarrow$ Find set of annotations with c_i from CV annotations
    $\sum \oint_i \leftarrow$ Create summary of annotations having criteria c_i from CV annotations
  end for
end for
for each criteria c_i in γ_j **do**
  for each summary s_i in $\sum \oint_i(c_i)$ **do**
    for each annotation a_i in $\gamma_j(c_i)$ **do**
      if $isMatch(a_i, s_i)$ **then**
        $addFeature(a_i, \text{“}relatedCV\text{”}, s_i)$
      end if
    end for
  end for
end for
for each criteria c_i in γ_j **do**
  for each summary s_i in $\sum \oint_i(c_i)$ **do**
    update $\hbar \leftarrow (cvName, count(\oint_j(c_i), s_i) * \delta(c_i))$
  end for
end for
$sort(\hbar)$
$printReport(\hbar)$
for each criteria c_i in γ_j **do**
  for each annotation a_i in $\gamma_j(c_i)$ **do**
    $addFeature(a_i, \text{“}sortedCV\text{”}, \hbar(keyset))$
  end for
end for
return $A(J)$

D. cvLoader

A java PR (Processing Resource) named *cvLoader* has been created that collects all the CV Documents from the CV collection stored in the user's home directory. We put both the CV and *JobPosting* in the same corpus. We distinguish between the *JobPosting* and the CVs by appending prefix 'job_' and 'cv_' to the name of the *JobPosting* and the CVs, respectively.

E. ANNIE

ANNIE (A Nearly-New IE system) is the PR (Processing Resource) controller of GATE which executes all the processing resources and generates annotated documents in the selected corpus. ANNIE with default consists of the following processing resources:

- *Document Reset PR*
  This pipeline resets documents to the original state, by removing all the annotation sets and their contents.
- *Tokeniser*
  ANNIE English tokeniser splits the text into very simple tokens (numbers, punctuation and words of different type) by applying some basic rules to those input text.
- *Gazetteer*
  Gazetteer identifies named entity such as organization name, person name, location, date using lookup lists. Gazetteer list is a plain text file containing named entity of same types.
- *Sentence Splitter*
  This PR defines the start and end point of sentences.
- *Part of Speech Tagger*
  It produces a part-of-speech tag as an annotation on each word or symbol.
- *Name Entity Transducer*
  A number of JAPE rules have been used to create new annotations using the existing annotations.
- *ANNIE Orthomatcher*
  It performs co-reference resolution of named entity by identifying relations among different name entities.

When the corpus is provided to ANNIE, Tokeniser PR is applied both the job post and CV Documents text to generate tokens of different kinds such as number, word, punctuation etc. Then Sentence Splitter splits job post and job offer text into sentences. There are some jape rules in NE transducer that

```
Rule: Academic2
((
   (
    ({Token.string == "B"}|{Token.string == "M"})
     {Token.string == "."}{Token.string == "Sc"}
     ({Token.string == "."})?)|
    ({Token.string == "P"}
     {Token.string == "."}
     {Token.string == "hD"}
     ({Token.string == "."})?)|
    {Token.string == "Degree"}|
    {Token.string == "Bachelor"}|
    {Token.string == "Diploma"}|
    {Token.string == "BS/MS"}|
    {Token.string == "BS"}|
    {Token.string == "MS"}|
    {Token.string == "PhD"}|
    {Token.string == "Masters"}|
    {Token.string == "Master"}
   )
  ({Token.string == "in"} | {Token.string == "of"})
  (
    (
    {Token.kind == word, Token.category == NNP, Token.orth == upperInitial}|
    {Token.kind == word, Token.category == NNPS, Token.orth == upperInitial}
    )+
   (
    ({Token.string == "and"}|{Token.string == "&"})
    ({Token.kind == word, Token.category == NNP, Token.orth == upperInitial}|
    ({Token.kind == word, Token.category == NNP, Token.orth == upperInitial}|
     {Token.kind == word, Token.category == NNPS, Token.orth == upperInitial}
     )+
    )
   )?
  ):pro
):aca
-->
:pro.Decipline={kind="Program",rule="Academic2"},
:aca.AcademicEducation={kind = "AcademicEducation",rule = "Academic2"}
```

Fig. 3. JAPE rule for academic education

creates name entity such as person , their contact Information, location, Organization of the CV documents.

F. JAPE Transducers

Name entity recognition is one of the most important task for a NLP service. Although GATE has a rich collection of JAPE rules and Gazetteer list still these are not sufficient to extract all the key information of *JobPosting* and CVs. For this reason, we extend GATE by adding some additional jape rules and lists to create annotation of entities such as skill, Academic Education, number of experience year, Employers designation, Employers name, other skills for both *JobPosting* and CV documents.

1) Custom Gazetteer lists: The following two gazetteer lists are added for *CVAnalyzer*:

- *technical_skill.lst*

  This gazetteer list contains various technical skill names including programming language, operating system, server, database and other application name related in computer Science domain
- *Position.lst*

  This gazetteer list contains Employers' designation in computer Science related jobs

2) Custom JAPE rules: We have developed the following Jape rules:

- *academiceducation*
- *experienceYear*
- *technicalSkill*
- *jobPosition*
- *otherSkill*

A multiphase transducer has been used that contains all the customized jape rules developed for the NLP service and tested over a corpus which contains collection of CVs and *JobPosting* document. Figure 3 presents a jape rule 'academic

```
Phase: JobPosition
Input: Lookup Token
Options: control = appelt

Rule: position2
(
 (
    (
       ({Token.string == "Jr"}|{Token.string == "Sr"})
       ({Token.string == "."})?
    )|
   (
       {Token.string =~"[Jj]unior"}|
       {Token.string =~"[Ii]ntermediate"}|
       {Token.string =~"[Ss]enior"}
    )
 )?
 {Lookup.majorType == jobtitle1}
)
:pos1
-->
:pos1.Position = {Kind= "JobPosition", rule = "position2"}
```

Fig. 4. JAPE rule for job position

education' that creates annotations for Academic degree. This jape rule first checks the string that contains educational degree related information such as B.Sc., M.Sc, Diploma,Phd and followed by some Noun Phrase, then it creates an annotation of 'academic education' by highlighting those texts. Figure 4 shows the 'jobposition' Jape rule which checks some specific strings such as Senior, Junior which are optional and also check the Gazetteer list of positions to determine whether the text is present on that list or not. If such match is found then annotation of 'Job position' is created by this jape rule. In the similar way, other custom Jape rule works to create annotations.

We present the component *cvRanker* in the following section.

IV. *cvRanker*

This section presents the *cvRanker* component in *CVAnalyzer*. *cvRanker* is a crucial component in *CVAnalyzer* as it finds the matching CVs and rank them based on predefined criteria.

cvRanker is the final component of *CVAnalyzer* which is depicted in the Figure 1. *cvLoader* loads the CVs and *JobPosting* in the same corpus in GATE, which are then process by ANNIE, Jape transducer and *cvRanker*.

cvRanker takes the annotated CVs and *JobPosting* as input and updates annotations of *JobPosting* with new features including 'string', 'relatedCV', and 'sortedCV'. It also produces HTML report in user's home directory.

cvRanker involves the following steps:

- Initialize matching criteria and ranking mapping
  A set of criteria (ψ) is defined for annotation matching between *JobPosting* and the collection of CVs.For example,'programming language', 'Technical Skill', 'Education' *etc*.. These criteria are set based on the annotations used from the *JobPosting* and CVs. A mapping of criteria to score (δ) is defined to assign priority in found annotations in the CVs to rank them efficiently. For example, 'Education' has more score than the score for 'Location'.
- Categories annotations
  Annotations in *JobPosting* and CVs are categorized based on the matching criteria ψ. These categorizes annotations are used in the later steps for matching and ranking process.
- Generates summary of annotations
  A summary of the annotations ($\oint$) from the categories (in ψ) are created to match annotations in constant time and reduce the complexity of the algorithm.
- Computes related CVs

A new feature 'relatedCV' is added to the featureMap in the annotations of *JobPosting* . For example, an annotation a_i containing keyword 'Java' is available to a set of CVs cv_1, cv_5, cv_9. In this scenario, a_i will have new feature 'relatedCV:cv_1, cv_5, cv_9'.

- Ranks the CVs and add feature
  In this step, *cvRanker* ranks the CVs based the mapping specified in δ. The process of ranking has two steps: (a) it computes a score for each CV by counting the number of matching annotations in a CV for each categories in ψ with those in *JobPosting* and multiplying the count with the score of the categories (from δ), then summing into a total score; (b) Sort the CVs in descending order based on their total score. CV with a higher score has higher rank. A new feature 'sortedCV' is added to the annotations in *JobPosting*.
- Generates report
  In this final step, *cvRanker* produces a HTML report showing the rank order of the CVs corresponding to the *JobPosting*. This report also contains categorised annotations from each CV.

Algorithm 1 represents the pseudo code for *cvRanker*. In algorithm 1, lines 8-11 finds the set of annotations and creates summary from the CVs and *JobPosting* based on the given matching criteria. Lines 13-21 adds the new feature 'relatedCV' in the *JobPosting* annotations. Lines 22-26 computes the score of the CVs based on the ranking mappings. The collection of CV is sorted based on their score in line 27. A report is created in HTML format in the line 28. Finally, a new feature 'sortedCV' is added to the *JobPosting* annotation in the lines 29-33. As we are using HashMap and HashSet as our fundamental data structures, our algorithm is bounded by the complexity of $O(n^2)$.

V. Evaluation

The developed *CVAnalyzer* NLP service for OPenOffice focuses on IT jobs. It is possible to extend *CVAnalyzer* for other domains by extending jape rules and gazetteers. A collection of fifteen CVs candidate for IT jobs are prepared. In the OpenOffice document, *JobPosting*is placed and *CVAnalyzer* service is used to find the relevant and ranked CVs. We perform two types of evaluation- (a) Intrinsic evaluation, and (b) Extrinsic evaluation. Intrinsic evaluation uses some measurement tools such as precision, recall & f-measure and compare the system generated results with the gold standards. There are three different criteria to measure the performance such as Strict, Lenient and Average.

- Strict only count those annotations as correct that matched perfectly
- Lenient consider partially matching annotations as correct
- Average is the average score of Strict and lenient scores

Manual Annotation has been performed for both CVs and *JobPosting* to create the gold standards. The $AnnotationDiff$ tool [7] has been used to compare the system generated results with respect to those by the Manual Annotations. Figure 5 represents a part of the $AnnotationDiff$ tool where system generated annotations are compared with respect to the manual annotations. In this figure, 'Key' defines the 'gold standard' and 'Default set' represents the system generated annotations. The $AnnotationDiff$ tool compares one document at a time for one type of annotation. For example, the upper part of the figure presents the annotations of type 'TechnicalSkill' for *JobPosting*. The lower part of the figure shows the precision, recall and f-measure for these annotations. compares and generates .

The corpus created from the collection of CVs and *JobPosting* is evaluated according to the precision, recall and f-measure using the tool $CorpusQualityAssurance$ that produces statistics both for the corpus as a whole and for each document separately by comparing two annotation sets; Manual Annotation and system generated results. Figure 6 presents the performance of overall corpus containing all the CVs and *JobPosting*. The performance of *CVAnalyzer* is satisfactory because the average precision lies between 0.55 to 0.89 where the recall lies between 0.5 to 1.0 and f-measure lies between 0.67 to 0.93.

To perform the 'Extrinsic' evaluation, we first find out the related CVs and rank them for a specific *JobPosting* and calculate the time. In the next step, we calculate the time to produce the output by our proposed system. Finally, we compare the result with respect to the time of manual process.

We have estimated and compared the processing time in *CVAnalyzer* and the manual process. At first, we check our system performance for five CVs and calculate the processing time. In the next step, We manually read the *JobPosting* and those five CVs to find the related CVs and rank them. In this scenario, the manual process required 300 seconds whereas *CVAnalyzer* processed in four seconds. We continued the same

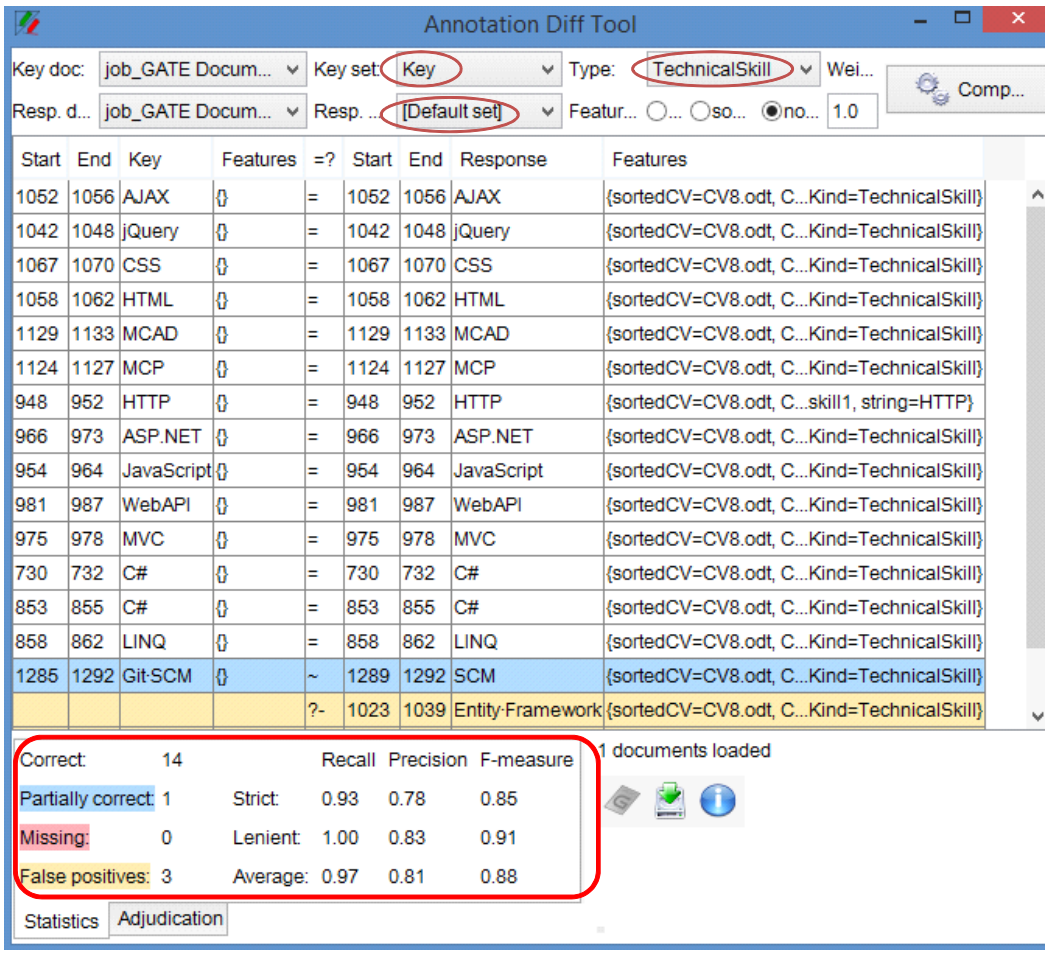

Fig. 5. *AnnotationDiff* tool showing the annotations and performance metrics

process for 10, 15 and 20 CVs and calculated the time. Figure 7 presents the response time for finding and ranking the related CVs for a specific *JobPosting* for the both cases. From the figure, it is revealed that the processing time for manual approach grows exponentially with the increase of the number of CVs in the collection whereas the processing time does not grow much in *CVAnalyzer*. From this result, we can claim that *CVAnalyzer* can play an important role the HR recruitment process by reducing the processing time substantially. As *CVAnalyzer* performs the annotation matching between a *JobPosting* and a CV, it will decrease the probability of ignoring right CVs for a *JobPosting*.

VI. Conclusion and Future Work

In this paper we have addressed the problem of finding and ranking CVs for a particular *JobPosting* in a timely manner. This paper presents a semantic tool *CVAnalyzer* to accelerate the HR recruitment process. It uses GATE [3] for CV and *JobPosting* annotations. *CVAnalyzer* is incorporated in the *Semantic Assistants* [1] framework to work with OpenOffice. It efficiently matches the CVs and ranks them based on the predefined categories. From the experimental results, it is revealed that *CVAnalyzer* requires much less time and effort to find related CVs and rank them with compared to the manual process. Although *CVAnalyzer* is not a complete automated process for HR recruitment, rather it helps HR people to find relevant CVs quickly and in an efficient manner based on predefined criteria.

In this work, we consider only IT-related jobs. We want to extend the proposed mechanism *CVAnalyzer* to ease the HR recruitment process in other fields.

References

[1] Semantic assistants, http://www.semanticsoftware.info/semantic-assistants.

Annotation Name	Matched Annotation	Found by Manual Annotation	Generated by System	Overlap	Strict			Lenient			Average		
					Precision	Recall	F-Measure	Precision	Recall	F-Measure	Precision	Recall	F-Measure
AcademicEducation	13	0	1	1	0.87	0.93	0.9	0.93	1	0.97	0.9	0.96	0.93
ExperienceYear	8	0	1	0	0.89	1	0.94	0.89	1	0.94	0.89	1	0.94
OtherSkill	7	7	1	2	0.7	0.44	0.54	0.9	0.56	0.69	0.8	0.5	0.62
Position	27	2	8	2	0.73	0.87	0.79	0.78	0.94	0.85	0.76	0.9	0.82
TechnicalSkill	184	21	146	37	0.5	0.76	0.6	0.6	0.91	0.73	0.55	0.84	0.67

Fig. 6. Different levels of performance for different annotations

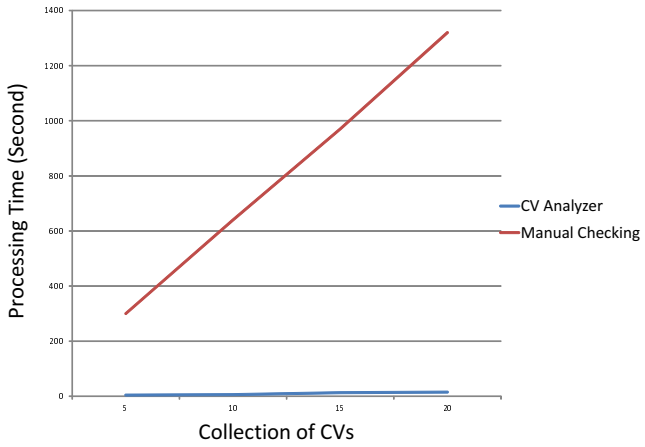

Fig. 7. Processing time for Manual approach and *CVAnalyzer*

[2] Soumaya Amdouni and Wahiba Ben abdessalem Karaa. Web-based recruiting. In *Computer Systems and Applications (AICCSA), 2010 IEEE/ACS International Conference on*, pages 1–7. IEEE, 2010.

[3] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. Gate: an architecture for development of robust hlt applications. In *Proceedings of the 40th annual meeting on association for*

computational linguistics, pages 168–175. Association for Computational Linguistics, 2002.

[4] Maryam Fazel-Zarandi and Mark S Fox. Semantic matchmaking for job recruitment: an ontology-based hybrid approach. In *Proceedings of the 3rd International Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web at the 8th International Semantic Web Conference, Washington DC, USA*, 2010.

[5] Wahiba Ben Abdessalem Karaa and Nouha Mhimdi. Using ontology for resume annotation. *International Journal of Metadata, Semantics and Ontologies*, 6(3):166–174, 2011.

[6] George Maniu and Ionela Maniu. A human resource ontology for recruitment process. *Review of General Management*, (2):12–18, 2009.

[7] Diana Maynard, Valentin Tablan, Cristian Ursu, Hamish Cunningham, and Yorick Wilks. Named entity recognition from diverse text types. In *Recent Advances in Natural Language Processing 2001 Conference*, pages 257–274, 2001.

[8] René Witte and Thomas Gitzinger. Semantic assistants–user-centric natural language processing services for desktop clients. In *The Semantic Web*, pages 360–374. Springer, 2008.

[9] Leila Yahiaoui, Zizette Boufaïda, and Yannick Prié. Semantic annotation of documents applied to e-recruitment. In *SWAP*, 2006.