YPA — an intelligent directory enquiry assistant

A De Roeck, U Kruschwitz, P Neal, P Scott, S Steel, R Turner and N Webb

The YPA project is building a system to make the information in classified directories more accessible. BT's Yellow Pages®[1] provides an example of a classified database with which this work would be useful.

There are two reasons for doing this: (i) directories like Yellow Pages contain much useful but hard-to-access information, especially in the free text in semi-display advertisements; (ii) more generally, the project is a demonstrator for exploitation of semi-structured data — data that is less systematic than database entries or logical clauses, but more systematic than free text because it has been marked up, for display or some other purpose.

Accessing the directory source data file requires both natural language processing (for softening the interface to the system, and separately for analysis of natural-language-like constructs in the data) and information retrieval techniques, which are assisted by shallow knowledge. Deep world knowledge is impractical.

The project seeks to get maximum effect from conveniently simplified approximations of standard natural language processing and knowledge representation. The paper gives an overview of the system, and illustrates its style with points about how the source data file is analysed. The YPA requires further development, but already demonstrates the effectiveness of shallow processing applied to semi-structured data.

1. Introduction

The YPA is a language engineering project for building an intelligent directory enquiry assistant where various techniques and resources are combined in a realistic application. It uses well-established natural language processing (NLP) and information retrieval (IR) techniques; its contribution lies in the way they are put together and the software engineering problems encountered while doing this. The major novelty resides in the way these techniques are applied to semi-structured data, using a sophisticated combination of NLP and IR after an extensive preprocessing of the given data sources. The YPA requires further development, but this paper describes the current working prototype which already demonstrates the effectiveness of shallow processing applied to semi-structured data.

Accessing classified directories via the YPA should allow a user to conduct a natural language dialogue in order to retrieve addresses and telephone numbers, but compared to the paper version of the directory there are some important differences. Rather than looking up a specific category or company name, the YPA should make extensive use of knowledge not contained in the category information, as in the following example:

I need a plumber with an emergency service.

In this case, the conventional look-up of addresses seems to be quite straightforward, but will probably yield a long list of plumbers which will have to be checked individually to find whether they offer an emergency service. But now consider:

Where can I buy teaspoons?

If there is no mention of teaspoons in the local classified directory, a look-up fails. Most commercial directory enquiry systems cannot deliver satisfactory results in these cases. This is the sort of information one does not necessarily get from the classification but which is usually hidden in the free text of the advertisements. Worse, it may not even exist explicitly in the directory source data. However, the YPA can find cutlery if it has some world knowledge — employing synonymy and cross-reference, it maps unknown input (teaspoons) to possibly relevant terms occurring in the classified directory (cutlery).

[1] Yellow Pages® and Talking Pages® are registered trade marks of British Telecommunications plc in the United Kingdom.

The classified directory source data file is not a purely relational address database. It contains additional information which suggests a more information retrieval based approach for accessing the addresses. This additional information comes, for example, as so-called free text with various address entries (e.g. 24-hour emergency service).

From the engineering point of view, the off-line construction of the back-end database (extracting as much information as possible) seems to be a crucial point. After all, the user acceptance of the on-line system largely depends on the sort and structure of data that can be employed in the dialogue.

The remainder of this paper is structured as follows. Firstly, the background of the project is briefly sketched and an overview is given of the YPA system, highlighting some of its components (sections 2 and 3). After that the structure of the source data file to be exploited is examined in section 4. Section 5 explains the automatic off-line construction of the back-end in some detail, and section 6 focuses on recent experiences and integration. Finally there is an outlook on future work.

2. Aims and related work

In order to have a base for evaluation the aims of the YPA have to be defined. The final system is most importantly characterised by:

- the functionality of Talking Pages (where a user can call to find out addresses listed in Yellow Pages),
- coverage of all the Yellow Pages classified data,
- a user-friendly dialogue.

Based on these aims, it can be assumed that a user knows what the classified directories are and that the user is co-operative and truly interested in a set of addresses. Also a scenario is assumed where the YPA works as an on-line system to be accessed via the Web, though this should be a flexible aim. Various systems address the same issues, but these tend to be more concerned with either front-end or back-end, or less concerned with a particular application.

Most natural language dialogue systems are concerned with the problem of schedules or time [1—5], something that does not apply to the YPA. More closely related to the YPA is the Voyager dialogue system [6—8] which deals with addresses from NYNEX Yellow Pages. However, as noted above, this project uses source data that is more than a pure address database. Hence it seems there are IR systems that are even more closely related. This paper uses concepts related to those in Strzalkowski [9] and Strzalkowski et al [10]. On the other hand, it did not make sense only to follow the IR approach for this project since these concepts are highly statistical and require large quantities of data, and the local classified directories are normally small in size. Furthermore, flat IR would overlook much of the information and structure it does have.

A number of on-line directory enquiry systems are actually in use, e.g. Freepages (UK) [2], NYNEX Yellow Pages (USA) [3], BigBook (USA) [4] and Switchboard (USA) [5].

These types of commercial system function quite differently from the previously mentioned NLP or IR systems. While they access large address databases of the same sort as those with which this project is dealing, they all share a number of limitations it is hoped to address with the development of the YPA:

- a very flat front-end which at most offers pattern matching (e.g. Freepages does not even allow the use of plurals),
- a very simple look-up database which does not permit the access of free text hidden in the addresses,
- the inability to cope with words not found in the database (i.e. in the categories or names).

Therefore the YPA has a heterogeneous structure which employs NLP as well as IR ideas. Naturally this is important in the on-line dialogue, but it also affects the modular design of the system and the off-line construction of the back-end.

3. System overview

The YPA is an interactive system. Figure 1 is an overview of the system architecture (depicting the data flow).

A conversation cycle with the YPA can be roughly described as follows. A user utterance (recognized by the speech recognition or typed in via the graphical user interface) is sent to the Dialogue Manager. The Dialogue Manager keeps track of the current stage in the dialogue and controls the use of several sub-modules. Before handing back control (together with the relevant data) to the Toplevel, the input is first sent to the parser which returns a so-called slot-and-filler query. The Dialogue Manager then consults the query construction component, passing to it the result of the parsing process (possibly modified depending on the dialogue history, etc). The purpose of the query construction component is to transform the input into a database query (making use of the back-end and possibly the world model), to query the back-end and to return the retrieved addresses (and some database information) to the Dialogue Manager. Finally the Dialogue Manager hands

[2] http://www.freepages.co.uk
[3] http://www.bigyellow.com
[4] http://www.bigbook.com
[5] http://www.switchboard.com

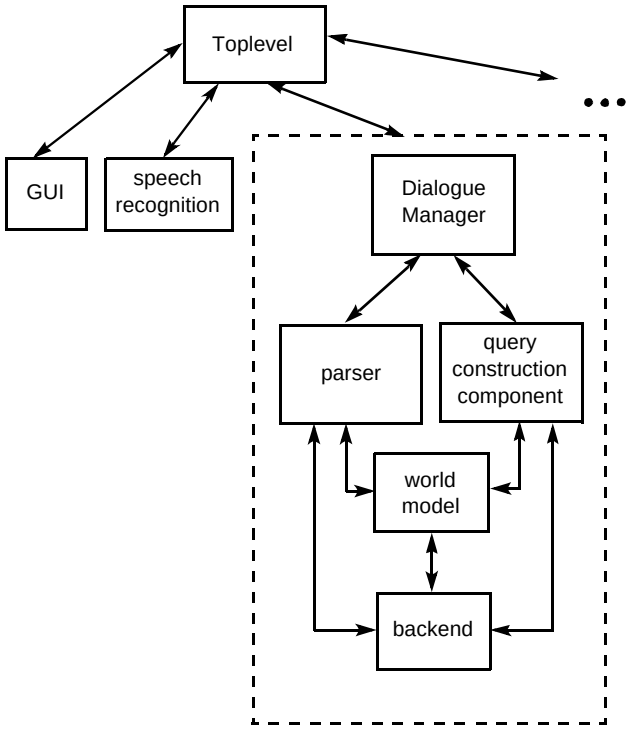


Fig 1 Architecture of the YPA.

back control to the Toplevel which, for example, displays the retrieved addresses. It could also put questions to the user which were passed to it by the Dialogue Manager, if the database access had not been successful (i.e. did not result in a set of addresses). At this stage the cycle starts again.

While the main focus is the back-end, a sketch of other modules will be given in the next sections in order to give a context.

3.1 The Dialogue Manager

The Dialogue Manager plays an important role by controlling the access to the various modules. The overall architecture of the Dialogue Manager is very similar to the PURE system (see Agarwal [11]).

The user input is passed to the Dialogue Manager, which calls the parser and, depending on the result, decides whether, for instance, the database should be accessed (by calling the query construction component) or whether the dialogue should be restarted.

The interaction between user and machine for processing one query can be roughly summarised as a filling of different slots in a slot-and-filler query with the information extracted from the user input, so that the database access finally retrieves an acceptable set of entries. In this process, the Dialogue Manager is the vital control module. Each stage in the dialogue with the user is characterised by a dialogue state, starting with the initial state. Every time the Dialogue Manager is called, it performs the following tasks:

- calling the parser,
- evaluating the parsed input and determining the new state of the dialogue, i.e. the new dialogue history,
- performing all actions corresponding to the state transition.

The actions performed might include calling the query construction component and passing the current slot-and-filler query as an argument.

The Dialogue Manager consists of a core Dialogue Manager which is the domain-independent heart of the system and an extension (also domain-independent) which adds the basic functionality of a Dialogue Manager to a frame-based approach and which is called the default Dialogue Manager. The administrator has to:

- set interfaces to the core Dialogue Manager and the default Dialogue Manager,
- customise the system for the specific application.

The general idea about a domain-independent basic Dialogue Manager is to have a core Dialogue Manager that covers all tasks to be handled by any similar dialogue system without having to access the database system. It should detect:

- that a user wants to quit (or restart) the dialogue,
- meta queries (where the user asks for some help, etc),
- that a user uttered some correction.

The default Dialogue Manager is this core engine expanded by adding coverage of the other states that can occur in a general spoken dialogue system:

- mandatory slots (which must be filled in order to submit a query to the database system) are not filled,
- unknown concepts occurred in the input,
- inconsistency occurred,
- a database access was successful,

- a database access results in too many matches,
- a database access results in too few matches.

This outline compares to the two-layered dialogue architecture in Agarwal [11], where the default Dialogue Manager covers the upper layer of dialogue states, and where customisation may refine those and add a second (domain-dependent) layer. Differences, however, are the set of dialogue states and the distinction made between the various possible states.

In order to keep the Dialogue Manager truly parametric, nothing is assumed about the structure of the query except that it is some sort of slot-and-filler list. In the YPA, the current Dialogue Manager is customised by defining the appropriate interfaces in the set-up files.

The YPA is being built in from both ends (front-end and back-end), and therefore changes to the Dialogue Manager are expected.

3.2 The parser

If the input is typed, it is not wise either to expect the user query to be composed of well-formed sentences or on the other hand to insist that the user must fill in given slots. The YPA expects arbitrary user input which is parsed by a relatively flat but robust parser, an adaption of the SNAP parser [12]. The main task is to detect syntactic information, leaving semantic processing to the query construction component. Later in the processing, an application-specific slot-filling process takes place (comparable to the functionality of the pragmatics component in Agarwal [11]).

The simple example in Fig 2 illustrates the process of parsing the user input into a slot-and-filler structure.

The set of slots depends on the domain. In this application, the three slots seem to be sufficient. The default slot is the goods-slot (which stands for 'goods and services'). Phrases whose function is doubtful, e.g. prepositional phrases like 'with an emergency service' or 'as a birthday present', will be put into that slot (Fig 3). The query construction component determines whether this information is relevant for the query itself or just for the ranking process that calculates a relevance value for each retrieved address before the addresses are displayed.

3.3 The query construction component

This component is the most important to the quality of the results of a user's query. The structure passed to the query construction component is a slot-and-filler query. The task is to match this query to a set of addresses by consulting different sources of knowledge, namely the transformed source data (part of the back-end) and knowledge sources which can be summarised as the world model. While the back-end supplies indices as well as ranking values, the shallow world model delivers information which can be employed on the back-end (e.g. for query extension). It is therefore the task of the query construction component to evaluate the various information sources (e.g. indices versus ranking values) and retrieve a set of addresses from the back-end if possible. A sample transformation (again simplified) might look like that shown in Fig 4.

The constructed query is sent to the address database. If this results in a set of addresses (up to a maximum number defined by the administrator in the set-up), then the query construction is finished. If too many addresses are retrieved, then the query will be further constrained. For instance, detected prepositional phrases in the slot-and-filler list might be added as conjuncts and a new query would be constructed. A query that resulted in no matching addresses at all could be relaxed (if possible). This involves exploiting the world model, e.g. checking for synonyms or cross-references in the directory heading structure.

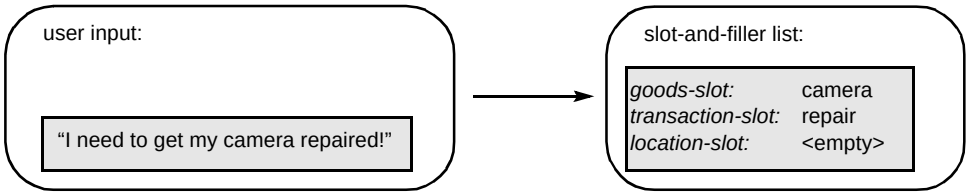


Fig 2 Parsing the user input.

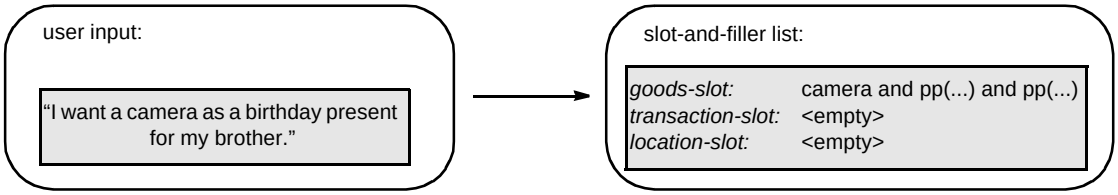


Fig 3 Dealing with doubtful phrases.

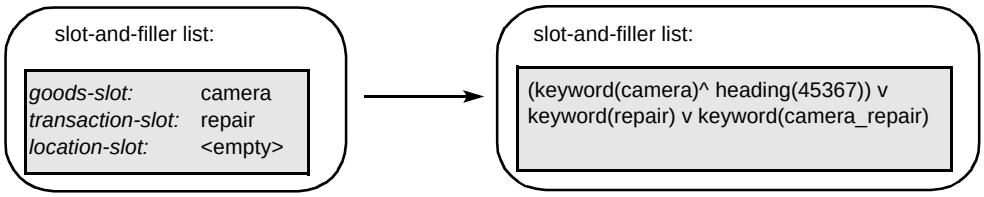


Fig 4 A sample transformation (as illustrated the query contains indices which come in various forms like ID numbers).

3.4 The world model

The world model is not a homogeneous component. It currently has these main parts:

- a large lexicon containing various simple hierarchies (WordNet [13]),
- the heading structure from the directory (see section 5.1 for the construction process),
- knowledge acquired by the user (updated via the YPA AdminTool).

3.5 The back-end database

The back-end contains three quite different parts:

- the (relational) database that contains tables extracted from the source data file, and other relations (e.g. matching locations to dialling codes),
- a database containing meta data for these indices (such as ranking values for the appropriate indices, relevant to the query construction component),
- the language module (e.g. for reduction of words to base forms) — this will be ignored when looking at the back-end construction since this part does not have to be constructed for each set of data.

4. The semi-structured input file

Before looking at the process of back-end construction, the structure of the given sources [6] needs to be examined.

All addresses, headings and various references are stored in a record structure, where each line is one record. However, there is no 1:1 relation between records and entries — address entries, heading entries, etc. More specifically, most addresses only stretch over one line (so-called free entries), but some addresses of a different type (e.g. semi-display entries as shown in Fig 5) will always consist of more than one line. An interesting part of these address entries is the free text which is an optional natural language portion to be printed in the advertisement along with the address as in the example in Fig 5.

[6] The source file used is the Colchester 1997 edition of Yellow Pages.

Kruschwitz Golf & Leisure Wear
Suppliers Of All Top Brand Golf Equipment
100 High Street Colchester 822990

Fig 5 Simple advertisement with free text.

There is no special attribute that marks a record as free text (for example, in Fig 5 'Suppliers Of All ....'). Thus, the source data file can be considered as a semi-structured text because:

- the record structure means that there is no unrestricted text,
- the record structure is relatively poor, so that while the extraction of an entry is straightforward, it is not obvious how to split its address (for instance into units).

After analysing the given file, the initial task of exploiting the given structure can be summarised in the following steps:

- layout analysis, which transforms the input file into a canonical form (as in phase (1) in the indexing process of Callan et al [14]),
- address extraction (of any address entry type),
- extraction of headings (of various types, as will be explained later),
- splitting address entries into smaller units (parts of addresses),
- conversion into relational schemata.

Performing these steps finally results in a relational database which is one part of the back-end.

The second back-end creation phase is characterised by transforming the information from the source data file into an information retrieval component:

- lexical analysis (removing stop-words, indexing words, etc, as in phase (2) in Callan et al [14]),
- calculation of ranking tables.

These two steps are applied to different parts of the extracted information. The lexical analysis of the headings entries is not the same as for the company names or parts of the addresses.

5. The back-end construction

Figure 6 reflects the data flow in the process of constructing the databases that form the back-end of the YPA.

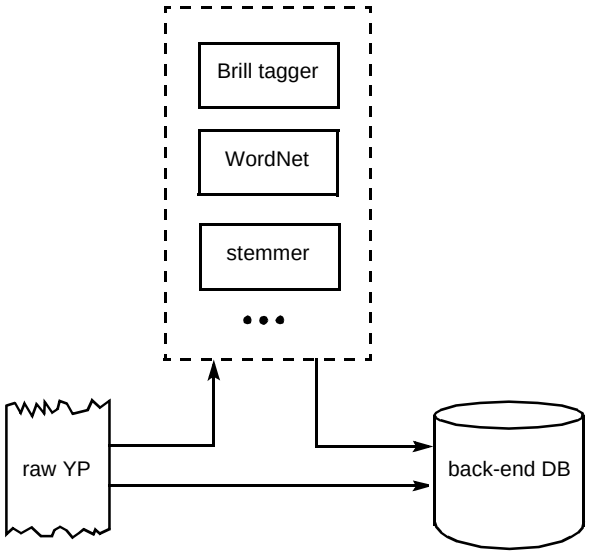


Fig 6 Extraction of the YPA back-end.

The data extraction and transformation takes place in several steps. It seems more useful to describe the processes involved according to the different sorts of input information, rather than the order in which the back-end is actually created.

It will be seen that the same extraction techniques will result in significant differences in the results when applied to different parts of the input.

However, it will be assumed some conditions hold in all further processing.

Firstly, for part-of-speech tagging the Brill tagger [15, 16] is used without training (using the supplied lexical rule and contextual rule files of the Wall Street Journal Corpus and the lexicon of both the Wall Street Journal and Brown Corpus). This tagger is particularly appropriate as a contextual tagger is needed, especially one that is robust enough (see also section 5.2). Furthermore the tagging follows the Penn Treebank guidelines [17] which makes the results comparable.

Secondly, WordNet [13] is used for indexing the keywords. To do this, the WordNet interface that performs morphological reduction to base forms is used. Then a stemmer is applied to further reduce the base forms delivered by WordNet. The result of the stemming does not have to be a proper lexicon entry (cf Strzalkowski [9]). However, the synonyms as provided by WordNet can still be made use of.

Thirdly, the ID for each entry in the data is the unique line number where this entry starts. Hence, there is automatically a key for most of the relational tables to be created.

5.1 Extraction of headings

Extracting the headings from the source data file is straightforward because one part of the record structure defines whether a record is a heading. Each heading entry occupies exactly one record (i.e. line). There are different types of heading (see below), but the differences do not affect the extraction and indexing process.

The content of the headings is extremely rich in information especially because there are very few cases where a heading consists of more than four words. This is also the reason why there has to be some preprocessing before tagging and indexing.

- Changing the word order

  Headings are often indexed on the main concept (e.g. concrete — ready-mixed). These cases can be detected by pattern matching and can be switched (to ready-mixed concrete). There are indeed cases where two swaps have to be performed.

- Replacing abbreviations

  The distribution of unknown words in the heading entries shows that in this case there were 140 unknown words in 3037 headings using the WordNet lexicon. Only five of them occur more than three times. Most of the rest of the words would be accepted by a different lexicon. The three most frequent unknown words in the headings are abbreviations that can be replaced before tagging — mfrs (269 times), eqpt (190 times), wh'salers (48 times).

Finally there are three general rules for selecting the indices. An index is created for:

- any single word,
- any compound consisting of two or three consecutive words,
- noun compounds consisting of more than two words — such compounds are indexed on any single word in the compound together with the last one.

All indices are written in a canonical form (see also section 5.2).

As noted, there are different types of headings. More specifically there are headings and heading references. The directories are divided into sections which each list a set of addresses. The name of such a section is a heading. Moreover, there are two types of heading references that can immediately follow such a heading. Firstly, 'see' references occur in sections that do not contain addresses at all but instead only refer to other headings (e.g. heading fishing agents contains only a reference to shooting and fishing agents). Secondly, there is a type of heading reference which is called a 'see-also' reference. This type can be found in sections which do contain addresses but where a reference might be a useful addition to the heading (e.g. heading zoos contains a 'see-also' reference to tourist attractions).

Technically, the transformation of a heading reference (which contains only textual information rather than a link into other parts of the input file) into a proper relation between entries has to be performed, but this can easily be automated. The interesting aspect of processing is deciding how to evaluate a keyword index that is detected for such references — how valuable is a keyword that occurs in a heading which does not contain addresses but a 'see' reference, etc? This will be discussed in more detail in section 5.3.

5.2 Address extraction

This process involves the detection of patterns for parts of addresses, the selection of free text, as well as the indexing process and further evaluation of the results.

Extracting parts of addresses

The problem of semi-structured text is that a selected address does not automatically tell us which portion of it denotes which 'concept'. Determining the location of a business is not straightforward. World knowledge has to be applied which at least contains all place names in the area; but this is not enough since the format of the addresses does not follow a convention.

However, patterns exist that allow the automatic extraction of all company names and many of the telephone numbers. (Since the postcode is a quite distinct pattern, it is extracted as well, but it actually appears in far less than 10% of all advertisements.) Experiments on the input file have shown that nearly 100% of addresses contain a pattern that is recognised as a telephone number. (The exceptions are a few advertisements that say only 'See our main ad under ...' and similar cases.) Most of those patterns can be split into a dialling code part and the telephone number. Detecting a telephone number is easier than splitting it into parts, but more than 90% of the detected lines that contain a telephone number can be split into proper parts by looking for very few possible patterns. The dialling code part is either the dialling code of the area or a place name whose dialling code can be found from the directory itself — 74 locations for the Colchester area. Thus for most addresses, the corresponding dialling code is extracted with minimal world knowledge.

Determining free text

The first difficulty in indexing the free text of the entries is to determine the appropriate lines. Some clues are given by the record structure (e.g. there is an indicator for a line that contains the company name which tells us that this is definitely not free text), but there is not much more.

Therefore the first task was finding the best extraction rules. This was done in a cycle. Each loop in this cycle was determined by deleting lines from the address corpus according to some heuristic and then checking the resulting output by the following criteria:

- deleting too many of the actual free text lines means not capturing certain addresses when looking for keywords and worsening the overall keyword ranking — especially in the source data, many possible candidates for indexing occur just once in the free text and might be missed totally,
- deleting too few lines from the address input means that parts of the addresses which are not free text lines are considered to be free text — this affects the ranking of the indices.

The results of each cycle were evaluated by automatically applying Unix tools together with a lexicon [13] to the extracted free text. This provided figures for some of the interesting phases of this extraction process, namely:

- the number of selected lines,
- the distribution of keywords (as well as compounds),
- the detection of certain typographical patterns,
- the distribution of words unknown to the lexicon.

The heuristics now applied mainly delete lines that contain telephone patterns or indicators for addresses as well as company names.

Indexing the selected text

Again a difficult situation arises. On the one hand, the text lines that are the base for free text indices are finally available, but on the other hand, this data does not conform to principles usually assumed in information retrieval or information extraction. (This is true for both indexing the free text and the extracted company names, which are

handled similarly.) The main problems can be summarised like this:

- there is no sentence structure, just a set of phrases,
- upper/lower case distinctions are often used for lexical analysis (for example, Callan et al [14] and Brill [16]), but in the source data file it is irrelevant — usually the complete address entry is in one case,
- additional tagging errors (e.g. tagging a noun as a verb) result from the other points.

One solution to the problems is to concentrate on nouns and compounds. It should be recalled that headings contain only phrases, mainly noun phrases, that are very compact. Even adjectives in a heading normally carry important information. In free text that is not so.

In the free text, all words that were not tagged as either a noun, a verb, or an adjective are deleted. For the resulting list of words, an index is written for:

- any single word tagged as a noun,
- any compound consisting of two or three consecutive words,
- noun compounds consisting of more than two words are indexed on any single word in the compound together with the last one (based on the assumption that usually modifiers modify the last item, though this is not always true).

Indexing keyword phrases of more than two words is common in information retrieval tasks (e.g. Strzalkowski [9]). But since the dialogue with the YPA is a conversation, relatively short and uncomplicated phrases can be assumed. Initially, it was thought it would make little sense to use three-word indices in this context, as a longer compound can still be reduced to the selection of parts of it. It was soon realized, however, that compounds like ‘equipment repair’ and ‘horticultural nurseries’ could relate to far too many addresses and therefore compound indices of up to length three are now constructed.

The indexing table is reduced to less than half its size by deleting all compound indices that are detected only once in the corpus. This makes the number of compound indices relatively small. The reason for deleting only those single occurrences is that a relatively small-scale ‘corpus’ is being used. By contrast, the INQUERY system eliminates concepts that occur less than 16 or more than 3000 times [18]. In the context of Yellow Pages, the source data, every single word (especially in the headings) may be significant.

The indices are written in a canonical form (i.e. alphabetical order). Any attempt to extract the head and modifier of a compound was abandoned, since the overall increase in recall, even for a purely alphabetically ordered index, is relatively small. Also Strzalkowski et al [10] report for their IR system dealing with much larger corpora:

> ‘This head+modifier normalisation has been used in our system... At the same time, while the gain in recall and precision has not been negligible, no dramatic breakthrough has occurred either’.

5.3 Definition of ranking values

The calculation of weights to be attached to the indices is very much project-specific.

A standard IR scheme for the assignment of weights to indices is the ‘term frequency—inverse document frequency’ formula (*tf.idf*):

$$tfidf_{ik} = \#(\text{term } i \text{ in document } k) * \log\left(\frac{\#(\text{documents})}{\#(\text{documents with term } i)}\right)$$

which assigns term i in document k the value $tfidf_{ik}$.

On the one hand, the indices are available and distribution, etc, can be calculated. On the other hand, it is not possible to apply a formula like *tf.idf* in order to get a value for each keyword reflecting its importance, since these documents are single entries, and normally each index occurs at most once in the address. Even if there are ten lines of free text this is still a very small document and moreover, ten lines of free text would be unusually long.

Different indices have different effects on the overall ranking. The importance of an index — its overall ranking value — is determined by the number of addresses that would be retrieved only using this index as the access key. Moreover, this relation can be applied to smaller domains, on the keywords in the headings, in the free text of addresses, and in the company names.

The two types of heading references that exist represent a special case. A keyword index in a heading which contains a ‘see’ reference instead of addresses will have the same ranking as it would have if all the addresses under the referenced headings were stored under the heading that contains the reference. The heading ‘yoga’ contains ‘see’ references to ‘health clubs and fitness centres’ and ‘sports clubs and associations’, but lists no addresses. If this principle were not applied, the word yoga would not have a ranking value as a keyword index.

‘See-also’ references are currently ignored for the calculation of ranking values. The reasons for that are:

- ‘see-also’ references are usually much more general than the heading under which they occur,
- when a ‘see-also’ reference turns up, it usually makes reference to many headings.

However, even though these reference types do not influence the ranking values, they are exploited in the dialogue (e.g. if no addresses can be retrieved otherwise).

6. Recent experiences and integration

This paper has discussed how the data from the source file can be transformed into a database. This database forms the backbone of the YPA. There is a sub-module of the query construction component, the address ranking component, which takes the initial query and the list of retrieved addresses as input and orders them according to relevance by giving each address a percentage of relevance for the given query. The ranking function implemented in the current system calculates an accumulative relevance value based on the following assumptions:

- heading indices are most important,
- indices in free text and in names get a lower initial value,
- indices that exist for knowledge retrieved from the world model are least significant, e.g. assume a user query: 'I need my hedges trimmed,' an address selected because it contained the keyword hedgerow as a synonym for hedge is assumed not to be as significant as an address that has an index on hedge,
- all indices are weighted by their frequency in the complete database.

The weights for each index type are defined heuristically and will have to be re-investigated during evaluation. However, because there is a separate address ranking component, other aspects can be added to the ranking function. It is possible to give semi-display advertisements a higher initial weight than free entries, because they are more expensive. Other factors might influence the address ranking, e.g. if it is undesirable to bias towards longer entries, the ranking function will weight each index by the length of the entry.

Further recent developments have involved the extraction of more address information from the data. As previously mentioned, the dialling code (i.e. the location) could be extracted easily. Now the rest of the entry (everything excluding name, free text, postcode, telephone pattern) can be split into street name, house number and rest of address, without having to apply additional knowledge. This information is useful in a query such as:

> 'Which restaurants are there in Colchester High Street?'

This additional information will be used only if there are otherwise too many addresses, and if the location slot had been filled already. It would not make much sense to display addresses for restaurants in High Street if it was not known that this is in Colchester. In any case, this information is used in the calculation of the address relevance.

The off-line processing of the indices, rankings, etc, can be performed by Unix tools (mainly Gawk), some C-programs together with the necessary libraries for the lexicon and the tagger scripts.

A new edition of the classified directory or directory for a different area can be handled automatically.

Figure 7 shows a simple query in the on-line dialogue with the YPA.

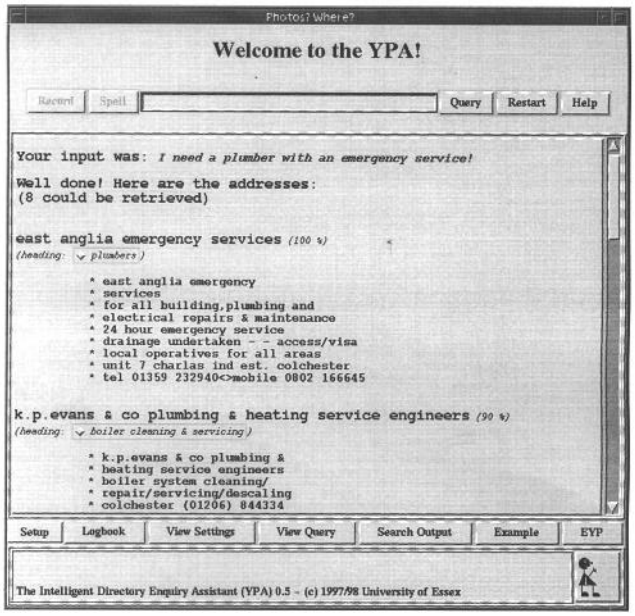


Fig 7 Screenshot of an on-line dialogue.

The current version YPA0.5 runs on a SPARC station 5-175 with selected domains as well as with the complete Colchester Yellow Pages (about 26 000 address entries). It is a Sicstus Prolog Executable controlled by a Tcl-Tk graphical user interface. As an alternative, a Web interface was also developed which is realised by a set of Perl scripts and accesses the YPA via socket connections.

The experimental speech recognition software which can optionally be used is constructed with the Nuance speech recognition tool-kit of Nuance Communications. Speech recognition is not a focus of this project, but it has been shown that the integration of speech input is possible.

7. Future work

Experiences to date with the prototype of the YPA are encouraging. For this project, it seems to be the only way to combine IR techniques with reasonable NLP. Relying entirely on IR would miss out too much relevant information, while a more powerful NLP approach would be inappropriate in a domain where both the data and the queries are so stylised.

Naturally a system with these goals must be evaluated. The system is not in its final form, with an extended query corpus being considered. The evaluation will start with the selected domains from the Colchester Yellow Pages, and the results will be published shortly.

It is also intended to challenge the current assumptions by testing how well the system extends to new examples of the domain — other areas and new editions of the directory.

The back-end is expected to show itself to be stable, but the key interest is in finding how the Dialogue Manager and the query construction component will need to be modified.

Another part of the future work will be the improvement of the YPA AdminTool, a tool (not described here) for the administrator not the user of the YPA. Encountering unknown terms may cause the Dialogue Manager to enter a simple clarification dialogue with the user. Knowledge so acquired is logged for the administrator who may choose to add it to the (world) knowledge base by using the YPA AdminTool.

Acknowledgements

This work has been funded by a contract from the ISR group at BT Laboratories, Martlesham Heath. The authors want to thank the reviewers as well as K C Tsui, Wayne Wobcke and Nader Azarmi for their helpful comments on this paper.

References

1 Aust H, Oerder M, Seide F and Steinbiss V: 'The Philips automatic train timetable information system', Speech Communication, 17, pp 249—262 (1995).

2 Wahlster W: 'Verbmobil: translation of face-to-face dialogues', Proceedings of the 3rd European Conference on Speech Communication and Technology, pp 29—38, Berlin, Germany (1993).

3 Sikorski T and Allen J F: 'A task-based evaluation of the TRAINS-95 dialogue system', Proceedings of the Workshop on Dialog Processing in Spoken Language Systems, ECAI-96, Budapest (1996).

4 McGlashan S, Fraser N, Gilbert N, Bilange E, Heisterkamp P and Youd N: 'Dialogue management for telephone information systems', Proceedings of the International Conference on Applied Language Processing, Trento, Italy (1992).

5 Heisterkamp P, McGlashan S and Youd N: 'Dialogue semantics for an oral dialogue system', Proceedings of the International Conference of Spoken Language Processing, Banff, Canada (1992).

6 Zue V, Glass J, Goodine D, Leung H, Phillips M, Polifroni J and Seneff S: 'The VOYAGER speech understanding system: preliminary development and evaluation', Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (1990).

7 Zue V: 'Toward systems that understand spoken language', IEEE Expert Magazine, pp 51—59 (February 1994).

8 Glass J, Flammia G, Goodine D, Phillips M, Polifroni J, Sakai S, Seneff S and Zue V: 'Multilingual spoken-language understanding in the MIT VOYAGER system', Speech Communication, 17, pp 1—18 (1995).

9 Strzalkowski T: 'Natural language information retrieval: TREC-4 report', Proceedings of the Fourth Text Retrieval Conference (TREC-4), NIST Special Publication 500-236 (1996).

10 Strzalkowski T, Guthrie L, Karlgren J, Leistensnider J, Lin F, Perez-Carballo J, Straszheim T, Wang J and Wilding J: 'Natural language information retrieval: TREC-5 report', Proceedings of the Fifth Text Retrieval Conference (TREC-5), NIST Special Publication 500-238 (1997).

11 Agarwal R: 'Towards a PURE spoken dialogue system for information access', Proceedings of the ACL/EACL Workshop on 'Interactive spoken dialog systems: bringing speech and NLP together in real applications', pp 90—97, Madrid (1997).

12 Carson J A and De Roeck A: 'The SNAP system: a natural language frontend to text and data bases', Proceedings of the Natural Language Processing and Industrial Applications Conference (NLP-IA), Moncton, Canada (1996).

13 WordNet (Five papers on WordNet): ftp://clarity.princeton.edu/pub/wordnet/5papers.ps.

14 Callan J P, Croft W B and Broglio J: 'TREC and TIPSTER experiments with INQUERY', Information Processing and Management, 31, No 3, pp 327—343 (1995).

15 Brill E: 'A simple rule-based part of speech tagger', Proceedings of the Third Conference on Applied Natural Language Processing, ACL, Trento, Italy (1992).

16 Brill E: 'Some advances in rule-based part of speech tagging', Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94), Seattle, Washington (1994).

17 Santorini B: 'Part-of-speech tagging guidelines for the Penn Treebank Project', Technical report MS-CIS-90-47, Department of Computer and Information Science, University of Pennsylvania (1990).

18 Allan J, Ballesteros L, Callan J P, Croft W B and Lu Z: 'Recent experiments with INQUERY', Proceedings of the Fourth Text Retrieval Conference (TREC-4), pp 49—63, NIST Special Publication 500-236 (1996).

Anne De Roeck is a senior lecturer and the head of the Department of Computer Science at Essex University.

She has an MSc in artificial intelligence, and has worked in many branches of natural language, both conventional and computational.

Udo Kruschwitz has a degree in computer science from the Humboldt University in Germany.

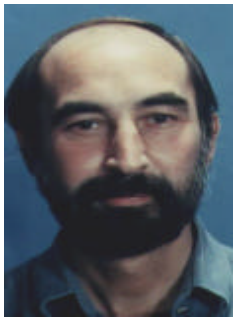
He also studied artificial intelligence in Edinburgh, and worked on the 'VERBmobil' natural lanaguage processing project in Berlin.

Paul Scott is a senior lecturer in the department of Computer Science at Essex University.

He holds a DPhil from Sussex in psychology, and has worked in several areas of artificial intelligence, especially machine learning.

Sam Steel is a senior lecturer in the department of Computer Science at Essex University.

He holds a PhD from Edinburgh in artificial intelligence, and is interested in modelling rational behaviour and in natural language as action.



Ray Turner is a professor in the department of Computer Science at Essex University.

He holds PhDs in both mathematics and philosophy, and is interested in the relation of logic to mathematics, computation and language.

Nick Webb studied Computer Science at Essex University, from where he has a BSc and an MSc.

He has also worked on the SNAP natural language processing project at Essex University.

Philip Neal has studied both actual and computational linguistics and holds a PhD in Middle High German as well as an MSc.

He has worked on computational linguistics applied to document capture at UMIST.