On the Tip of My Thought: Playing the Guillotine Game

Giovanni Semeraro, Pasquale Lops, Pierpaolo Basile, Marco de Gemmis
Department of Informatics - University of Bari "Aldo Moro", Italy
{semeraro, lops, basilepp, degemmis}@di.uniba.it

Abstract

In this paper we propose a system to solve a language game, called *Guillotine*, which requires a player with a strong *cultural* and *linguistic* background knowledge. The player observes a set of five words, generally unrelated to each other, and in one minute she has to provide a sixth word, semantically connected to the others. Several knowledge sources, such as a dictionary and a set of proverbs, have been modeled and integrated in order to realize a *knowledge infusion* process into the system. The main motivation for designing an artificial player for Guillotine is the challenge of providing the machine with the cultural and linguistic background knowledge which makes it similar to a human being, with the ability of interpreting natural language documents and reasoning on their content. Experiments carried out showed promising results, and both the knowledge source modeling and the reasoning mechanisms (implementing a spreading activation algorithm to find out the solution) seem to be appropriate. We are convinced that the approach has a great potential for other more practical applications besides solving a language game, such as semantic search.

1 Introduction

Words are popular features of many games, and they play a central role in many language games. A *language game* is defined as a game involving natural language in which word meanings play an important role [Littman, 2000]. Language games draw their challenge and excitement from the richness and ambiguity of natural language. Furthermore this type of games are inconsistent with the closed world assumption: no fixed sets of rules are sufficient to define the game play, relationships not explicitly stated to hold do not hold. In this paper we present a system that tries to play the *Guillotine* game. The Guillotine is a language game played in a show on RAI, the Italian National Broadcasting Service, in which a player is given a set of five words (clues), each linked in some way to a specific word that represents the unique solution of the game. She receives one word at a time, and must choose between two different proposed words: one is correct, the other one is wrong. Each time she chooses the wrong word, the prize money is divided by half (the reason for the name *Guillotine*). The five words are generally unrelated to each other, but each of them is strongly related to the word representing the solution. Once the five clues are given, the player has one minute to provide the solution. An example of the game follows: Given the five words *Capital, Pope, City, Colosseum, YellowAndRed*, the solution is *Rome*, because Rome is *Capital* of Italy, the *Pope* lives in Rome, Rome is a *City*, the *Colosseum* is in Rome and *YellowAndRed* is an alternative name for one of the Rome football teams. Often the solution is not so intuitive and the player needs different knowledge sources to reason and find the correct word. This paper presents OTTHO (On the Tip of my THOught), an information system that tries to solve the final stage of the game. We assume that the five words are provided at the same time, neglecting the initial phase of choosing the words, that only concerns the reduction of the initial prize.

The paper is structured as follows. Related work are briefly analyzed in Section 2. Section 3 describes the system architecture, while details about the modeling of different types of knowledge sources are in Section 4. The reasoning mechanism for finding the solution of the game is in Section 5 and experiments performed for evaluating the system are in Section 6. Conclusions and future works are in Section 7.

2 Related Work

The work on intelligent computer games has a long history [Maybury *et al.*, 2006] and has been one of the most successful and visible results of Artificial Intelligence research. We are particularly interested to games related to the language. The literature classifies these games in two main categories: 1) *word games* and 2) *language games*. *Word games* do not involve true language, because word meanings are not important. Examples of word games are *Scrabble*, in which players take turn placing letters in a grid to form words, and the *hangman*, in which a player must identify a hidden word by guessing letters from the alphabet. On the other side, *language games* strongly involve natural language, since word meanings play an important role. An example is *Wheel of Fortune*, with three players competing to be the first to solve a hangman-like puzzle, with the main difference that players are told the category of the answer, which provides a hint to the meaning of the answer phrase. Another example is

Who Wants to be a Millionaire?, in which the player is asked a series of multiple-choice trivial questions. This often requires common sense or knowledge on popular culture besides excellence in natural language processing. Another interesting language game is solving crossword-puzzles. The first experience reported in literature is Proverb [Littman *et al.*, 2002], that reached human-like performance on American crosswords using a great number of knowledge-specific expert modules. Another remarkable system is WebCrow [Ernandes *et al.*, 2005], the first solver for Italian crosswords and the first system that tackles a language game using the Web as knowledge base. Our philosophy for playing Guillotine is different. Much of Proverb's knowledge and success comes from large libraries of clues and solutions to past crossword puzzles. In Guillotine it makes no sense to learn from a set of previously solved games. WebCrow and Millionaire take their inspiration from the related work on question answering. In the Millionaire a Web-based knowledge agent is adopted to select one answer out of a small number of possible choices. WebCrow relies on a Web Search Module to deal with encyclopedic knowledge for the clue-answering process. Our system accesses different knowledge sources as well, but it is a bit different. Our "questions" are *single words*, and the answer is a *single word*. Co-occurrences of terms, providing the evidence of a strong relationship between words, is the key factor for finding a set of candidate words that likely contains the solution. We need to access several sources, such as a dictionary, an encyclopedia, a list of proverbs, etc. As for WebCrow and Millionaire, our system does not depend on the used language since modeling sources is a language independent process.

3 The System Architecture

Guillotine is a *cultural* and *linguistic* game, and for this reason we need to define an extended knowledge base for representing the *cultural* and *linguistic* background knowledge of the player. A huge number of examples of the game has been analyzed in order to identify a bunch of possible sources to build the extended knowledge base for playing *Guillotine*. The correlation existing between the clues and the solution has been analyzed, and the following *classes* or *types* of knowledge sources have been identified, ranked according to the frequency with which they were helpful in finding the solution of the game:

1) DICTIONARY: the word representing the solution is contained in the description of a lemma or in some example phrases using that lemma;
2) ENCYCLOPEDIA: as for the dictionary, the description of an article contains the solution, but in this case it is necessary to process a more detailed description of information;
3) PROVERBS AND APHORISMS: short pieces of text in which the solution is found very close to the clues;
4) ACRONYMS: the solution is one term of the acronym;
5) TITLES OF MOVIES, SONGS, BOOKS, POETRIES.

In order to exploit these types of sources it is necessary to organize data gathered from different sources of the same type, to process that information in order to extract and model relationships between words, and to define a reasoning mechanism that, given the clues, is able to select the correct solution of the game among a set of candidate words. The architecture of OTTHO, depicted in Figure 1, is modular and allows to plug in additional *ad-hoc* modules for enriching the background knowledge base without modifying the reasoning mechanism.

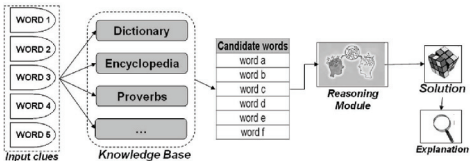

Figure 1: System architecture

4 Knowledge Base Modeling

It is important to model each type of knowledge source in order to make it usable in an efficient and effective way by the reasoning module. The above mentioned types of sources have different characteristics, therefore different heuristics should be used for building the model. Another important aspect is to define a uniform representation of that model. Since we are interested in finding relationships existing between words, the model of a knowledge source will be represented by the set of correlations existing between terms occurring in that specific source (a proverb, a definition in a dictionary, etc). We decided to use a *term-term matrix* containing terms occurring in the modeled knowledge source; each cell of the matrix contains the weight representing the degree of correlation between the term on the row and the one on the column. The computation of the weights is different for each type of knowledge source and takes into account several parameters, as described in the following subsections.

4.1 Modeling a Dictionary

The dictionary is a valuable knowledge source since the analysis of several examples of the game showed that it helps to find the solution (Section 6). We used the on-line De Mauro Paravia Italian dictionary (old.demauroparavia.it), containing 160,000 lemmas. We obtained a lemma-term matrix containing weights representing the relationship between a lemma and terms used to describe it in the dictionary. Each Web page describing a lemma has been preprocessed in order to extract the most relevant information useful for computing weights in the matrix. More specifically, each Web page contains:
DEFINITION of the lemma, which describes its possible meanings and some example phrases that use that lemma;
COLLOCATIONS, a group of words having a meaning not inferable from the meaning of the single words. It is a valuable source of information, albeit not available for all the lemmas.
The process for producing the matrix is as follows:

- The text of the Web page is processed in order to skip the HTML tags, even if the formatting information is preserved. Stopwords are eliminated and abbreviations used in the definition of the lemma are expanded.
- Weights are computed using a strategy based on a TF-IDF scheme, improved through the following heuristics:

– the relationship between a lemma and a term depends on the section in which the term occurs;
– terms in the description of the lemma formatted using bold or italic font are given higher weights than other terms, since this generally represents a strong relationship between that term and the lemma;
– terms occurring more than once in the definition are more important than terms occurring only once;
– terms closer to the lemma are given higher weights than other terms.

We also need to define some normalization criteria for weights with respect to the length of the definition in which the term occurs and the length of the entire dictionary. Given a term t_j occurring in the description of a lemma l_i, the weight w_{ij} representing the correlation between t_j and l_i is:

$$w_{ij} = \begin{cases} \frac{(k_d+\alpha)(1+\beta)}{L(Def_i)} log(\frac{N}{n_j}) \text{ if } t_j \text{ occurs in DEFINITION} \\ \\ \frac{(k_c+\alpha)(1+\beta)}{L(Coll_i)} \text{ if } t_j \text{ occurs in COLLOCATIONS} \end{cases} \quad (1)$$

- k_d and k_c are parameters useful to give higher scores to terms in the DEFINITION or in the COLLOCATIONS;
- α is a boosting factor for terms formatted using italic or bold font; β is a boosting factor for terms closer to the lemma in the DEFINITION or COLLOCATIONS;
- N is the cardinality of the vocabulary, n_j is the number of distinct lemmas correlated with t_j;
- $L(Def_i)$ ($L(Coll_i)$) is a normalization factor representing the length of DEFINITION (COLLOCATIONS).

The method computes a weight for each occurrence of t_j in the description of l_i. Each weight is different depending on the section in which the term occurs, the font used to write it and the position with respect to the lemma. For terms occurring more than once in the definition, the correlation between t_j and l_i is obtained by summing all the single correlations between the different occurrences of t_j and l_i.

The β boosting factor is computed using a Gaussian function $G(x)$ ($\mu = 0$, $\sigma = 2$), where x is the distance between the term and the closest occurrence of the lemma among those occurring in its description. We assign a null value to β when the distance between the term and the lemma is ≥ 5.

4.2 Modeling Proverbs

As for the dictionary, a TF-IDF strategy has been used for defining the weights in the term-term matrix modeling the knowledge source of proverbs[1]. The following heuristics for improving the weight computation have been also exploited:

- Terms in a proverb are correlated with all other terms in the same proverb;
- The closer the terms, the stronger their correlation, and the shorter the sentence, the stronger the correlation between terms;
- The greater the number of co-occurrences of two terms, the weaker their correlation.

Given two terms t_i and t_j, the correlation w_{ij} is defined as:

$$w_{ij} = \frac{occ_{ij}}{dist_{ij} \cdot len_{ij}} \quad (2)$$

where $dist_{ij}$ is the smallest distance between occurrences of t_i and t_j in the proverb, normalized with respect to the maximum distance observable in the proverb, len_{ij} is the length of the proverb containing terms t_i and t_j, normalized with respect to the maximum length observable in the corpus, and:

$$occ_{ij} = \begin{cases} log\frac{N}{n_{ij}} \text{ if } n_{ij} > 0 \\ \\ 0 \text{ if } n_{ij} = 0 \end{cases} \quad (3)$$

N is the total number of proverbs in the corpus and n_{ij} is the number of proverbs containing both t_i and t_j.

5 What Comes to Mind, and Why: the Reasoning Module

All of us have stored the meanings of thousands of words, as well as their spelling, grammatical features, contextual relations [Spitzer, 2000]. How are these meanings organized and coded? Is their organization, if there is any, different in each human being? More than two thousands years ago, Aristotele asserted that meanings are not unrelated within memory. He knew from his own experience that words come to mind in a sequence determined by meaningful relations of similarity or contrast and, hence, that words must be stored in the same way. You might try this out for yourself by performing this lexical task: Please say, as quickly as possible, for each of the five words in the following, the first word that comes to mind: *father*, *table*, *cold*, *sun*, *sing*. There is a good chance that you come up with the following associations: *mother*, *chair*, *hot*, *moon*, *song*. In order to find the solution for Guillotine, a human being has to perform a similar lexical decision task, but even more complicated: in one minute she has to find *one* word associated in some way to *all* the five clues. The main difference is that the answer is not as easy as the first word that comes to mind when reading the clues. Indeed, since clues for Guillotine are more focused than in the classical lexical decision task, the human being has to perform a *memory retrieval* task: the essential process to find the solution is the retrieval operation within the facts retained in her own knowledge.

What we have done by modeling several sources is a kind of "knowledge infusion" into the system, in order to create a memory of world facts and linguistic knowledge. The next step is to design an algorithm for retrieving the most appropriate *pieces of knowledge* associated with the clues, simulating the cognitive mechanism of a human being in the most faithful way. A large number of studies on the meaningful relations between "psychologically" associated words have produced data that were interpreted some decades ago as evidence for a *network structure of semantic memory*. Words and their meanings are not stored alphabetically in the mind, nor in a random way. Instead, they are stored in a network-like structure [Anderson, 1983]. Within such a network, the

[1] $1,600$ proverbs from *http://web.tiscali.it/proverbiitaliani* and *http://giavelli.interfree.it/proverbi_ita.html*

meaning of a word is represented by nodes as well as by the relations of these nodes with other nearby nodes. This theory of mental organization of our lexicon inspired the design of WordNet [Fellbaum, 1998]. The network theory of semantic memory fits well with what has been termed the *spreading activation model*. The pure spreading activation model consists of a network data structure upon which simple processing techniques are applied. The network data structure consists of nodes interconnected by links. Links may be labeled and/or weighted and usually have directions. The processing is initiated by labeling a set of *source nodes* with activation weights and proceeds by iteratively propagating that activation to other nodes linked to the source nodes. For each iteration, a termination condition is checked in order to end the search process over the network. We decided to adopt this model as reasoning mechanism of OTTHO. In the network for Guillotine, nodes represent words, while links denote associations between words, obtained from the sources processed as described in Section 4. The spreading activation is triggered by words given as clues. The activation of clues causes words with related meaning (as modeled in the sources) to become active. At the end of the weight propagation process, the most "active" words represent good candidates to be the solution of the game.

5.1 Building the Network

This section describes the method adopted to construct the Spreading Activation Network (SAN) for Guillotine, along with the weighting scheme for the edges. We assume that *M* knowledge sources $KS_1, \ldots, KS_M$ have been modeled as described in Section 4 and that five clues $k_1, \ldots, k_5$ are provided. The idea is to populate the graph with nodes by running n expansion phases (we set $n = 4$ since we noticed that for $n \geq 5$ no nodes are activated; more details in Section 5.2). For the sake of simplicity, together with the general method, we will show an example with the only two clues *apple* and *sunrise*, and the two sources DICTIONARY and PROVERBS. Initially, source nodes $N_1, \ldots, N_5$ are added to the network. Each N_i is labeled with the word k_i given as a clue. In the first expansion phase, for each k_i, the most related words and corresponding correlation weights are extracted from KS_m $(m = 1, \ldots, M)$. We take as the most related words for k_i only those k_j with correlation weights $w_{ij} \geq \lambda$ (in our experiments $\lambda = 0.1$). At the end of this query process, we get M lists of pairs L_i^m correlated with k_i:

$$L_i^m = \langle (k_{m_1}, w_{im_1}), \ldots, (k_{m_h}, w_{im_h}) \rangle$$

where k_{m_j} is the j^{th} word extracted from KS_m and w_{im_j} is the correlation weight between k_i and k_{m_j}. In our simple example, at this stage we have:

$L_{apple}^{dictionary} = \langle (fruit, 1.42), (jam, 0.55), (peel, 0.26) \rangle$
$L_{apple}^{proverbs} = \langle (day, 1.47), (doctor, 2.41), (away, 1.47) \rangle$
$L_{sunrise}^{dictionary} = \langle (beginning, 1.45), (day, 1.41), (sunset, 0.55) \rangle$
$L_{sunrise}^{proverbs} = \langle \ \rangle$

For each pair (k_{m_j}, w_{im_j}) in L_i^m, one node labeled with k_{m_j} is added to the SAN and linked to the source node k_i. The edges are oriented from the source node to the nodes of the related words and labeled with w_{im_j}. Thus, an edge represents the association between two words, while the correlation weight measures the strength of that relationship. Once the process is completed for each L_i^m, the first stage is over. In the next expansion phase, the same process continues for the nodes added during the previous phase. Hence, new nodes are linked to *fruit*, *jam*, etc. Notice that whether for some pair (k_{m_j}, w_{im_j}) in L_i^m, a node labeled with k_{m_j} is already included in the SAN, and edge labeled with w_{im_j} is created from k_i to that node, without adding any new node. The status of the SAN after two expansion stages for our small example is depicted in Figure 2. After $n - 1$ expansion stages, the SAN stops growing. During the n^{th} phase, we only look for correlations with (no source) nodes already existing in the SAN. For each pair (k_{m_j}, w_{im_j}) in L_i^m, if a node labeled with k_{m_j} is in the SAN, an edge is created from k_i to that node and labeled with w_{im_j}, otherwise the pair is discarded. The motivation for this choice is that we want to avoid the addition of nodes "too far" from the source nodes. Actually, the highest n the lower the probability that new nodes added at the n^{th} stage will be fired, due to the low activation input they would receive. The final step for completing the SAN is the normalization of the correlation weights (that might vary in different ranges), since weights come from different sources and are computed according to different heuristics. After normalization all edge weights are in the $[0, 1]$ interval.

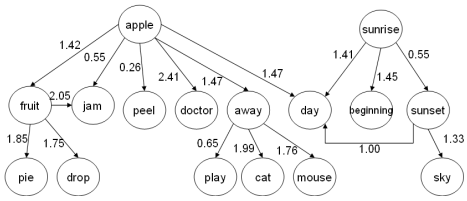

Figure 2: The SAN after two expansion stages

5.2 The Spreading Activation Strategy

Our spreading activation strategy consists of iterations (*pulses*). Given a SAN populated by nodes $N_1, \ldots, N_T$, each node has an associated activation value at iteration p, $A_i(p)$. A firing threshold F determines if a node is fired, that is whether it can spread its activation value over the SAN. At each iteration, every node propagates its activation to its neighbors as a function of its current activation value and the weights of the edges that connect it with its neighbors. The spreading strategy is described in the following.

STEP #1 - INITIALIZATION: Iteration $p = 1$. The SAN is initialized by setting all activation values $A_i(p) = 0$, with exception of the clue nodes whose activation value is set to 1.

STEP #2 - MARKING: Each node N_i with activation value $A_i(p) \geq F$ is marked as FIRED.

STEP #3 - FIRING: For each fired node N_i, its output value is computed as a function of its activation level:

$$O_i(p) = \frac{A_i(p)}{p}(1 - \alpha_i) \tag{4}$$

where the factor $\frac{1}{p}$ decreases the output of a node as "pulses" go further in the SAN, while α_i is a penalty factor

that reduces the influence of a node N_i on its neighbors according to the number of associations branching out from N_i. Let $NODES(n)$ be the set of nodes expanded during the n^{th} stage of the SAN building procedure, in our example $NODES(1) = \{apple, sunrise\}$, $NODES(2) = \{fruit, jam, peel, doctor, away, day, beginning, sunset\}$. Let $\bar{n}$ be the expansion stage for N_i, the penalty factor is:

$$\alpha_i = \frac{edges(N_i)}{\sum_{N_j \in NODES(\bar{n})} edges(N_j)} \quad (5)$$

where $edges(N_i)$ is the number of edges branching out from N_i. In other words, α_i is the fraction of *all* edges created in an expansion stage branching out from a single node. For example, in the SAN in Figure 2, $\alpha_{apple} = 6/9$ and $\alpha_{sunset} = 2/8$. The motivation for this choice is that a large fan-out means that the node has very broad semantic meaning. The spreading from this kind of node should be more limited than that coming from "more specific" nodes.

STEP #4 - SPREADING: For each link connecting the FIRED node N_i to the target nodes N_j, adjust $A_j(p)$:

$$A_j(p+1) = A_j(p) + w_{ij} O_i(p) \quad (6)$$

Notice that, in order to avoid loops, once a node has been fired it cannot be fired again.

STEP #5 - TERMINATION CHECK: $p = p + 1$. If $p < maxpulses \wedge$ FIRED(p) then go to STEP #2 otherwise END (FIRED(p) = true iff there is at least one node fired at pulse p).

The final result of the spreading activation process is the activation level for each node in the SAN at termination time. As an example, we reported in Table 1 the status after $n = 2$ pulses for some nodes belonging to the SAN depicted in Figure 2 (firing threshold 0.2). Activation values of fired nodes are in bold. At pulse 1, only *apple* and *sunrise* are fired and spread their output. The activation values of their neighbors are updated. Notice that the activation value for node *day* has been updated for spreading from both *apple* and *sunrise*. After the second pulse, the best candidate for the solution is *day*, whose activation value has been updated in both pulses.

6 Experimental Evaluation

The goal of experiments was to measure the number of games solved by the system. A game is solved whether the solution occurs in the Candidate Solutions List (CSL) produced by the system, which contains the labels of the nodes with the highest activation levels at termination time. A CSL is obtained from the SAN built for a game by selecting all the nodes N_i whose activation values are greater than a fixed threshold λ at termination time. This constraint is set in order to avoid the inclusion of "weak" solutions in the list. All selected nodes are ranked according to their activation level (descending order), and the labels of first k nodes are included in the list. Hereafter, CSL_k^λ denotes the CSL built with constraints k and λ. The reason for this evaluation method is that the ranking of nodes might be a good strategy for determining some *candidate* solutions, but more sophisticated techniques are needed to select a *unique* answer among them, as the rules of the game require. The answer is the candidate solution which better "explains" the maximum number of clues. At the current stage of the system, the component that picks up the answer from the CSL, namely the SOLVER, is not yet complete, therefore the evaluation is limited to the best k candidate solutions in the CSL. The dataset is a collection of $N = 50$ games attempted during the show by human players. On average, they were able to provide the correct solution in 8 cases, thus the "human accuracy" is 16%. For evaluating the system, we adopted two performance measures: Absolute Accuracy (AA) and Relative Accuracy (RA). Let N_k^λ be the number of games for which the correct solution is found in CSL_k^λ:

Node	$A_i(1)$	$O_i(1)$	$A_i(2)$	$O_i(2)$	$A_i(3)$
apple	**1.00**	0.33	1.00	0.00	1.00
sunrise	**1.00**	0.67	1.00	0.00	1.00
jam	0.00	0.00	0.18	0.00	0.18
doctor	0.00	0.00	**0.80**	0.40	0.80
away	0.00	0.00	**0.49**	0.15	0.49
day	0.00	0.00	**1.43**	0.72	1.57
sunset	0.00	0.00	**0.37**	0.14	0.37
cat	0.00	0.00	0.00	0.00	0.30

Table 1: Example of output and activation levels for nodes

$$AA = N_k^\lambda / N \quad (7)$$

Let $\#SOLVABLE_{Dictionary,Proverbs}$ be the number of games in the collection for which the solution can be found searching the clues in the term-term matrices corresponding to the knowledge sources currently available to the system. Relative Accuracy is the fraction of $\#SOLVABLE_{Dictionary,Proverbs}$ successfully solved:

$$RA = N_k^\lambda / \#SOLVABLE_{Dictionary,Proverbs} \quad (8)$$

For our dataset $\#SOLVABLE_{Dictionary,Proverbs} = 42$, thus 84% of the solutions occurs in the sources available to the system, that is these solutions are included in the SANs built for these games. Anyway, the presence of the solution in the SAN does not guarantee that the system is able to solve the game easily. For example, if the solution for a game is connected just to a single clue, the activation level of the solution can be too low to be part of the CSL. To give a more precise idea of the difficulty of the games in the dataset, we also analyzed the *coverage* of the clues by using DICTIONARY and PROVERBS. A clue is *covered* by the available sources whether an entry for that clue is found in (at least) one term-term matrix. The coverage of a game is defined as the number of covered clues (it varies from 0 to 5). High coverage coupled with the presence of the solution in the SAN might be indicative of games that can be actually solved by our system. Results of coverage analysis are in Table 2. Values in the 2nd column represent the number of games whose coverage is reported in the 1st column. The 3rd column reports $\#SOLVABLE_{Dictionary,Proverbs}$ (detailed with respect to coverage), while the remaining columns show N_k^λ, where $\lambda = 0.1$ and different values are set for k. For 4 of the 42 solvable games (9.5%) the solution is hard to find out due to low coverage (less than 3). Indeed, results obtained

COVERAGE	#GAMES	#SOLVABLE	$N_{k=10}^{\lambda=0.1}$	$N_{k=15}^{\lambda=0.1}$	$N_{k=100}^{\lambda=0.1}$
1	1	0	-	-	-
2	2	1	0	0	0
3	3	3	0	0	0
4	12	10	1	1	6
5	32	28	3	4	20
	50	42	4	5	26

Table 2: Coverage analysis

	$N_{k=10}^{\lambda=0.1}$	$N_{k=15}^{\lambda=0.1}$	$N_{k=100}^{\lambda=0.1}$	$N_{k=150}^{\lambda=0.1}$
AA	0.08	0.10	0.52	0.68
RA	0.10	0.12	0.62	0.90

Table 3: Accuracy of the system

manifested the high difficulty to solve games with low coverage: none of them has been solved by the system. On the other hand, 88% of games (44 out of 50) has coverage $\geq$4. For 38 of them, the solution occurs in the SAN, therefore 90.4% of the solvable games (38 over 42) has high coverage. When stricter contraints are set for k and λ ($k = 10$ or 15, λ=0.1), the system is able to solve 5 games at most, which corresponds to 13% of the solvable games with high coverage. When a larger list of candidate solutions is generated ($k = 100$), this percentage increases up to 68%. Detailed results of accuracy, as defined in equations (7) and (8), are reported in Table 3. When strict constraints are set on the CSLs, both absolute and relative accuracy of the system are lower than human accuracy (16%), even if the difference is not so marked. By the way at this stage of development the cultural knowledge of the artificial player is very limited. More encouraging results are obtained by allowing more extended lists of candidate solutions (3rd and 4th columns in Table 3). We are aware that it is not fair to compare these higher accuracy values with human accuracy, but these results suggest that the system *can* reach the solution in at least half of the attempted games. The main conclusion after this first stage of experiments is that the most promising strategy for the development of the SOLVER module is to process large lists of candidate solutions, without setting severe constraints on them. As told before, the SOLVER should select the unique answer which "better explains" the input clues, thus it will discard solutions not able to explain the clues rather than discard *a priori* candidate solutions with low activation values. We are also evaluating whether showing words in the CSLs to the player might provide a useful hint to solve the game.

7 Conclusions and Future Work

We have proposed an artificial player for a language game consisting in guessing a hidden word semantically related to five words given as clues. The basic idea is to define a knowledge base for representing the *cultural* and *linguistic* background knowledge of the player. This was realized by modeling several knowledge sources as term-term matrices, in order to create a memory of world facts and linguistic knowledge into the system. The brain of the artificial player is a spreading activation algorithm able to retrieve the most appropriate *pieces of knowledge* associated with the clues. Experiments showed that the system accuracy is slightly lower than that of the human player, but there is room for improvements: indeed the system overall architecture allows to plug in additional sources. We are currently working for integrating encyclopedic knowledge (Wikipedia) and sources containing movie and song titles. We are convinced that the proposed approach has a great potential for other more practical applications. The system could be used for implementing an alternative paradigm for *associative retrieval* on collections of text documents [Crestani, 1997], in which an initial indexing phase of documents can *spread* further "hidden" terms for retrieving other related documents. The identification of hidden terms might rely on the integration of specific pieces of knowledge relevant for the domain of interest. This might represent a valuable strategy for several domains, such as search engine advertising, in which customers' search terms (and interests) need to be matched with those of advertisers [Goldfarb and Tucker, 2008]. Spreading activation can be also effectively combined with document retrieval for semantic desktop search [Schumacher *et al.*, 2008].

References

[Anderson, 1983] J. R. Anderson. A Spreading Activation Theory of Memory. *Journal of Verbal Learning and Verbal Behavior*, 22:261–295, 1983.

[Crestani, 1997] F. Crestani. Application of Spreading Activation Techniques in Information Retrieval. *Artificial Intelligence*, 11(6):453–482, 1997.

[Ernandes *et al.*, 2005] M. Ernandes, G. Angelini, and M. Gori. WebCrow: A Web-Based System for Crossword Solving. In M. M. Veloso and S. Kambhampati, editors, *Proc. 20^{th} Nat. Conf. Artif. Intell., 17^{th} Innov. Appl. Artif. Intell.*, pages 1412–1417. AAAI Press/MIT Press, 2005.

[Fellbaum, 1998] C. Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[Goldfarb and Tucker, 2008] A. Goldfarb and C. Tucker. Economic and Business Dimensions: Search Engine Advertising. *Comm. of the ACM*, 51(11):22–24, 2008.

[Littman *et al.*, 2002] M. L. Littman, G. A. Keim, and N. Shazeer. A probabilistic approach to solving crossword puzzles. *Artificial Intelligence*, 134:23–55, 2002.

[Littman, 2000] M. L. Littman. Review: Computer Language Games. In T. A. Marsland and I. Frank, editors, *Computers and Games, 2nd Int.Conf., Revised Papers*, volume 2063 of *LNCS*, pages 396–404. Springer, 2000.

[Maybury *et al.*, 2006] M. Maybury, O. Stock, and W. Wahlster. Intelligent Interactive Entertainment Grand Challenges. *IEEE Intelligent Systems*, 21(5):14–18, 2006.

[Schumacher *et al.*, 2008] K. Schumacher, M. Sintek, and L. Sauermann. Combining Fact and Document Retrieval with Spreading Activation for Semantic Desktop Search. In *The Semantic Web: Research and Applications, Proc. of ESWC 2008*, LNCS 5021, pages 569–583. Springer, 2008.

[Spitzer, 2000] M. Spitzer. *The Mind within the Net: Models of Learning, Thinking, and Acting*. MIT Press, 2000.