Connectionist and genetic approaches to achieve IR

Mohand Boughanem, Claude Chrisment,
Josiane Mothe, Chantal Soule-Dupuy, and
Lynda Tamine

Institut de Recherche en Informatique de Toulouse
118 Route de Narbonne, 31062 Toulouse CEDEX, France

Abstract. In the past few decades, knowledge based techniques have made an impressive contribution to intelligent information retrieval. These techniques stem from research on artificial intelligence, neural networks (NN) and genetic algorithms (GA) and are used to answer three main IR tasks : information modelling, query evaluation and relevance feedback. The paper describes an IR approach based on connectionist and genetic approaches. Our goal is to take benefits of these techniques to fulfill the user information needs. More precisely a multi-layer NN, Mercure, is used to represent the document space in an associative way, to evaluate the query using spreading activation and to implement a relevance feedback process by relevance back-propagation. Another query reformulation technique is investigated which uses the GA approach. The GA generates several queries that explore different areas of the document space. Experiments and results obtained with both techniques are shown and discussed.

1 Introduction

The explosive growth of the Internet and other information sources had lead to the availability of large amounts of information. As a consequence, the effort required to retrieve relevant information in these large information collections has become significantly more difficult.
Moreover, it is well-known that the initial user's query generally does not retrieve all the relevant documents. The initial query formulation is based on some parameters such as the user's need for information, his/her knowledge on the subject, on the system used and on the collections. It is obvious that a user's query would be more efficient if it could take into account the way the documents are written or indexed. Unfortunately, most of the time the users have little knowledge on all these inner features. Moreover, the users only have a vague idea of what they can find in the collection and how the system works. Thus, they mostly proceed by trial and error mechanisms. After submitting a first query they use the retrieved documents in order to improve their query formulation.
Automatic relevance feedback has been introduced in order to automate the query reformulation or to help the users when reformulating their queries. According to relevance feedback, the user's assessments on previously retrieved

documents are automatically combined with the initial query to build a new query. The query is modified by eliminating terms that occur in non relevant documents, adding terms from the relevant documents or modifying the initial term query weights according to the document relevance. This process has become an important and active research area. It has been implemented in various models (e.g. vector space and probabilistic models, Bayesian network) and has been shown to be efficient to improve retrieval [18], [20], [23], [34],[37].

More recently, there has been a growing interest in machine learning techniques with the same general goal of providing adaptive systems. Among these techniques, the Neural Network (NN) approach has been used [7], [13], [28], [43], [44], because many of its features can be useful in achieving the complex task of information retrieval. First, the associative representation of the information is the base of the implementation of spreading activation searches [2], [39]. The other main feature of the connectionist approach is its learning abilities which is interesting and useful in implementing adaptive systems. In this paper, the NN system we have implemented is presented and includes some experiments.

An alternative method has been studied in the literature to implement adaptive searching through Genetic Algorithms (GA). The GA approach [21] is based on both the principles of natural evolution processes and Darwinian survival theory. It has proved to be efficient as a stochastic optimization method. GA have been applied in information retrieval to optimize document descriptions [17] and to improve query formulations [2], [11], [45].

Nevertheless, whatever the technique used to reformulate the queries, these techniques assume that the relevant document set can be retrieved using a single query and that a single query can partition the relevant and the non relevant documents. One can agree that the relevant documents are not necessarily all similar to each other but can be confined in different places according to the indexing term space. This paper will present a process that combines NN and GA in order to implement a multi-query reformulation that aims at addressing the latter problem. The GA is used to build a population of queries that explore several areas of the document space.

The paper is organized as follows. First we present the Mercure system based on a multi-layered NN model. We describe the spreading activation search mechanism used and the way a query can be reformulated by back-spreading activation to improve retrieval. Experiments and results evaluating the system on samples from the TREC6 collection are discussed. The second part of the paper presents our approach combining NN and GA. Experiments and results carried out on AP88 TREC sub-collection are presented. The last section concludes the paper and outlines areas that need further work.

2 Overview of Neural Networks and Genetic Algorithms in Information Retrieval

Machine learning techniques, ie. automatic knowledge acquisition from examples, have provided new capabilities to Information Retrieval and indexing [11]. Among them, multi-layered neural networks and genetic algorithms have been used. Hybrid representations have also been tested. These techniques are presented below, as well as references to various implementations.

Neural Networks and Information Retrieval

The foundation of formal neurones and neural algorithms were laid in the 1950s [11], [36]. In connectionist models, knowledge is represented by a single layer of interconnected neurones (features maps) or by a multi-layered network. These two main approaches have been used in IR. Indeed, artificial neural networks provide convenient knowledge representation for IR applications and are proving to be active. In addition, their learning abilities mean that fulfill the goal of implementing adaptive systems.
Formal neurones represent information retrieval objects such as keywords, references, document excerpts or document classes. The formal synapses are implemented through weighted links that represent the relevance levels of the associations - that may be learned - between the formal neurones.
In IR, multi-layered models are used to implement spreading activation searches [39]. In most of the models, each document and each index term is represented by a node. These nodes are associated by weighted links. A query first activates some term nodes, this activation is then spread through the network to activate document nodes. This approach has been used in many works [1], [2], [9], [15], [29], [43].
In fact, the associative representation feature enables the linking of documents and term nodes and also thelinking of terms themselves and of documents themselves. For example, term nodes can be linked by semantic links (e.g. co-occurrence, synonymy) [30]. Thus, documents can be activated even if they are not directly linked to activated terms. In addition, multi-layered models are of interest owing to their learning capabilities. Indeed, the learning process used in a multi-layer neural network [14], [35], consists of presenting a training set of input-desired output pairs as examples, comparing the network-obtained output to the desired output, and computing the error. This error is back-propagated through the network and then the connection strengths are changed, minimizing the error in a next spreading activation. Several promising research peojects have been performed using this approach in IR [2], [4], [15], [28], [30], [43].
Belew [2] presents a multi-layered network using relevance feedback. In this system, the goal of the learning process was to create new connections between terms and between documents. Relevance feedback from users was

used to modify the document representations and the result was a consensual meaning of terms and documents shared by different groups of users.
Wilkinson et al. [43] implement the vector space model in a multi-layered neural network. They have shown that spreading activation through related terms can improve retrieval performance.
Kwok implements a multi-layered network based on a probabilistic IR model. A modified Hebbian learning rule is used [28], [29]. The model implements query expansion and modification via an artificial neural network and relevance feedback.
Wong et al. [44] present a model based on an adaptive bilinear retrieval model. They implement this model by using a three layer feed-forward neural network. They recorded some problems with their learning process related to the large dimensionality of the term association matrix. Their preliminary results reported in [44] suggest that neural networks should be explored further.
Crestani [13] reports experiments on relevance feedback based on a three-layered NN. This technique is compared with a probabilistic relevance feedback and the results have shown that using small bases (Cranfield) and with a too few training, probabilistic relevant feedback gives better results. He concludes that further research using different NN architectures are necessary.
The models based on single layer networks are less numerous. Following the earlier works of Mozer [31], Kohonen feature maps [25] have also been used in IR [32]. Feature maps provide unsupervised learning techniques that are used in IR to automatically classify the documents according to their features (the indexing terms). Once the learning stage is terminated, a "winner takes all" activation process is used to retrieve the most matching class. Poincot et al. [32] present a multi-layered feature map network that provides a promising technique to solve the problem of the size of both the feature map and the feature vectors.

Genetic Algorithms and Information Retrieval

Genetic algorithms are stochastic optimization methods developed by Holland [21] and based on both the principles of natural evolution processes and Darwinian survival theory. GA provide self adaptability properties. In such algorithms, an individual is represented by a set of genes and a population of individuals undergoes an evolution which consists of a sequence of transformations -crossover and mutation of the genes. A population of potential solutions is renewed at each generation by encouraging the reproduction of the fittest ones. After a number of generations, the program converges to the best individual representing the optimal solution.
In information retrieval, individuals are represented by documents or queries and genes by (weighted) keywords or indexing terms. Most of the approaches take advantage of relevance feedback information. The initial population is (randomly) initialized with a set of queries or a set of documents. The GA uses the relevance judgments in order to build either an optimal query [11],

[26], [45] or optimal document representations [17].
Gordon adopted GA to derive better descriptions of documents as presented in [17]. Each document is assigned N descriptions where each description is a set of indexing terms. Genetic operators and relevance judgement are applied to the descriptions in order to build the best document description. The author has shown that the GA produces better document descriptions than the ones generated by the probabilistic model.
Gordon has exploited these results and has defined a classification method based on clustering the relevant documents for a specific query [17].
Yang et al. [45] presents a GA to query optimization by re-weighting the document term indexing without query expansion. They used a selection operator based on a stochastic sample, a blind crossover at two crossing points, and a classical mutation to renew the population of queries. The experiments showed that the queries converge to their relevant documents after six generations.
Kraft et al. [26] apply a GA programming in order to improve the weighted boolean query formulation. The documents are viewed as vectors of index terms. A weighted boolean query is represented as a chromosome in the Koza's genetic model [27]. The goal of the GA is to modify the query in order to improve the search performance in terms of recall and precision. Their first experiments showed that the GA programming is a viable method for deriving good queries.

Hybrid approaches

Different hybrid models combine genetic algorithms and neural networks. Chen et al. [12] present a hybrid genetic and neural network based system called GANNET. This system performs concept (keyword) optimization for user-selected documents using GA and uses the optimized concepts to perform concept exploration in a Hopfield net [22] representing related concepts. The retrieving process is cyclic and is done in two stages. The first stage is the concept optimization, the GA manipulates input documents and their associated keywords to generate an initial set of optimized concepts. The second stage is the concept exploration where the set of optimized concepts is used by the Hopfield net to produce other relevant concepts (new genes suggested by nature) which are included in the GA for the next concept optimization. This process is repeated until there is no further improvement.
Other systems [19], [24] perform symbolic induction using genetic algorithms and a fuzzy connectionist expert system. Some results from empirical studies are discussed in [24]. These studies have shown that the performances were improved using genetic algorithms for neural network design. Belew et al. [3] have also experimented the combination of connectionist learning algorithms and GA. They compare the results obtained by the hybrid system with the ones obtained using each technique separately. They conclude that the GA well-complement the connectionist techniques.

3 Mercure : a neural network based IRS

Mercure [4], [7] is an information retrieval system based on a neural network approach. The model is a multi-layered network that allows document retrieval using a spreading activation process and query optimization using relevance back-propagation.

3.1 The Mercure neural network model

As shown in Fig. 1, each layer of the Mercure NN represents a different type of information. More precisely, the network is composed as follows:

- An input layer representing the user's information need,
- A term neurone layer,
- A document neurone layer,
- An output layer representing the result of the query evaluation (search).

The neurones are interconnected by weighted links that are computed based on earlier and current works in IR.

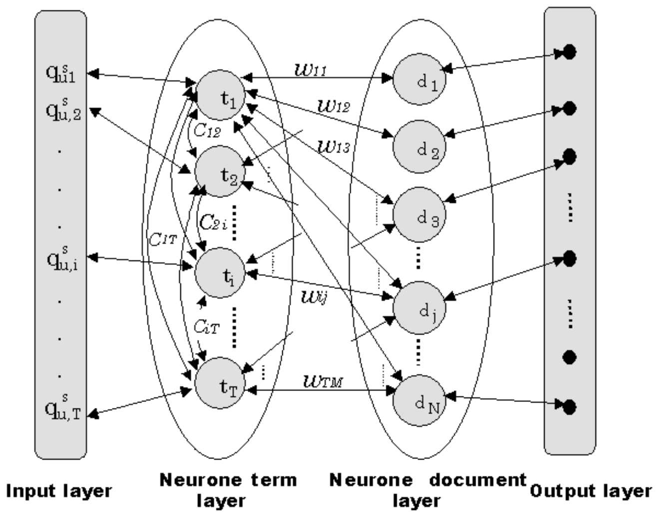

Fig. 1. The multi-layer neurone network of Mercure

With:
T the total number of indexing terms,
N the total number of documents,

$q_{ui}^{(s)}$ the weight of the term t_i in the query u at the stage s,
t_i the neurone corresponding to the term t_i,
d_j the neurone corresponding to the document d_j,
w_{ji} the weight of the link between the term t_i and the document d_j.

A document is represented by a list of terms that result from the indexing process. A document neurone corresponds to a document and a term neurone corresponds to an indexing term. These two kinds of neurones are interconnected by bi-directional weighted links that represent indexing links. The weight of these links is based on a combination of the term frequency (tf) and the inverse document frequency (idf). The formula we use is inspired by the Okapi [33] and Smart [38] term weight and has been evaluated in the TREC6 Conference. The formula used is the following :

$$w_{ji} = \frac{(1 + log(tf_{ji})) * (h_1 + h_2 * log(\frac{N}{n_i}))}{h_3 + h_4 * \frac{Doclen_j}{AvgDoclen}}$$

where
tf_{ji} is the frequency of t_i in d_j,
n_i is the number of documents that contain the term t_i,
$Doclen_j$ is the d_j document length (in words without stop words),
$AvgDoclen$ is the average document length,
h_1, h_2, h_3, h_4 are parameters which depend on the database and possibly on the nature of the queries. Their values have been determined by a series of experiments [6] and are given by $h_1 = 0.8$, $h_2 = 0.2$, $h_3 = 0.8$, $h_4 = 0.2$.

The input and output layers, representing respectively the user's queries and the system results are involved during the searching process and are linked respectively to the term layer and the document layer. Both input and output layers are in fact represented by vectors. The input vector is computed from the user's query whereas the output vector is computed using a spreading activation process as described in the following paragraphs.

Spreading activation process

The spreading activation process has been defined for the artificial neural network based on the way the brain neurones function. It works as follows. At a given time, each neurone is defined by an input, an activation, an output and an output threshold value. It is also defined by functions to compute the activation and output values, as well as to propagate the signal.
Giving N_i the neurone i and r_{ji} the weight of the link between the neurone N_i and the neurone N_j . The N_i neurone values are defined as follows:

- The Input value is

$$In(N_i) = \sum_j Out(N_j) * r_{ji}$$

- The Activation at stage c is

$$Activation^c(N_i) = Activation^{c-1}(N_i) + g(In(N_i))$$

At the initial stage ($c = 0$), $Activation^0(N_i) = 0$

- The Output is

$$Out(N_i) = \begin{cases} Activation^c(N_i) \text{ if } Activation^c(N_i) > Threshold \\ 0 \text{ otherwise} \end{cases}$$

Where $Threshold$ is a positive constant (we chose $Threshold = 0$ in our experiments).

Various functions $g()$ can be used to compute the activation value. We choose the identity function, taking into account that several functions, such as the sigmoid function, have been used in previous experiments [5].

3.2 Query evaluation

A query is evaluated using the spreading activation process described below. The user's query is used to compute the initial input (on the input layer) which is spread through the network to the output layer. The output results give a ranked list of documents. More precisely, the process is as follows:

- The user's query is transformed into a list of terms by a series of query processing modules (term extraction, stemming, weighting, ...). As a result of the query processing modules, the input vector is computed as $Input_u^{(s)} = Q_u^{(s)} = (q_{u1}^{(s)}, q_{u2}^{(s)}, ..., q_{uT}^{(s)})$. A $q_{ui}^{(s)}$ value represents the weight of the query term t_i in the query at the stage s ($s = 0$ for the initial search) and is computed as follows:

$$q_{ui}^{(s)} = \frac{(1 + log(tf_{ui})) * log(\frac{N}{n_i}))}{\sum_{j=1}^{T}(1 + log(tf_{uj})) * log(\frac{N}{n_j})^2}$$

using the same notations as below
- This input is applied to the term layer. Each term neurone computes an input value,

$$In(t_i) = q_{ui}^{(s)}$$

And then an output value,

$$Out(t_i) = g(In(t_i))$$

- These signals are propagated forward through the network. Each document neurone computes an input and an output value:

$$In(d_j) = \sum_{i=1}^{T}(Out(t_i) * w_{ji})$$

then,

$$Out(d_j) = g(In(d_j))$$

- Finally, the output vector corresponding to the output layer is :

$$Output = (Out(d_1), Out(d_2), ..., Out(d_N))$$

The output value computed by the document neurones enables the ranking of the activated document neurones and therefore of the retrieved document list. In addition, the use of an output function enables the forcing of the output values to be bound to a given interval [2].

3.3 Query reformulation by relevance back-propagation

The relevance feedback query reformulation is based on back-spreading the user's satisfaction on the retrieved documents, from the output layer to the input layer [4], [30].

- The top retrieved documents are judged and a *relevance value* is assigned to each document (positive for relevant documents, negative for non-relevant documents and nil for non-judged documents).
- These values are used to compute the *DesiredOutput* vector

$$DesiredOutput = (des_1, des_2, ..., des_N)$$

$$des_j = rel_j * \frac{1}{Nb}$$

Where rel_j is either negative, positive or nil. These values can be given by the user. In our experiments, $rel_j = 1$ for relevant documents and $rel_j = -0.75$ for non relevant documents. Nb is the number of relevant documents for a positive rel_j ; Nb is the number of non relevant documents for a negative rel_j and $des_j = 0$ for non judged documents.

- This output is in fact considered as an "input" in the back-spreading process and is applied to the document layer. Each document neurone computes an input value,

$$In(d_j) = des_j$$

and then an output signal,

$$Out(d_j) = g(In(d_j))$$

- This activation is back-spread to the term layer. Each term neurone computes an input value,

$$In(t_i) = \sum_{j=1}^{N} (w_{ji} * Out(d_j))$$

and then an output signal,

$$Out(t_i) = g(In(t_i))$$

- Finally, a new vector is computed for the input layer corresponding to the new query

$$Input_u^{(s+1)} = Q_u^{(s+1)} = (q_{u1}^{(s+1)}, q_{u2}^{(s+1)}, ..., q_{uT}^{(s+1)})$$

with

$$q_{ui}^{(s+1)} = M_a * q_{ui}^{(s)} + M_b * Out(t_i)$$

where M_a and M_b are parameters which are used to build new query schemes. Their values have been tuned and determined by a series of experiments and been set to $M_a = 2$ and $M_b = 0.75$

This new input (new query) can then be used in a new spreading activation process to retrieve the documents according to this improved query. This process (propagation of the input and back-propagation of the desired output) can be repeated several times.

3.4 Experiment results

The experiments reported in this paper have been carried out on the TREC6 Ad-hoc and French collections (Table 1) [41].

Table 1. Details of the collections used

Collection	Ad hoc Trec6	French Trec 6
Document number	555 625	141 656
Query number	25 (topics 301-325)	21

The goal of these experiments was to evaluate the automatic relevance back-propagation vs an initial search. The process was the following one. An initial search was performed using initial queries built from the *Title*, the *Descriptive* and the *Narrative* fields of the TREC topics. The x top documents were assumed as relevant (we used $x = 12$) whereas the low ranked documents (from 501 to 1000) were considered as non relevant. The query was then modified as presented section 3.3 and the new query was evaluated.
All the results are presented as in TREC, using the SMART methodology [41] and the top evaluation method.

Automatic feedback results and discussion

Tables from 2 to 4 report the results obtained on the Ad-hoc collection whereas tables from 5 to 7 report the results on the French collection.
The results show that on both the Ad-hoc and the French collections the automatic (blind) back-propagation feedback is efficient. using the Ad-hoc collection, 209 additional relevant documents were retrieved i.e. about a 15% increase (Table2). Moreover, Table 3 and 4 show that the precision at any recall point and at any top document point is improved. The average improvement is about 18% (from 0.19 to 0.23) for average precision at the 11 recall points and 11.41% for the exact precision.
Comparable results have been obtained using the French collection. Improvements are noted in both average precision (about 10%) and exact precision (about 4%).
In both experiments, the higher difference between the automatic feedback and the initial search are obtained at the recall points 0.8, 0.9 and 1 (more than +50% in the Ad-hoc case and from +29% to −50% in the French case). In fact at these points, the absolute difference is minimal (e.g. from a precision of 0.0058 to 0.01). Thus the results obtained at lower recall points are much more significant.

Table 2. Ad-hoc - Automatic feedback vs single pass results - Total number of documents over all queries

	Initial search	Automatic feedback
Retrieved	25000	25000
Relevant	2712	2712
RelRet	1351	1560

Table 3. Ad-hoc - Automatic feedback vs single pass results - Interpolated Recall Precision Averages

	Initial search	Automatic feedback	% improvement
at 0.00	0.6540	0.7421	13.47%
at 0.10	0.4225	0.5361	26.89%
at 0.20	0.3344	0.4145	23.95%
at 0.30	0.2822	0.3365	19.24%
at 0.40	0.2137	0.2375	11.14%
at 0.50	0.1905	0.2004	5.20%
at 0.60	0.1188	0.1306	9.93%
at 0.70	0.0714	0.0905	26.75%
at 0.80	0.0317	0.0476	50.16%
at 0.90	0.0075	0.0111	48.00%
at 1.00	0.0058	0.0104	79.31%
avg prec.	0.1956	0.2303	17.74%

Table 4. Ad-hoc - Automatic feedback vs single pass results - Precision

	Initial search	Automatic feedback	% improvement
at 5	0.5120	0.5360	4.69%
at 10	0.4240	0.4680	10.38%
at 15	0.3653	0.4187	14.62%
at 20	0.3440	0.3920	13.95%
at 30	0.3120	0.3520	12.82%
at 100	0.2224	0.2428	9.17%
at 200	0.1532	0.1720	12.27%
at 500	0.0858	0.0994	15.85%
at 1000	0.0540	0.0624	15.56%
Exact	0.2400	0.2674	11.41%

Table 5. French - Automatic feedback vs single pass results - Total number of documents over all queries

	Initial search	Automatic feedback
Retrieved	20213	21000
Relevant	1239	1239
RelRet	1018	1110 (+9.04%)

Table 6. French - Automatic feedback vs single pass results - Interpolated Recall Precision Averages

	Initial search	Automatic feedback	% improvement
at 0.00	0.7441	0.7812	4.99%
at 0.10	0.6341	0.7040	11.02%
at 0.20	0.5569	0.6334	13.74%
at 0.30	0.5075	0.5579	9.93%
at 0.40	0.4287	0.4710	9.87%
at 0.50	0.3667	0.4313	17.62%
at 0.60	0.3367	0.3759	11.64%
at 0.70	0.2523	0.2999	18.87%
at 0.80	0.1982	0.2572	29.77%
at 0.90	0.1288	0.1183	-8.15%
at 1.00	0.0521	0.0244	-53.17%
Avg prec.	0.3680	0.4039	9.76%

Table 7. French - Automatic feedback vs single pass results - Precision

	Initial search	Automatic feedback	% improvement
at 5	0.6000	0.6095	1.58%
at 10	0.5524	0.5857	6.03%
at 15	0.4889	0.5365	9.74%
at 20	0.4548	0.5095	12.03%
at 30	0.4079	0.4460	9.34%
at 100	0.2548	0.2767	8.59%
at 200	0.1731	0.1936	11.84%
at 500	0.0865	0.0974	12.60%
at 1000	0.0485	0.0529	9.07%
Exact	0.3978	0.4141	4.10%

Additional results have been reported in [5], [6], [7], [8] with different sources. These include works on manual back-propagation feedback (in that case the relevance is exact relevance instead of blind relevance).

4 Combining AG and NN to improve query reformulation

4.1 Approach

The neural network approach has been shown to be efficient [8], [29] either using a single pass retrieval process or relevance back-propagation retrieval. Nevertheless the query reformulation methods used are based on the assumption that a query that partitions relevant and non relevant documents can be found. We argue that several versions of a query could have more success. With this i mind, we combine the neural network model with genetic algorithms.
Fig. 2, shows the model used when combining NN and GA. The NN model is used to evaluate a single query whereas the GA is used to generate a population of queries and to select the ones that best match the user's needs. The final system response is obtained by merging the results of the best queries. In the following subsections, we detail the way in which the GA works.

4.2 The GA model for query space exploration

The genetic algorithm attempts to make evolve, generation by generation, a population of queries towards those improving the outcome of the system.

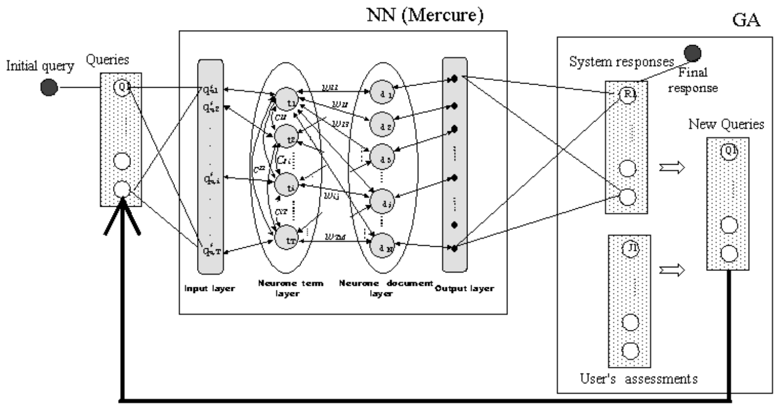

Fig. 2. IRS model based on the combination of NN and AG

The main characteristics of our model are its knowledge-based operators, instead of the blind operators generally used, and its niching technique.

Query individual and query population

In our approach, the genetic individual is a query. Each gene or chromosome corresponds to an indexing term or concept. The *locus* (the existence or the absence of a given gene) is represented by query term weight. The initial query population (set of queries), $(Pop^{(0)})$, contains the initial query and the relevant documents retrieved by this initial query. Thus, at the first iteration of the GA, a relevant document is considered as a query. The query population is renewed at each iteration of the GA by applying selection, crossover and mutation operators.

Fitness function

A $fitness$ is assigned to each query in the population. This fitness represents the query effectiveness during the retrieving stage. It is computed according to the relevance of the retrieved documents. The formula used is:

$$QFitness(Q_u^{(s)}) = \frac{\frac{1}{\|D_r\|} * \sum_{d_j \in D_r} Sim(d_j, Q_u^{(s)})}{\frac{1}{\|D_{nr}\|} * \sum_{d_j \in D_{nr}} Sim(d_j, Q_u^{(s)})}$$

where $Q_u^{(s)}$ is query u at the generation (s) of the GA, $\|D_r\|$ is the number of relevant documents retrieved through all generations, $\|D_{nr}\|$ the number of nonrelevant documents retrieved through all generations and $Sim(d_j, Q_u^{(s)})$ is the Jaccard function given as follows :

$$Sim(d_j, Q_u^{(s)}) = \frac{\sum_{i=1}^{T} q_{ui}^{(s)}.w_{ji}}{\sum_{i=1}^{T} {q_{ui}^{(s)}}^2 + \sum_{i=1}^{T} w_{ji}^2 - \sum_{i=1}^{T} q_{ui}^{(s)}.w_{ji}}$$

Thus the more a query retrieves relevant documents, the fitter the query is. This fitness function would favour the reproduction of queries which are close to relevant documents and far away from nonrelevant documents.
As mentioned previously, the document relevance is multi-modal in the sense that relevant documents corresponding to the same information need may be confined in different regions of the document space and therefore have some different vector components. However, usually, the GA evolves moving the query population towards an optimal query, which retrieves documents of similar representation. Thus, we adjust the fitness function in order to favour the construction of queries retrieving documents that are confined in different regions. Using the techniques of speciation in ecological theory [16], we define a **niche** as a set of queries that retrieve documents of similar genotype. It is defined as follows:

$$Niche(Q_u^{(s)}) = \{Q_v^{(s)} \ as \ Eucl\_Distance(Q_u^{(s)}, Q_v^{(s)}) \leq niching\_threshold\}$$

Here, $Q_u^{(s)}$ and $Q_v^{(s)}$ are two queries, $Eucl\_Distance(Q_u^{(s)}, Q_v^{(s)})$ is the Euclidean distance computed as follows:

$$Eucl\_Distance(Q_u^{(s)}, Q_v^{(s)}) = sqrt(\sum (q_{ui}^{(s)} - q_{vi}^{(s)})^2)$$

The $niching\_threshold$ is determined experimentally. The fitness function becomes:

$$Fitness(Q_u^{(s)}) = \frac{QFitness(Q_u^{(s)})}{||Niche(Q_u^{(s)})||}$$

Thus, the query fitness is proportional to the query similarity with relevant documents and inversely proportional to both the query similarity with nonrelevant documents and the number of queries in the same niche. This favours the selection and reproduction of queries in niches that are little populated. Therefore, new regions are explored according to the average fitness of the associated queries.
Finally, the query population is represented as follows:

$$Pop^{(s)} = \{(Q_u^{(s)}, Fitness(Q_u^{(s)})) \ as \ u = 1..Pop\_size\}$$

Where Pop_size corresponds to the size of the population.

Genetic operators

Some genetic operators defined in this work are not pure genetic operators. They have been adapted to take advantage of techniques developed in IR. For this reason the defined operators are qualified as knowledge based operators.

Selection. The selection consists of the reproduction of a new population, from the previous population, according to the individual fitness values. Our selection procedure is based on roulette wheel selection [16]. It consists in assigning to every individual of the population a number of copies in the next generation, proportional to the individual relative fitness. Then, one spin of the wheel selects a single individual each time until the population size is reached.

Crossover. In GA, the crossover is applied to a pair of individuals which are selected according to the crossover probability, noted P_c. These individuals exchange part of their genes delimited by randomly selected crossing points. This process leads either to one or two new individuals ; the initial individuals are called the *parents* of the newly generated individuals. In our approach, the GA uses the individual queries according to their niche in order to continue the exploration in different regions of the document space. The goal is to reach relevant documents with different representations. We define two crossover operators: a crossover based on query term weight which is used for the queries belonging to the same niche; and the crossover based on term co-occurrence in the relevant documents which is used for the queries from different niches.

Crossover based on term weight
This crossover does not use a crossing point, it modifies the term weights according to their distribution in the relevant and nonrelevant documents. Let us consider $Q_u^{(s)}$ and $Q_v^{(s)}$ two individuals (queries) selected for crossover. The result is the new individual $Q_p^{(s)}$ defined as follows :

$$\left.\begin{array}{l} Q_u^{(s)}(q_{u1}^{(s)}, q_{u2}^{(s)}, \ldots, q_{uT}^{(s)}) \\ Q_v^{(s)}(q_{v1}^{(s)}, q_{v2}^{(s)}, \ldots, q_{vT}^{(s)}) \end{array}\right\} \longrightarrow Q_p^{(s+1)}(q_{p1}^{(s+1)}, q_{p2}^{(s+1)}, \ldots, q_{pT}^{(s+1)})$$

$$q_{pi}^{(s)} = \begin{cases} Max(q_{ui}^{(s)}, q_{vi}^{(s)}) \ if \ weight(t_i, D_r^{(s)}) \geq weight(t_i, D_{nr}^{(s)}) \\ Min(q_{ui}^{(s)}, q_{vi}^{(s)}) \ otherwise \end{cases}$$

Where the weight of term t_i in a set of documents D is defined by $weight(t_i, D) = \sum_{d_j \in D} w_{ji}$ and $D_r^{(s)}$ is the set of relevant documents retrieved by the population $Pop^{(s)}$ and $D_{nr}^{(s)}$ is set of nonrelevant documents retrieved by the $Pop^{(s)}$.

In other words, if the weight of term t_i in the set of relevant documents is higher than its weight in the set of nonrelevant documents, this term is retained as significant and the highest weight among ($q_{ui}^{(s)}$, $q_{vi}^{(s)}$) is assigned to it in the new query $Q_p^{(s+1)}$. Otherwise, it is the lowest weight which is assigned to the term in the new query. This operator is applied to the queries from the same niche. Indeed, the goal of this operator is to adjust the descriptions of the new queries in order to recall further relevant documents in the same region of the document space.

Crossover based on term co-occurrence

In GA, the crossover operator consists of exchanging a part of the genes between the selected individuals. The second crossover operator we define uses the same idea, but it is based on the term co-occurrence in the relevant documents.

Let us consider $Q_u^{(s)}$ and $Q_v^{(s)}$ two individuals selected for crossover with k as the crossing point and $Q_u^{(s+1)}$ and $Q_v^{(s+1)}$ as the new individuals resulting from the crossover. The new individuals copy the first k terms from their parents and the remaining terms are built as follows: each term among the non-copied terms of one parent, is replaced by the most co-occurrent term in the other parent. Formally, this operator is defined as :

$$\left.\begin{array}{l} Q_u^{(s)}(q_{u1}^{(s)}, q_{u2}^{(s)}, ..., q_{uT}^{(s)}) \\ Q_v^{(s)}(q_{v1}^{(s)}, q_{v2}^{(s)}, ..., q_{vT}^{(s)}) \end{array}\right\} \longrightarrow \begin{array}{l} Q_u^{(s+1)}(q_{u1}^{\prime(s+1)}, q_{u2}^{\prime(s+1)}, ..., q_{uT}^{\prime(s+1)}) \\ Q_v^{(s+1)}(q_{v1}^{\prime(s+1)}, q_{v2}^{\prime(s+1)}, ..., q_{vT}^{\prime(s+1)}) \end{array}$$

where

$$\begin{array}{l} q_{ui}^{\prime(s+1)} = q_{ui}^{(s)} \quad if \ i \leq k \\ otherwise \ for \ i = k+1 \ to \ T \\ \qquad lookup \ t_l \ in \ Q_v^{(s)} \ as \ l > k \ \wedge \ max^D(C_{t_i,t_l}, Q_v^{(s)}) \\ \qquad q'_{ul} = q_{vl}^{(s)} \\ \qquad endfor \end{array}$$

$Q_v^{(s+1)}$ is built in same way. $max^D(C_{t_i,t_l}, Q_v^{(s)})$ returns the term t_l of $Q_v^{(s)}$ that has the highest co-occurrence weight with t_i in the document set D.
The co-occurrence of two terms in a set of documents is defined as follows :

$$C_{t_i,t_l} = \frac{\sum_{d_j \in D_r} w_{ji} * w_{jl}}{\sum_{d_j \in D_r} w_{ji}^2 + \sum_{d_j \in D_r} w_{jl}^2 - \sum_{d_j \in D_r} w_{ji} * w_{jl}}$$

This operator is applied to the queries belonging to different niches. It allows the exchange of a section of information between two queries retrieving relevant documents confined in different regions of the document space. This would enable exploration of a new region in the document space.

Mutation based on term relevance. In GA, the mutation is an unary operator applied to an individual with a given mutation probability P_m. Each gene of this individual can be mutated (modified) when $(r < P_m)$ where r is a random value from the range of [0..1]. In our approach, the mutation operator explores the terms occurring in the relevant documents in order to adjust the corresponding gene values in the query selected for the mutation. Let us consider $Q_u^{(s)}$ as the selected individual query and $L^{(s)}$ as the set of terms from $D_r^{(s)}$, the relevant documents retrieved in the previous generation of the GA. The mutation alters the genes of the selected individual on the basis of the $L^{(s)}$ terms and on the probability P_m. The $L^{(s)}$ terms are sorted according to a score value calculated as follows:

$$Score(t_i) = \frac{\sum_{d_j \in D_r^{(s)}} w_{ji}}{\|D_r^{(s)}\|}$$

This score is used to limit the number of terms that can be used for the mutation process [40]. The mutation operation is done as follows :

for each term t_i *in* $L^{(s)}$
  if (random(p) $< P_m$*) then* /* mutate the gene t_i
  $q_{ui}^{(s)} = average(Q_i^{(s)}) - \delta$ /* modify the weight of t_i in $Q_u^{(s)}$
  endif
endfor

Where P_m is the mutation probability, $random(p)$ generates a random number p in the range of [0..1]. The average function is computed as follows :

$$average(Q_u^{(s)}) = \frac{\sum_j^T q_{ui}^{(s)}}{n_{q_{ui}^{(s)}}}$$

where $n_{q_{ui}^{(s)}}$ is number of $q_{ui}^{(s)} \neq 0$ in $Q_u^{(s)}$. δ is a parameter used to control the average value (we used $\delta = 0$).

4.3 Experiments and results

The experiments were carried out on a subset of AP88 collection (15000 documents) and topics 1-10 issued from the TREC6 programme [41]. The initial queries were built from the *Title*, the *Description* and the *Narrative* fields. Two experiments have been performed. The first measures the effectiveness of the genetic model and the second compares the GA to the relevance back-propagation process. Both experiments have been done in an interactive environment with multiple iterations. In this context, an initial search was done, the user marked the relevant documents from the top 15

retrieved documents. Then, depending on the experiment, either GA or relevance back-propagation, the system automatically builds a new query (or a population of queries) and displays 15 new documents, never seen by the user. The user selects the relevant ones and so on, until a fixed number of iterations. At each iteration of the search, the number of relevant retrieved documents in the top 15 was reported. Thus, the effectiveness of the experimented method was evaluated by considering the number of relevant documents retrieved at each iteration of the search and the total cumulative number of relevant retrieved documents at that iteration.
Notice that the GA experiment was performed using the niching technique. Table 8 shows genetic operators combination defined for this experiment. The

Table 8. Genetic operators combination used in the niching technique

Niching operator	
Queries from the same niche	Queries from different niches
Crossover based on term weight then mutation	Crossover based on term co-occurrence then mutation

niching operator is applied as follows:

- When two queries are selected for transformations, if they belong to the same niche, crossover based on term weight is applied to these queries, then mutation is applied to the generated query.
- If the selected queries are from different niches, crossover based on term co-occurrence is applied and then mutation is applied to each new generated query.

For these experiments the following parameters have been used $P_c = 0.7$, $P_m = 0.07$, $niching\_threshold = 0.1$, $Po\_size = 5$. These parameters have been tuned in a previous experiment [40].

GA experiment and result

This experiment measures the effectiveness of our GA by comparing the results obtained with GA to the one obtained with no GA. Thus, let us consider that the initial search (iteration 0) is done; the user selects the relevant documents from the top 15 shown by the system. At iteration 1, we have:

- With GA, the system runs the GA and presents to the user a new top 15 documents never seen before. The user selects the relevant ones.
- With no GA, the system selects the next 15 documents from the list retrieved at the previous iteration (iteration 0),
- and so on.

To evaluate the effect of the GA, we compare the number of relevant documents retrieved by the GA at a given iteration to the number of relevant documents resulting from just keeping the next 15 documents from the previous iteration results.

Table 9. Effect of the GA operator

Iteration	0	1	2	3	4	5	6	7	8
With GA									
Relevant ret. per iteration	43	30	4	2	13	1	3	5	1
Cumulative rel. ret.	43	73	77	79	92	93	96	101	102
Percentage cum. rel./(total rel.)	30%	50%	53%	54%	63%	64%	66%	70%	70%
Queries retrieving all rel. doc.	1	2	3	3	5	5	5	5	5
With no GA									
Relevant ret. per iteration	43	12	5	5	4	2	4	1	2
Cumulative rel. ret.	43	55	60	65	69	71	75	76	78
Percentage cum. rel./(total rel.)	30%	38%	41%	45%	48%	49%	52%	52%	54%
Queries retrieving all rel. doc.	1	2	2	2	2	2	2	2	2

Table 9 compares the results with GA and with no GA. The columns numbered from 0 to 8 represent the iteration number; the iteration 0 corresponds to the initial search. The "*relevant ret. per iteration*" lines list the number of relevant documents retrieved at a given iteration (totalled across all the queries). The "*Cumulative rel. ret.*" lines give the cumulative total number of relevant documents retrieved at a given iteration. The "*Percentage cum. rel.(total rel.)*" lines report the percentage between the cumulative number and the total known relevant documents. The "*Queries retrieving all rel. doc.*" lines list the total number of queries retrieving all relevant documents.

With GA, the total number of relevant retrieved documents is much higher in the first iterations than with no GA. No GA continues to get relevant documents in the next iterations but the cumulative is still lower comparing to the GA. After 8 iterations we obtained 102 relevant documents with GA and 78 with no GA. Moreover, it can be seen that, with GA, five queries retrieve all relevant documents since the fourth iteration of the GA, whereas with no GA only two queries retrieve all their relevant documents.

GA and relevance back-propagation comparison

Table 10 compares the GA with the relevance back-propagation method.

Table 10. Comparison of GA with relevance back-propagation

Using GA									
Iteration	0	1	2	3	4	5	6	7	8
Relevant ret. per iteration	43	30	4	2	13	1	3	5	1
Cumulative rel. ret.	43	73	77	79	92	93	96	101	102
Percentage cum. rel./(total rel.)	30%	50%	53%	54%	63%	64%	66%	70%	70%
Queries retrieving all rel. doc.	1	2	3	3	5	5	5	5	5
relevance back-propagation									
Relevant ret. per iteration	43	17	7	8	3	4	5	6	3
Cumulative rel. ret.	43	60	67	75	78	82	87	93	96
Percentage cum. rel./(total rel.)	30%	41%	46%	52%	54%	57%	60%	64%	66%
Queries retrieving all rel. doc.	1	2	2	2	2	2	2	2	2

It can be seen that the total number of relevant documents is slightly higher with GA than with relevance back-propagationand it is so after the first iteration of the GA. But the relevance back-propagation continues to select new relevant documents at each search iteration. Moreover, at some iterations this number is higher than with GA. This can be explained by the fact that a lot of these new documents have been already selected by the GA. The interesting result of this experiment comes from the number of queries retrieving all their relevant documents. Indeed, with GA, after the second iteration of the search, 3 queries then 5 queries from 10 retrieve all their relevant documents, whereas in relevance back-propagation only two queries retrieve all their relevant documents. This difference might come from the use of the niching technique. Indeed the GA operators are built in order to explore several regions of document space simultaneously. These regions are explored by two suitable crossover operators. One, based on term weight, allows the building of the best query from those selecting relevant documents in the same region. The second, based on co-occurrence, tries to combine queries selecting documents from different regions of the document space.

Conclusion

In this paper, we described a neural model to implement an associative information retrieval, named Mercure. This neural model which takes advantage of recent advances in IR, allows a single pass search. In addition this model implements a query modification based on relevance back-propagation. This system has been experimented in manual and automatic environments using different collections of data. In this paper, we have focused on auto-

matic query reformulation. Results obtained using TREC6 sub-collections have been discussed. These results set Mercure among the best systems that took part in an automatic Adhoc TREC6 experiment.
The other main contribution of this paper is to combine this neural model with a GA in order to improve retrieval. The GA generates a query population that explores different parts of the document space. We have shown that the GA for IR improves the results of the search in the interactive environment, even though the random aspect of the GA is still a big question. The comparison with the back-propagation method shows that the GA is more effective. Nevertheless, these results should be taken carefully as the test collection used was small.
We also notice that the number of generations and individuals of these experiments is very small compared to the GA theory, further experiments will be undertaken.
This first investigation makes us optimistic concerning future works in this direction, particularly when the documents are heterogeneous. In this case the use of several queries, each one with specific representation could be more efficient than a single one.
Our future works will use this approach on the English TREC data [41] in several TREC tasks (Adhoc and the interactive tracks). Therefore collection dependent parameters could be well-tuned and the approach better evaluated.

References

1. Bein, J., Smolensky, P. (1988): Application of the interactive model to document retrieval. Tech. Rep. CU-CS-405-88, Dept. of Computer Sciences, Univ. of Colorado, Boulder, Colorado.
2. Belew, R.K (1989): Adaptive Information Retrieval : Using a connectionist representation to retrieve and learn about documents. In ACM/SIGIR International Conference on Research and development in Information Retrieval, 11–20.
3. Belew, R.K., McInerney, J., Schraudolf, N, (1990): Evolving Networks: using the Genetic Algorithm with Connectionist Learning, CSE TR, CS90-174.
4. Boughanem, M., Soule-Dupuy, C. (1994): Relevance Feedback and Neural Networks. In 4th RIAO International Conference on Computer-Assisted Information Retrieval.
5. Boughanem, M., Soule-Dupuy, C. (1997): Query modification based on relevance back-propagation. In 5th RIAO International Conference on Computer-Assisted Information Retrieval, 469–487.
6. Boughanem, M., Soule-Dupuy, C. (1997): Mercure at TREC6. In Harman D.K. (Ed.), NIST SP, 6th International Conference on Text Retrieval TREC6.
7. Boughanem, M., Dkaki T., Mothe, J., Soule-Dupuy, C. (1998): Mercure at TREC7. In Harman D.K. (Ed.), NIST SP, 7th International Conference on Text Retrieval TREC7.
8. Boughanem, M., Chrisment, C., Soule-Dupuy, C. (1999): Query modification based on relevance back-propagation in an ad hoc environment, to be published in Information Processing Managment.

9. Brachman, R.J., MacGuiness, D.L. (1988): Knowledge representation, connectionism, and conceptual retrieval. In 11th international conference on R and D in IR Y. Chiaramella, Ed. PUG.
10. Chang, Y.K., Cirillo, G.C., Razon, J. (1971): Evaluation of feedback retrieval using modified freezing, residual collection and test and control groups, 355–370. In The Smart Retrieval System : Experiments in Automatic Document Processing. Prentice-Hall Inc.
11. Chen, H. (1995): Machine Learning for Information Retrieval: Neural Networks, Symbolic Learning and Genetic Algorithms. Journal of American Society for Information Science, **46(3)**, 194–216.
12. Chen, H., Lynch, J., Basu, K. , Ng, T. (1993): Generating, integrating, and activating thesauri for concept based document retrieval, IEEE Expert, Special series of Artificial Intelligence in Text-Based Information Systems, **8(2)**, 25–34.
13. Crestani, F. (1995): Implementation and evaluation of relevance feedback device based on neural networks, From natural to artificial neural computation : international workshop on artificial neural networks, J. Mira and J. Cabastany (Ed.) Springer Verlag, 597–604.
14. Le Cun, Y. (1985): Une procédure d'apprentissage pour réseau seuil asymtrique, Cognitiva Conference A la Frontière de l'Intelligence Artificielle Des Sciences de la Connaissance, des Neurosciences.
15. Doszkocs, T.E., Tamas, E., Reggia, J., Xia Lin (1990): Connectionist Models and Information Retrieval. Information Science and Technology. Williams, Martha E. (Ed.), Elsevier Science Publishers, **25**, 209–260.
16. Goldberg, D. E. (1994): Algorithmes génétiques. Exploration, optimisation et apprentissage automatique, Edition Addison-Wesley, France.
17. Gordon M.D. (1988): Probabilistic and Genetic Algorithms for Document Retrieval. Communications of the ACM, **31(10)**,1208–1218.
18. Harman, D.K. (1992): Relevance Feedback Revisited. ACM/SIGIR International Conference on Research and development in Information Retrieval, 1–10.
19. Harp, S., Samad, T., Guha, A. (1989): Towards the Genetic Synthesis of Neural Networks. In 3rd International Conference on Genetic Algorithms. IEEE Transactions on Knowledge and data Engineering, **5(1)**, 29–40.
20. Harper, D.J., Van Rijsbergen, C.V. (1978): An evaluation of feedback in document retrieval using co-occurrence data. Journal of Documentation, **34(4)**, 189–216.
21. Holland, J. (1975): Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor.
22. Hopfield, J.J. (1982): Neural Networks and Physical Systems with Emergent Collective Computational Abilities, Proceedings of the National Academy of Science, USA, **79**, 2554–2558.
23. Ide, E. (1971): New Experiments in Relevance Feedback. In Salton G. (Ed.), The SMART Retrieval System, Englewood Cliffs, NJ: Prentice-HaIl, 337–354.
24. Kitano, H. (1990): Empirical studies on the speed of convergence of neural network training using genetic algorithms, 8th National Conference on Artificial Intelligence (AAAI-90), 789–795.
25. Kohonen, T. (1989): Self-Organization and Associative Memory, 3rd Edition, Springer Verlag, Berlin.
26. Kraft, D.H., Petry, F.E., Buckles, B.P., Sadasivan, T. (1994). The Use of Genetic Programming to Build Queries for Information Retrieval. IEEE Symposium on Evolutional Conference.

27. Koza, J.R. (1991): A Hierarchical Approach to Learning the Boolean Multiplexer Function. In Rawlins, G., Foundation of Genetic Algorithms, 324–353.
28. Kwok, K.L. (1995): A Network Approach to Probabilistic Information Retrieval. ACM Transactions on Information Systems, **13(3)**, 324–353.
29. Kwok, K.L. (1997): TREC-6 English and Chinese Retrieval Experiments using PIRCS. In Harman D.K. (Ed.), NIST SP, 6th International Conference on Text Retrieval, TREC6.
30. Mothe, J. (1994): Search mechanisms using a neural network-Comparison with the vector space model. In 4th RIAO Intelligent Multimedia Information Retrieval Systems and Management, 275–294.
31. Mozer, M.C. (1984): Inductive information retrieval using parallel distributed computation, Technical report, Inst. Cognitive Science, UCSD, La Jolla, CA.
32. Poincot, P., Lesteven, S., Murtagh, F. (1998): A spatial user interface to the astronomical literature, Astronomy and Astrophysics Supplement, **130**, 183-191.
33. Roberston, S.E. et al. (1998): Okapi at TREC7, In 7th International Conference on Text Retrieval TREC7.
34. Rocchio, J. (1971): Relevance Feedback in Information Retrieval. In G. Salton (Ed.), The Smart System experiments in Automatic Document Processing, 313–323. Prentice-Hill, Inc., Englewood Cliffs, NJ.
35. Rumelhart, D.E., Hinton, G.E., Williams, R. (1986): Learning internal representations by error propagation. In Rumelhart, J. McClelland (Ed.), Parallel Distributed Proceedings,1(D). The MIT Press.
36. Rumelhart, D.E., Widrow, B., Lehr, M.A. (1994): The Basic Idea in Neural Networks. Communication of the ACM, **37(3)**, 97–92.
37. Salton, G. (1970): Evaluation problems in interactive information retrieval. Information Storage and Retrieval, **6(1)**, 29–44.
38. Salton, G. (1971): The SMART Retrieval System. Englewood Cliffs, NJ: Prentice-HiII.
39. Salton, G., Buckley C., (1988): On the use of spreading activation methods in automatic information retrieval, ACM/SIGIR International Conference on Research and development in Information Retrieval, 147–160.
40. Tamine, L. (1997): Reformulation automatique de requêtes basée sur l'algorithmique génétique, Inforsid : Informatique des organisations et systèmes d'information et de décision, 643–662.
41. TREC6 (1998): Harman D.K. (Ed.), NIST SP, 6th International Conference on Text Retrieval TREC6.
42. Van Rijsbergen, C.J. (1986): A New Theorithecal Framework For Information Retrieval, ACM/SIGIR International Conference on Research and development in Information Retrieval.
43. Wilkinson, R., Hingston, P. (1991): Using the Cosine Measure in a Neural Network for Document Retrieval, ACM/SIGIR International Conference on Research and development in Information Retrieval, 202–210.
44. Wong, S.K.M., Cai, Y.J., Yao, Y.Y. (1993): Computation of term associations by a neural network, ACM/SIGIR International Conference on Research and development in Information Retrieval, 107–115.
45. Yang J-J., Korfhage R. (1992): Query Modification Using Genetic Algorithms in Vector Space Models, Technical Report LIS045/I592001.