Are Many Heads Better Than One—On Combining Information from Multiple Internet Sources

Jakub Stadnik[1], Maria Ganzha[2], and Marcin Paprzycki[2]

[1] Warsaw University of Technology, Warsaw, Poland
[2] Systems Research Institute Polish Academy of Sciences, Warsaw, Poland
Maria.Ganzha,Marcin.Paprzycki@ibspan.waw.pl

Summary. In this paper we look into three approaches, based on: *Game Theory, Auction* and *Consensus* methods, to combine information from multiple sources. As originally introduced, they are conceptualized using an agent metaphor and implemented using a JADE agent platform. Preliminary performance comparison completes the presentation.

1 Introduction

Since different Internet search engines produce different results for the same query, we can say that they "see" the world differently. The question then arises: how to combine answers from different sources in such a way that the obtained answer would be "better" than when using only a single source? What suggest combining "advice" from multiple sources is a standard situation, when a panel of experts is used to address a problem. Combining multiple suggestions can be achieved, among others, utilizing a *Consensus method* [5, 2, 4], *Game Theory* and *Auctions* [8, 7]. These approaches have been originally proposed as based on software agents. While this is somewhat spurious (proposed functionalities can be achieved without agents), we follow predecessors and use JADE agent platform to implement combining information from multiple Internet sources.

This note is organized as follows. In the next section we introduce the three approaches to information joining. We follow with preliminary experimental results and their analysis.

2 System setup

Proposed system can be split into two main parts: *Client Module* (the interface) and the *Main Agent* (system manager). *Client Module* is responsible

for interacting with the end-user. The *Main Agent* receives requests from the *Client Module* and manages agents for information retrieval and combining results.

At the beginning there is only the *Main Agent* (*MA*) waits for a query and the processing algorithm of choice. When the input is received the MA creates as many *Search Agents* (*SA*) as there are selected search engines (user can specify how many and/or which search engines to include, or a default number will be used). Each *SA* is assigned a different search engine. SA's query the database using the content of the query and information about selected algorithm, to retrieve the weights set, which are used during data processing. Weights are the ranks of search engines; computed for a given algorithm based on previous results for a given query. Their belong to interval $(0, 1)$ depending on how the algorithm evaluated result set of a particular search engine. If engine performed badly—results were not satisfactory in the sense of the algorithm; it is assigned a smaller weight than the engine results of which were considered as better ones (in the sense of the algorithm). If this is the first time for a given query, all ranks are set to 1. Those weights are used during the ranking processes to "boost" URLs originating from engines, which contributed better results in previous runs for that query and algorithm.

Next, *SA*s execute the query and return their results to the *MA*, which processes them according to the selected algorithm. When the processing is finished, the MA sends the final results to the web application to be displayed. Note that if the algorithm was able to find the best result, the result list is displayed and knowledge base is updated instantaneously. The search engine which yielded the top result is ranked as the best and other engines are ranked according to how close they were to this engine. If, however, algorithm was not able to yield a "satisfactory" answer; application displays "an answer" and an option to provide feedback (subjective evaluation performed by the user). Feedback (if received) ranks search engines. In the knowledge base we store—for each query, search engine and method of answer processing, and the engine rank.

2.1 Common algorithms

There is a number of algorithms that are used by multiple approaches. First, the algorithm for the initial URL ranking. This initial ranking is performed before the *Game Theory* and *Auction* methods (not the *Consensus* method) start their computations. Its purpose is to calculate confidence values of *Search Agents* about retrieved URLs. In general, the confidence value is calculated as follows: $|$*result set of agent*$| -$ *position of the URL in resulting set.* However, the *Game Theory* and the *Auction* methods require that each result set contains the same URLs (in any order). If this is not the case they break, since comparison of ranks certain URLs cannot be performed. Therefore, this algorithm updates the result sets with missing URLs. It also determines if the main computational parts of the two approaches can even be performed. The

rule is as follows: if for all pairs of result sets A and B the $A \cap B = \varnothing$ then the main part of the *Game Theory* and *Auction* cannot start. Thus, if a result set has no common URL with any other result set it is removed from the process as being not suitable for the algorithms which require every URL to be in every result set. The pseudo-code of the algorithm is as follows.

URL ranking algorithm for *Game Theory* and *Auction* methods

Input Map of results $\langle a^i, r^i \rangle$ provided by

$$m$$

Search Agents—each in the form $r^i = \langle U_1^i, U_2^i, \ldots, U_n^i \rangle$, where U_j^i, $j = 1, \ldots, n$, are URLs.
Output Map containing URL ranking.
BEGIN

1. for each agent in the map:
   - check if other agents result sets contain any of the URLs of the agent
   - construct matrix representing how many URLs of the agent are contained in the each result set of other agents
2. check if each agent has at least one common URL with another if not—remove it from further processing
3. if result set of every agent is disjoint with each result set of every other agent—stop algorithm
4. for each agent in map:
   - for each URL in agent result set:
     - rank the URL as follows:

$$rank(U^i) = (|r| - i) \times weight(r)$$

       , where i is a position of URL in r
     - find agents which result set does not contain the URL, update their rankings:

$$rank(U^i) = 1.0 \times weight(r)$$

       (weights calculation—listing 2.1)
5. return ranking

END

Weights calculation for *Game Theory* and *Auction* methods

Weights calculation is performed after *Game Theory* and *Auction* methods finish their main negotiation parts. This algorithm is to rank the search engines according to how the URLs from a given engine were evaluated (placed) in the final answer. The topmost URL is chosen to be the feedback result and other result sets are weighted accordingly to the number of URLs overlapping with the result set which provided the URL.

Input: Result from feedback; initial result sets
Output: Map of weights with corresponding agents
BEGIN

1. find the agent whose result set contains the "best" result, set his weight to 1

2. for all other agents:

$$find\, d(r^{(i)}, r^w)$$

$$W[i] = \frac{|r^{(i)}| - d(r^{(i)}, r^w)}{|r^{(i)}}$$

where $d(r^{(i)}, r^w)$ is the number of different URLs between the result set of agent i and the "winner" agent

3. return weights

END

After this part is finished, ranks are stored in the knowledge base for further use. These weights are used as follows: when issuing the query for the second time for a particular method (*Game theory* or *Auction* in this case) the weight of the result set is used to decrease the rank of the URL which originates from this result set. The rank of such URL is multiplied by this weight. Thus, if it is less than 1 it is being decreased. This process gives handicap to URLs which are returned by the search engines with low weights—those contributed "not so good" results for a particular query. If the weight is equal to zero the rank is multiplied by 0.01 (to still keep it in "in the game").

3 Three main algorithms

3.1 Game Theory

This approach was used in the *NeurAge* system [8, 7, 1]. In its modified version, instead of voting for certain "classes of data," agents vote for URLs retrieved by search engines. Confidence values from the original algorithm have been replaced by URL ranks (obtained after above-described the pre-processing). Furthermore, in the original algorithm agents delivered a single "data class" as the answer. However, in Internet searching multiple, ranked responses are expected. Therefore, in the adapted approach, 10 "best" URLs are returned. Here, we utilize an iterative approach, where each iteration starts the selection process from the beginning, without previously selected URLs. This modification does not violate main assumptions of the algorithm [9].In general, a "game" consists of set of players, set of moves (strategies) and specification of payoffs for each combination of moves. In a normal form the game that is defined as follows:

There is a finite set P of players, which we label $\{1, 2, ..., m\}$. Each player k has finite number of pure strategies (moves) $S_k = \{1, 2, ..., n_k\}$ A pure strategy profile is an association of strategies to players, that is m-tuple $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m)$ such that $\sigma_1 \in S_1, \quad \sigma_2 \in S_2, \ldots, \sigma_m \in S_m$ Let strategy profiles be denoted by Σ A payoff function is a function $F : \Sigma \rightarrow \Re$ which intended representation is the award given to a single player as the outcome of the game. Accordingly to specify a game the payoff function has to be specified for each player in the player set $P = \{1, 2, \ldots, m\}$.

Definition 1. *A game in normal form is a structure (P, S, F) Where $P = \{1, 2, ..., m\}$ is a set of players, $S = (S_1, S_2, \ldots, S_m)$ is a mtuple of pure strategy sets, one for each player and $F = (F_1, F_2, \ldots, F_m)$ is a m-tuple of payoff.*

In the game considered here, components are as follows: (a) players are agents, (b) possible moves are to change or to keep the URL, (c) payoffs for those moves are defined as a 2×2 matrix. Each agent is assigned two values: for the keeping the URL and for changing it. Those values may or may not change each round of the game, depending on the outcome of the previous round.

At the beginning of the process, results obtained by the *Manager Agent* from *Search Agents* are filtered, ranked and updated (see above). The URL ranking represents confidence in a specific URL. In each round of the game two agents with highest ranked URLs have two possibilities: to keep their answer or to change it. If the keep action has higher value than the change action, the agent will be assigned the action to keep its URL for the next round. If, however, the agent is assigned the action to change its URL and the second agent is assigned the action to keep its URL, the latter is considered a *winner* of the round and the former is considered a *loser*—it and its result set are discarded from further considerations. Then the next round starts (without the agent and its result set; removed in previous round) and so on, until there is only one agent and its top URL is the winner. Process is then repeated, without the URL that was selected in the previous "big" round (this URL is removed from all result sets; recall that all sets have all URL's included; see above). Game continues until 10 (best) URLs are selected.

3.2 Auction-based approach

Auction-based approach was originally used in the *NeurAge* system [7], and was adapted to return 10 distinct URLs (rather than a single result). In each round of the auction each agent has its "product" (URL) assigned. Afterward, the "cost" for each assigned URL is calculated. Costs are compared and agent with the highest cost is considered to be a loser. Afterward, the confidence values for selected URLs are updated by subtracting the cost from their value. Henceforth, the next round takes place. If the agent that was marked before as a loser, loses again, it and its result set are discarded from further auctions. After removal of a twice-looser, process enters the next round, and continues until a single agent remains with its selected answer. This process is repeated 10 times and after each round the URL that was just selected, is removed from result-sets of all agents.

Here, we present an example of flow of one round of the *Auction* method:

Input: Map containing URL rankings.
Output: 10 URLs.
BEGIN
1. repeat until there are 10 URLs in answer list
2. repeat until one agent remains
3. find highest ranked URLs for all agents and pair them $(A^{(i)}, U^{(i)})$
4. calculate costs for each agent:

$$cost(A^{(i)}) = \frac{\sum_{i=1, i \neq j}^{m} (rank(A^{(i)}, U^{(i)}) - rank(A^{(i)}, U^{(j)}))}{10}$$

where $U^{(i)}$ is URL from pair $(A^{(i)}, U^{(i)})$ (highest ranked URL for agent $A^{(i)}$ and $U^{(i)}$ is a highest ranked URL for agent $A^{(i)}$

5. find agent with highest cost—a loser; it may happen that all agents have the same costs—if it occurs twice the agent which is assigned the URL initially ranked as the lowest is considered a loser and thus removed from further negotiation, if it is so, go to 7.
6. if the agent is a loser twice in a row remove it from further auctions
7. update URL rankings for all agents as follows:

$$rank(A^{(i)}, U^{(i)}) = rank(A^{(i)}, U^{(i)}) - cost(A^{(i)})$$

where the pair $(A^{(i)}, U^{(i)})$ is found at the beginning; at this point the winning URL can be changed

8. add URL to the answer list
9. remove the URL from all answer sets

3.3 Consensus method

The *Consensus* method was used previously in the AGWI system [5, 3, 6, 2, 4]. Its aim is to combine a set of answers into a final joint answer. The difference in the modified approach are as follows. The algorithm for measuring distances between result sets was adapted (to use the Levenshtein method). Furthermore, in the AGWI system there were more search engines than there were Search Agents (and thus only some of them were selected to be used). Here, there are as many Search Agents as there search engines.

In the *Consensus method*, result sets are evaluated and a combined result set (without repeating URLs) is created. Next, for each URL its average position in all result sets is calculated. In what follows, the combined result sets are sorted according to the average position of each URL. Then the consensus answer is found and its consistency checked. Before performing the calculation, however, the result sets and consensus are normalized; only a specific number of top URLs are incorporated into the answer. This number is of size of the smallest non-zero result set. To check consistency of the consensus answer, average of distances between result sets and average of distances between each result set and the consensus answer are evaluated. If the average of distances is bigger than average of distances of result sets and the consensus, then consensus answer is consistent; if not, the consensus answer is not consistent. The following listing presents the pseudo code of algorithm for finding the consensus answer.

Input: Map of results provided by m Search Agents. Map containing weights for result sets.
Output: *Consensus* answer.
BEGIN

1. create set ***URLS*** from all URLs from all result sets (without repetitions)
2. for each $U \in$ ***URLS***
   - create array $<t_1, t_2, \ldots, t_n>$, where t_i is position on which $\boldsymbol{U}$ appears in $r^{(i)}$;
   - if $\boldsymbol{U}$ does not appear in $r^{(i)}$ then set t_i as the length of the longest ranking increased by 1
   - divide each t_i by $weight(r^{(i)})$; if $weight(r^{(i)}) = 0$ divide by 0.01
   - calculate average $t(U)$ of values $(t_1, t_2, \ldots, t_n)$
3. consensus answer is obtained by ordering elements of according to values

END

Having checked the consistency of the result set, the algorithm decides on the next step. When the consistency is low, the answer containing all results is returned and feedback is requested. If the consistency is high, 10 first URLs from the consensus answer are presented.

Depending on the outcome of the consistency check the different entry point is used for the weight calculation algorithm. If the consistency of the consensus is high, agent whose result set has the smallest distance to the consensus is selected as the agent whose weight will be equal to 1 and the algorithm in following listing does not require the feedback URL as an input—thus, step 1 is omitted. If the consistency is low, the first step of the algorithm must be performed to find the "winner" agent.

Input: Result from feedback; initial result sets
Output: Map of weights with corresponding agents
BEGIN
1. find the agent whose result set contains the best result from feedback, set his weight to 1
2. for all other agents:

$$find\, d(r^{(i)}, C)$$
$$W[i] = \frac{|r^{(i)}| - d(r^{(i)}, C)}{|r^{(i)}}$$

where $d(r^{(i)}, C)$ is the the Levenshtein distance
3. return weights

END

4 Initial experimental results

In our initial set of experiments three queries were issued for the testing purposes: "`consensus decision making`", "`consensus decision making for conflict solving`", and "`is consensus decision making for conflict solving good enough or maybe Game theory or auction is better`". The idea was to take three queries which are related to the same topic; however first was to be simple, second intermediately complex, and third was to be very complex, while retaining coherence. There were 5 search engines queried. Four of them were English-language-based: *Google, Ask.com, Live, Yahoo!* and one of Polish origin—*Interia*, which in fact is a Google based engine; however very often it produces results which differ from its parent engine. System was set-up to return 20 results for each query. In this note, due to the lack of space, in Table 4 we present only two "performance measures;" the *Set Coverage* and the and *URL to URL coverage* for each of the three approaches, for each of the queries. The Set Coverage measures how many URLs from the final result are contained in the result set returned by a search engine regardless of the position of the URL. In other words, this measure tells us if there is a relationship between the combined answer and answers returned separately by each search method. The and URL to URL coverage measures how many URLs were at the same position in both results (of the algorithm and that of the search engine).

***Auction* method, simple query**					
Auction	Ask.com	Live	Interia	Yahoo	Google
Set Coverage	60%	40%	110%	60%	70%
URL to URL	0%	10%	20%	30%	20%
***Game theory* method, simple query**					
Game theory	Ask.com	Live	Interia	Yahoo	Google
Set Coverage	60%	60%	70%	80%	60%
URL to URL	30%%	10%	0%	50%	0%
***Consensus* method, simple query**					
Consensus	Ask.com	Live	Interia	Yahoo	Google
Set Coverage	70%	50%	80%	70%	80%
URL to URL	20%	20%	10%	20%	10%
***Auction* method, intermediate query**					
Auction	Ask.com	Live	Interia	Yahoo	Google
Set Coverage	10%	10%	50%	10%	40%
URL to URL	0%	10%	0%	0%	0%
***Game theory* method, intermediate query**					
Game theory	Ask.com	Live	Interia	Yahoo	Google
Set Coverage	60%	40%	30%	40%	30%
URL to URL	40%	0%	10%	0%	10%
***Consensus* method, intermediate query**					
Consensus	Ask.com	Live	Interia	Yahoo	Google
Set Coverage	50%	30%	70%	40%	60%
URL to URL	0%	20%	0%	0%	0%
***Auction* method, very complex query**					
Auction	Ask.com	Live	Interia	Yahoo	Google
Set Coverage	0%	0%	30%	0%	40%
URL to URL	0%	0%	10%	0%	20%
***Game theory* method, very complex query**					
Game theory	Ask.com	Live	Interia	Yahoo	Google
Set Coverage	0%	40%	30%	10%	50%
URL to URL	0%	0%	0%	10%	0%
***Consensus* method, very complex query**					
Consensus	Ask.com	Live	Interia	Yahoo	Google
Set Coverage	10%	20%	90%	20%	40%
URL to URL	0%	0%	30%	0%	0%

Table 1. Summary of experimental results

Let us observe that as the query becomes more complex, the coverage drastically decreases. This can be explained by the fact that the responses generated by various search engines have less and less in common. Therefore, regardless of the method used, the final answer set becomes a collection of "separate links" chosen from each individual answer-set. This trend is even more drastic in the URL to URL comparison. Here, already for the intermediate query practically no URL is in the same location in the answer set as it is in any of the search engines. This indicates also that this performance measure is not particularly useful for the application in question.

As expected, results returned by *Interia* and *Google* are very similar, with both performance measures varying, randomly favoring either one of them. Interestingly, these two search engines seem to have best performance for the complex query. However, this may be a result of collusion, where two similar search engines "dominate" views of the others. This observation provides also a warning, that the selection of the "groups of experts" has to provide as "orthogonal" view of the subject as possible. Otherwise, regardless of the method used, the returned combined answer may be dominated by a few experts that see the problem similarly.

Observed results suggest also that the consensus method does what its name suggests—delivers response that is closest to consensus. This can be seen particularly in the case of the complex query, where for the consensus method the Set Coverage is non-zero also for search engines other than *Interia* and *Google*.

Overall, on the basis of all of our experiments (also these not reported here), we can state that: (1) Results delivered by the *Auction method* are highly dependent on each individual result set and do not represent well the "combined view" of all search engines. No matter if the URL is in many result sets, it may not make it to the final (combined) result. Instead, returned are "winning" URLs, which appear in a single result sets. (2) *Game Theory* method also does not seem to create a combined view of initial answers. However, if a URL is at of of top places of more than one result set, it is very likely to be incorporated into the final result set (even though it may be locate much lower than its average position). (3) The *Consensus method* returned the results which represent the most common view of participating search engines. However in three tested cases all returned result sets were inconsistent(!) according to consensus theory itself. This happens due to the high "position dispersion" of URL's throughout the result sets. There are situations where a URL is, for instance, on the 1st place in one result set, on the 9th place in another result set, and on the 5th in the next. For this result, the Levenshtein distance between response sets is relatively large and thus the final result set is inconsistent. Nevertheless, if one was not to take the consistency into account (as in its current form it may not be a useful measure after all), the *Consensus method* provided results which could be claimed to be "the best overall."

5 Concluding Remarks

In this note we discussed three methods for combining results from multiple Internet sources and presented initial evaluation of their performance. Our results indicate that each method leads to a different combined answer set. Out of these methods, the *Consensus method* seems to generate the most "common" view of the initial answers, while the remaining two methods tend to favor certain answers over others. This is particularly the case for the *Auction theory*. We are currently performing additional experiments with the three methods and starting to look more qualitatively into obtained answer sets (to establish their value for actual users).

6 Acknowledgment

Work was supported from the "Funds for Science" of the Polish Ministry for Science and Higher Education for years 2008-2011, as a research project.

References

1. A. M. P. Canuto and M. Abreu. Analyzing the benefits of using a fuzzyneuro model in the accuracy of the neurage system: an agentbased system for classification tasks. In *Proceedings of the International Joint Conference on Neural Networks*, pages 2951–2958, 2006.
2. N. Nguyen. Consensus system for solving conflicts in distributed systems. *Journal of Information Sciences*, (147):91–122, 2002.
3. N. Nguyen. Processing inconsistency of knowledge at semantic level. *Journal of Universal Computer Science*, 11(2):285–302, 2005.
4. N. Nguyen. Methods for achieving susceptibility to consensus for conflict profiles. *Journal of Intelligent and Fuzzy Systems: Applications in Engineering and Technology*, 17(3):219–229, 2006.
5. N. Nguyen, M. Ganzha, and M. Paprzycki. A consensus-based approach for information retrieval in internet. Number 3993 in Lecture Notes in Computer Science, pages 208–215. Springer, 2006.
6. N. Nguyen and M. Małowiecki. Consistency measures for conflict profiles. In *Transactions on Rough Sets*, volume 1 of *LNCS*, pages 169–186. Springer, 2004.
7. L. Santana, A. Canuto, and M. Abreu. Analyzing the performance of an agent-based neural system for classification tasks using data distribution among the agents. In *Proceedings of the International Joint Conference on Neural Networks*, pages 2951–2958, 2006.
8. L. Santana, A. Canuto, J. Junior Xavier, and A. Campos. A comparative analysis of data distribution methods in an agentbased neural system for classification tasks.santana l.e.a., canuto a. m. p., junior xavier j.c., campos a.m.c., a comparative analysis of data distribution methods in an agentbased neural system for classification tasks. In *Proceedings of the Sixth International Conference on Hybrid Intelligent Systems*, number 9, 2006.
9. E. Szymanska. Personal communication.