Towards Enhanced Decision Support through Learning from Past Experiences

Maher Rahmouni

HP Laboratories
HPL-2009-136

Keyword(s):
decision support, data mining, optimization, statistical learning

Abstract:
The number of changes that IT departments have to deal with is growing at a fast pace in response to changing business needs of enterprises. As changes are getting executed and deployed, knowledge is being created and stored. It is of paramount importance to the success of the business to re-use that knowledge for future changes. In fact, those who do not learn from past experiences are doomed to repeat the same mistakes as well as not bear the fruit of the ones that were successful. This paper addresses this concern by providing for every change being worked out the most similar past changes. Our approach uses an optimization paradigm to model the problem of finding past similar changes by designing and learning similarity functions. Our approach enhances the efficiency and effectiveness of dealing with changes, by reducing the risk and shortening the time of introducing new changes.

External Posting Date: June 21, 2009 [Fulltext] Approved for External Publication
Internal Posting Date: June 21, 2009 [Fulltext]

Towards Enhanced Decision Support through Learning from Past Experiences

Maher Rahmouni

Maher.Rahmouni@hp.com

Abstract

The number of changes that IT departments have to deal with is growing at a fast pace in response to changing business needs of enterprises. As changes are getting executed and deployed, knowledge is being created and stored. It is of paramount importance to the success of the business to re-use that knowledge for future changes. In fact, those who do not learn from past experiences are doomed to repeat the same mistakes as well as not bear the fruit of the ones that were successful. This paper addresses this concern by providing for every change being worked out the most similar past changes. Our approach uses an optimization paradigm to model the problem of finding past similar changes by designing and learning similarity functions. Our approach enhances the efficiency and effectiveness of dealing with changes, by reducing the risk and shortening the time of introducing new changes.

Introduction

IT environments are becoming increasingly complex and dynamic. Changes are constantly introduced responding to changing business needs such as improving services or reducing costs as well as solving existing incidents or problems. A typical enterprise deals with hundreds of thousands of changes every year. For these reasons, a disciplined process is needed to ensure that required changes are carried out in a planned and authorised manner while minimising the impact and risk of those changes upon service quality. There are several activities involved in managing an individual change. It starts by creating and submitting a request for a change (RFC). The RFC gets reviewed, assessed and eventually authorised or rejected. Once the RFC is authorised, plans are created including implementation plans, test plans and back out plans in case the change fails. Then, the change is built, tested and implemented. Finally, the change is reviewed to ensure that the change has the desired effect and met the required objectives. During the lifecycle of a change, information is collected and knowledge is created and stored in the Configuration Management Database. A CMDB contains details of the organizations' elements referred to as Configurations Items (CIs) that are used in the provision and management of its IT services. Among those details, a CMDB contains information about all the changes that have been performed on those elements. It includes the plans for implementing those changes, the risk involved, the back out plan, the resources used, the impacted applications, the incidents triggered by implementing those changes, whether or not a change has been successfully implemented, the reason(s) why a change has failed and so on. Since knowledge is created during the lifetime of a change, it is of paramount importance to the success of the business to re-use that knowledge for future changes. In fact, those who do not learn from past experiences are doomed to repeat the same mistakes and also would not bear the fruit of the ones that were successful. What if a change manager (person responsible for approving and overseeing the change from start to end), is presented with the outcomes of the most similar changes? He will be able to better assess the risk and impact associated with the change at hand and consequently take the appropriate course of action. What if a change planner (person responsible for designing the different plans for a change) is presented with the plans of the most similar changes? He will be able to learn from those plans and produce plans in a shorter period of time with a lower risk of failing. What if a change reviewer (person responsible for monitoring changes after their implementation) is presented with applications that were disrupted by past changes similar to a recently completed change request? He will be able to identify it as a potential root cause to newly created incidents.

In order to answer all these questions, we need to be able to, given a change, find the most similar past changes and display them. The information contained in a change record consists of various attribute types, ranging from complex types such as text documents or graph representations, to basic types such as strings, integers and booleans. There are various techniques in literature for computing the similarities of such attributes [1, 9, 10, and

11]. While the similarity of such attributes is important, it does not necessarily mean that the corresponding changes are similar. Consequently, given the similarity of such attributes, there is a need for a way of classifying two changes as being similar or not, more importantly, if two changes are similar, there is a need for a measure of how similar they are. In literature, there are several classification techniques [12] which attempt to place individual items into groups based on information about those items as well as a training set of previously classified items. The classified items are usually generated with the help of a subject matter expert. The main difference of our approach compared to existing approaches is the fact that we model the classification problem as an optimization problem where we try to find one or more similarity functions while at the same time classifying the changes as being similar or not with respect to those similarity functions. This will allow us to pick from the set of similar changes only the changes that have high similarity values to the change at hand.

The remainder of the paper is organized as follows. Section 2 describes the framework for designing and learning similarity between changes using their attributes similarity as well as expert feedback. Section 3 discusses some techniques for capturing similarity between single attributes according to their type. In section 4, details of the approach used to find and learn similarity between changes are depicted. Empirical results of our approach are presented in section 5. Finally, we draw our conclusions and give a preview of the next steps in section 6.

Change similarity design and learning system

Figure 1 depicts our solution. Every time a new Request for a Change (RFC) comes in to the system, it is compared to changes stored in the Change Knowledge Base. The CKB consists of the set of all changes performed in the past. The comparison of the actual RFC with a particular change makes use of a similarity function that will be described in section 4. The most similar changes will be displayed to the user along with a measure (value between 0 and 1) of how similar the change is. A domain expert will provide feedback on whether or not the displayed changes are genuinely similar to the actual RFC. The feedback consists of stating whether the two changes are similar or not. Once the expert feedback has been recorded, a set of new training elements will be created and added to the training set. A training element is an object representing the actual RFC, a change deemed to be similar according to the actual similarity functions, the similarity between each attribute of both changes, along with the expert feedback. A training element with a positive feedback is called *positive element* and a training element with a negative feedback is called *negative element*. Note that the training elements are added to the training set if and only if there is at least one change deemed similar according to the similarity functions but not similar according to the expert. In fact, if all feedback were positive then the existing similarity functions are able to accurately classify changes hence there is no need to learn new ones.

If the training set has been updated, then the actual similarity functions are not able to accurately determine the similarity between an RFC and a change. Consequently, there is a need to learn new similarity functions using all expert feedback. Once new similarity functions are learned, it will be used for future detection of similarities between an RFC and similar changes.

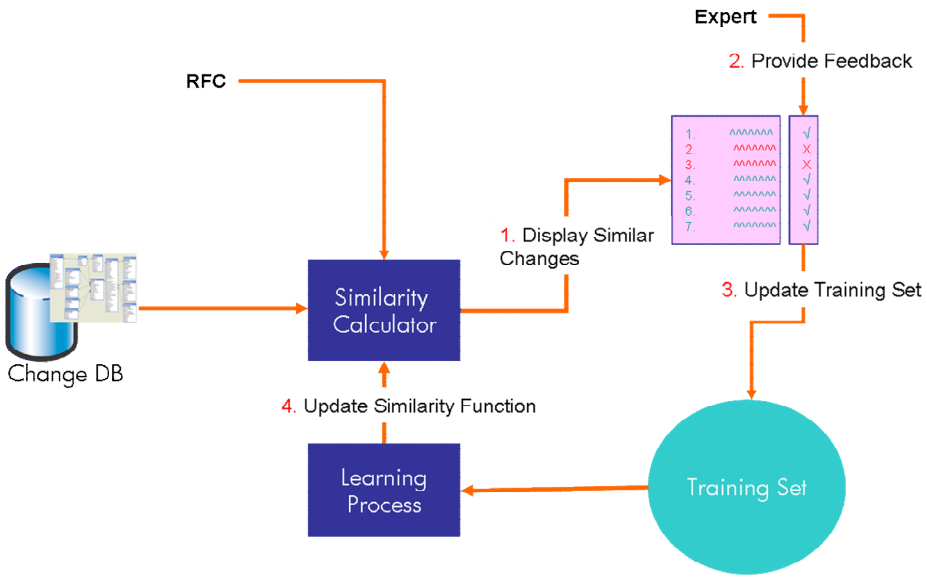

Figure 1: Framework for change similarity design and learning

Similarity Calculation

An RFC is modelled as an object with a set of attributes as shown in table 1. The list of attributes is not exhaustive, so in practice some attributes could be present, others could be missing.

Table 1. Example of attributes of a Request for a Change

Attributes	**Description**	**Type**
Description	This is a summary description of the change.	Text
CIs to be changed	List of CIs	Set
Affected Applications	List of applications affected and their topology	Tree or a graph
Change Implementer (s)	Who will implement the change (a single person or a group)	Object or Set of objects

Depending on the type of the attribute, we use an appropriate similarity function based on a pre-defined distance (similarity and distance notions are dual) [1] to calculate the similarity between the same attributes of different RFCs. For example, for a vector of integer's type, we could use a similarity function based on the Euclidian distance, a similarity function based on a Hamming distance is useful for strings of same length and a Levenshtein distance is more suitable for strings of different length. We have derived a modified version of the Levenshtein distance to calculate similarity between a set of objects. For text documents a cosine similarity is the most appropriate. In the following, we will address more in detail the cosine similarity since a lot of knowledge is captured in a textual form, the Levenshtein distance for its usefulness for more than a type of attributes and finally we will tackle how to transform a distance into a similarity measure.

Cosine Similarity

The root of the cosine similarity measure comes from vector analysis theory. The cosine angle between two vectors represents how close those vectors are. In other words, the closer the two vectors are, the more the cosine of the angle approaches 1. The cosine angle between two vectors is represented by the ratio between their dot product and their Euclidian distance. To apply this approach to compare text documents in a collection, the latter are represented as term vectors where the coordinates of the vector are the term weights. There are many ways to represent a term weight, the most widely used weight is: $tf * log\ (N/n)$ where tf is the term frequency, N is the total

number of text documents in the collection and *n* is the number of documents containing the term. The cosine similarity is used in many search engines to rank text documents with respect to a given query.

Levenshtein Distance

The Levenshtein distance also called the edit distance have been widely used especially to measure similarity between strings. This distance is computed as the number of edits (insertions, deletion and substitutions) needed to transform a string A into a string B. There are many applications to this measure including spell checking, to identify duplicated content and plagiarism, for spam stemming and spamdexing search engines, regular expressions approximate matching and many others. While it has been mainly used for strings, it could be used to calculate the similarity between a set of objects or even trees. Since strings are ordered characters, if we consider objects or tree nodes as characters and define an order for (the set of objects) or a traversal order on trees then it would be easy to apply this measure to compare two different sets of objects or two trees.

From distance to similarity

As we said earlier, there is a notion of duality between similarity and distance. A similarity could be modelled as a function *S=f(d)* where:

- *f(0)=1*, meaning that two objects are similar if their distance is equal to 0.
- If *d1 <= d2* then *f(d1) >= f(d2)*, meaning that the closer the distance of two objects, the more similar they are.

Unfortunately there are many functions that could satisfy those properties and the choice of a good function depends on the problem at hand. Well known functions are:

- *S=1/1+d*
- *S=exp(-d/2)*
- *S=1-d/M* for *d<M*, 0 otherwise.

Similarity Learning

The learning process consists of finding at least a similarity function (S_{ij}) and a threshold (t) that maximises the number of training elements set correctly. If we define the similarity function as the linear combination of the attributes similarity:

$$S_{ij} = \sum_{k=1}^{n} w_k S_{ijk} \qquad (1)$$

Where:

- S_{ij} defines the similarity between RFC_i and RFC_j;
- w_k is the weight associated to attribute k. The values of the weights w_k represent the significance of the corresponding attribute in determining the similarity between RFCs.
- S_{ijk} is the similarity between the k^{th} attributes of RFC_i and RFC_j.

Then the learning process could be modelled as an MIP (Mixed Integer Program):

(*P1*): Maximise the number of training elements set correctly

$$Max \quad \sum_{c=1}^{m} u_c \quad \text{such that:}$$

$u_c = 1$ if training element c is set correctly, 0 otherwise.

A training element (RFC_i ,RFC_j , (S_{ij1},.., S_{ijn}), f) is considered to be set correctly if and only if:

- If f is true, then $S_{ij} - t >= 0$
- If f is false, then $S_{ij} - t < 0$

$$\sum_{k=1}^{n} w_k = 1$$

$0 < w_k < 1$

This problem is known as the *maximum feasible subsystem problem* (*MFSP*) [4]. Unfortunately, this problem is known to be NP-hard [4, 5, and 6] hence finding the optimal solution for a large training set is out of reach. However there are several fast heuristics [2] to solve the *MFSP* problem producing results covering more than 90% of the optimal set of feasible constraints in polynomial time. Figure 2 depicts the algorithm (*LearnOneSimilarityFunction*) for finding the *MSFP* which could be described as follows:

1. We transform *P1* into a linear program *P2*. Contrary to an *MIP*, a linear program can be solved in polynomial time [7].

   (*P2*): Max t such that:

   $$\sum_{k=1}^{n} w_k = 1$$

   $0 < w_k < 1$

   For each training element:

   - If f is true, then $S_{ij} - t >= 0$
   - If f is false, then $S_{ij} - t < 0$

2. Elasticize *P2* into *EP2*. An elastic program [8] is a relaxation of an LP by adding/subtracting (depending on the direction of the inequality) non negative elastic variables to the constraints. By design an elastic program is always feasible. An elastic program has the advantage of providing useful information about the infeasibility of the original LP. The objective function of the elastic program is to minimize the sum of the elastic variables. If the value of the objective function is equal to 0 then the original LP (*P2*) is feasible. The value of an elastic variable indicates whether the corresponding constraint is violated (value equal to 0) or not (value different than 0).

(*EP2*): *Min* $\sum_{c=1}^{m} e_c$ such that:

$$\sum_{k=1}^{n} w_k = 1$$

$0 < w_k < 1$

For each training element :

- If f is true, then $S_{ij} - t + e_c >= 0$
- If f is false, then $S_{ij} - t - e_c < 0$

$e_c >= 0$

3. The last step consists at iterating through a process of deleting a constraint from the elastic program until the original LP (*P2*) becomes feasible. The choice of the constraint depends on the impact of its deletion on the drop on the objective function. As observed by [2], "*a good predictor of the magnitude of the drop in the objective function that will be obtained by deleting the constraint is given by the product (constraint violation)* × | *(constraint sensitivity)*|", where the constraint violation represents the value of the elastic variable and the constraint sensitivity is the reduced cost of the variable associated to the constraint (value of the dual variable of the constraint).

4. Once all constraints causing the infeasibility of the original LP problem (P2) have been removed, P2 becomes feasible and solving this problem will produce the similarity function and the threshold separating negative and positive elements.

```
INPUT:    Original infeasible Linear program (P2)
STEP1:    CoverSet=ø
          Elasticize P2 into EP2
          Solve EP2
STEP2:    If number of constraints with non zero elastic variable = 1 then:
              Add corresponding constraint to CoverSet
              Goto STEP3
          Else
              Remove the constraint with the highest product (elastic variable * |constraint sensitivity|) from EP2
              Add corresponding constraint to CoverSet
              Solve EP2
              If objective function = 0 then:
                 Goto STEP3
              Goto STEP 2
STEP3:    Remove all constraints in CoverSet from P2
          Solve P2
OUTPUT:   Similarity function represented by (w1 , ..,wn) and threshold t
```

Figure 2: Algorithm for learning a similarity function: *LearnOneSimilarityFunction*

While this algorithm produces very good results in solving the MSFP problem, it doesn't however cover a big part of the training set. This drawback is mainly due to the nature of the problem at hand meaning that one similarity function is not sufficient to cover all the cases of similarities in the training set. The following example will illustrate this issue:

Let's assume a change is composed of 4 attributes and the training set consists of the following 6 elements:

Table 2. Example of a training set

Training element	**Changes**	***Expert Feedback***	***Attribute 0 Similarity***	***Attribute 1 Similarity***	***Attribute 2 Similarity***	***Attribute 3 Similarity***
t0	c1, c2	true	0.75	0.12	0.89	0.3
t1	c3, c4	true	0.88	0.2	0.73	0.15
t2	c5, c6	true	0.05	0.92	0.11	0.77
t3	c7, c8	true	0.1	0.72	0.23	0.88
t4	c9, c10	false	1	0.7	0.03	0.2
t5	c11, c12	false	0.09	0.8	0.77	0.14

Each training element consists of a couple of changes (c_i, c_j), the user feedback (true/false) stating whether or not the two changes are similar and the similarity of the attributes of both changes. From the given example, we can see that t0 and t1 share high similarities for attributes a0 and a2, while t2 and t3 share high similarities for attributes a1 and a3. Training elements t4 and t5 state that two changes having high similarity for attributes (a0, a1) respectively (a1, a2) are not similar. Furthermore, we can observe that any MSFP would contain either t0 and t1 or t2 and t3 but not both. If we apply the algorithm on figure 2 over this training set, it will eliminate t0 and t1 (as the corresponding constraints were causing the infeasibility of the system) and produces a similarity function *f1* represented by (w_1 , w_2, w_3, w_4) = (0, 0.35, 0, 0.65) and a threshold $t = 0.82$. If we construct a new training set composed of t0, t1 (positive elements excluded by applying the algorithm on the whole training set) as well as the negative elements t4, t5 then by applying the same algorithm, we obtain a similarity function *f2* represented by (w_1 , w_2, w_3, w_4) = (0.55, 0, 0.45, 0) and a threshold $t = 0.81$. The negative elements t4 and t5 have been added to the training set to make sure that every similarity function will exclude all negative elements from being classified as similar. Table 3 shows the calculated similarities for all training elements using the functions *f1* and *f2*.

Table 3. Similarity results for example training set

Training element	**Changes**	***f1***	***f2***
t0	c1, c2	-	0.81
t1	c3, c4	-	0.81
t2	c5, c6	0.82	-
t3	c7, c8	0.82	-
t4	c9, c10	0.37	0.56
t5	c11, c12	0.37	0.39

The new algorithm for learning multiple similarity functions is displayed in figure 3. At each iteration of the algorithm, a new similarity function is created by applying the *LearnOneSimilarity* algorithm onto the training set. The training set is composed of all negative elements and the remaining positive elements (all positive elements whose corresponding constraints have been identified as violated). The algorithm stops when there are no remaining positive elements or the MSFP contains only negative training elements.

```
INPUT:    Training Set
          LOOP
              Apply the LearnOneSimilairtyFunction onto the training Set
              Remaining Positive Elements:=CoverSet;
              Training Set  = Negative Elements + Remaining Positive Elements
          Until (Remaining Positive Elements= ø or no new function can be learned)

OUTPUT:   All similarity functions
```

Figure 3. Algorithm for learning multiple similarity functions

Empirical Results

We have implemented the algorithm for learning multiple similarity functions and we have designed a set of examples to test the performance and efficiency of the algorithm. Since real data were not available, we have generated random training elements, in other words, for each attribute a random similarity (between 0 and 1) have been generated. In order to emulate the expert feedback, we have classified training elements according to the similarity of the attributes as shown in table 4. For each example, changes are considered similar if one of the conditions over the attributes similarity is satisfied.

Table 4. Test cases for the algorithm

	Conditions for similarity between changes	**Number of attributes**
Example 1	• s0 >= 0.9 and s2 >= 0.6 • s1 >= 0.6 and s3 >= 0.8	4
Example 2	• s4 >= 0.6 and s7 >= 0.8 and s0 >= 0.7 and s8 >= 0.8 • s0 >= 0.9 and s2 >= 0.6 and s6 >= 0.8 • s1 >= 0.6 and s3 >= 0.8 • s2 >= 0.6 and s9 >= 0.8 • s5 >= 0.9	10

Figure 4 shows the results of applying the algorithm on training sets of different sizes (number of training elements). Once the similarity functions have been identified, we generated randomly 50000 training elements and checked the accuracy of the algorithm on classifying those training elements against the similarity functions.

A positive training element is considered to be misclassified if its similarity value is below the threshold for every similarity function. However, a negative training element is misclassified if its similarity value is above the threshold for at least one similarity function.

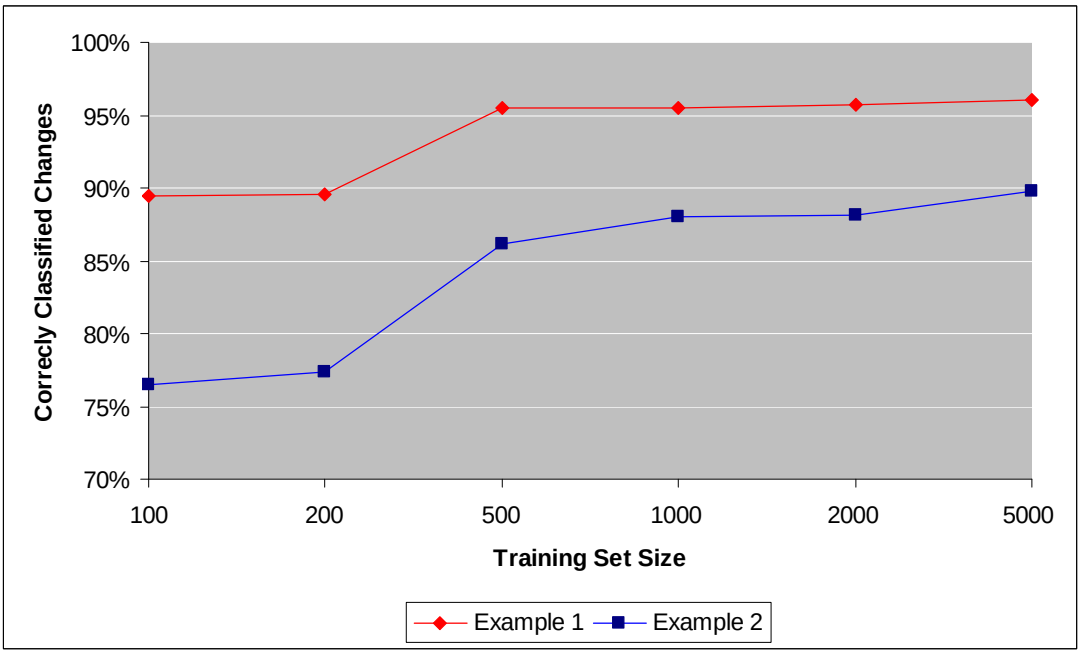

Figure 4: Algorithm classification accuracy

It is obvious to see that the bigger the training set the more accurate the results are. For both examples, a big increase in efficiency (more than 5%) is seen between moving from a training set size of 200 to 500. However, beyond that point the gains in accuracy are significantly lower especially if we look at the tradeoff between the performance of the algorithm and its efficiency as shown in table 5. Consequently, a training set of a size of 500 is an excellent candidate for an initial training set, because it has a far better accuracy than 200 with little decrease in performance and it has a far better performance than 5000 with little increase in accuracy.

Table 5. Algorithm performance

	Example 1				Example 2			
Training Set size	LPs	Time (sec)	avg/LP (sec)	similarity functions	LPs	Time (sec)	avg/LP (sec)	similarity functions
100	9	0.078	0.008	2	27	0.188	0.007	4
200	11	0.203	0.018	3	56	0.750	0.013	6
500	71	1.422	0.020	3	150	5.172	0.034	8
1000	180	7.438	0.041	4	410	44.25	0.108	5
2000	192	36.172	0.181	2	904	370.199	0.409	6
5000	510	1138.265	2.231	2	2580	14564.53	5.640	7

Conclusion

We are in the process of finalizing a prototype for our solution. The learning algorithm is promising and produces very good results for large training sets in terms of the cover of different similarity cases. The algorithms presented in this paper have been implemented in the Control and Change Management Center (CCM) and at the time of writing this paper, are being validated at customer sites. A patent covering the approach is being filed. The next steps would be to get more customer data and extensively test our solution. Our research efforts will be next focused at how best the results (similar changes) could be used to enhance and better guide the decision maker for the best possible choices. In terms of enhancements, we are exploring graph based similarity distances especially for attributes represented topologically like the configuration items. Furthermore, we are also looking at applying

the same approach in other areas of ITSM (IT Service Management) like incident and problem management.

References

[1] Dekang Lin, "An Information-Theoretic Definition of Similarity", *Proceedings of International Conference on Machine Learning*, Madison, Wisconsin, July, 1998.

[2] John W. Chinneck, "Fast Heuristics for the Maximum Feasible Subsystem Problem", INFORMS Journal on Computing, vol. 13, no. 3, pp. 210-223, 2001.

[3] J. F. Roddick et al, "A Unifying Semantic Distance Model for Determining the Similarity of Attribute Values", Proceedings of the 26th Australasian Computer Science Conference, Adelaide, Australia, Vol 16, ACS, pp. 111-118.

[4] Amaldi, E., V. Kann. 1995. The complexity and approximability of finding maximum feasible subsystems of linear relations. *Theoretical Computer Science* 147 181–210.

[5] Sankaran, J. K. 1993. A note on resolving infeasibility in linear programs by constraint relaxation. *Operations ResearchLetters* 13 19–20.

[6] Chakravarti, N. 1994. Some Results Concerning Post-Infeasibility Analysis. *European Journal of Operational Research* 73 139–143.

[7] Karmarkar, N. 1984. A new polynomial time algorithm for linear programming. *Combinatorica, Vol. 4, No. 4. (December 1984), pp. 373-395*.

[8] Brown, G., G. Graves. Elastic programming: a new approach to large-scale mixed integer optimization. ORSA/TIMS conference, Las Vegas, NV, 1995.

[9] Irina Matveeva, " Document Representation and Multilevel Measures of Document Similarity", *Proceedings of the Human Language Technology Conference of the North American Chapter of the ACL*, pages 235–238.

[10] J. Tomita, H. Nakawatase, M. Ishii, "Calculating Similarity Between Texts using Graph-based Text Representation Model", *CIKM'04,* November 8–13, 2004, Washington, DC, USA.

[11] P. Resnik. Semantic similarity in a taxonomy: An information based measure and its application to problems of ambiguity in natural language. *Journal of Artificial Intelligence Research*, 11:95-130, 1999.

[12] M. James, "*Classification Algorithms"*, Wiley-Interscience, 1995.