Error Detection, Recovery, and Repair in the Translation of Inorganic Nomenclatures. 2. A Proposed Strategy

I. Luque Ruiz,*,‡ J. L. Cruz Soto,† and M. A. Gómez-Nieto‡

E.U.P., Avda. Menéndez Pidal s/n, and Science Faculty, Avda, S. Alberto Magno s/n, Córdoba University, 14071-Córdoba, Spain

Received June 13, 1995⊗

The repair of lexicographic errors occuring during the computer translation of knowledge is performed by approximate string matching algorithms. This article presents a new algorithm that attempts to repair lexical errors in inorganic chemical names. The algorithm allows the repair of an incorrect string containing one or more errors due to deletion, insertion, or transposition of multiple characters. Repair may be carried out on all or only part of the incorrect string, so the algorithm may be used to repair strings containing more than one incorrect morpheme. A model is proposed for the use of this algorithm in the repair of lexical errors in inorganic chemical names. The model attempts to generate a set of possible repair strings free from either syntactic or lexicographic errors. Repair is performed by comparing all or part of the error string with the set of terminal symbols or morphemes included in the grammar. A new model is proposed for the calculation of similarity; it is highly discriminating and allows the repair of parts of an error string.

1. INTRODUCTION

Pattern-matching has long been a problem in computing science and is relevant to all areas of arts and sciences. The fastest known algorithms are based on the original ideas of Boyer−Moore for exact string matching,[1−5] although in the last decade much work has been directed at the problem of approximate string matching. Two types of algorithms have been developed to this end: one aims to find all occurrences of a pattern P in a text T with a limit of k differences (*k mismatches problem*); the second type attempts to find all substrings T' of a text T whose difference when compared with a pattern P is below a threshold k (*k differences problem*). Both algorithm types rely on measuring the edit distance in order to determine the *similarity* between strings;[6,7] in other words, the number of deletion, insertion, or substitution operations needed for the pattern and the text to be identical, as described in the first article of this series.[8]

The use of distance as a metric to determine the similarity of two objects is widespread in many areas of research, particularly in chemistry for computer-assisted synthesis design (CASD), database access, searching for molecular substructures, etc.[6,9] With respect to string matching, however, there is currently only an absolute measurement of additional computing cost and no conceptual measurement of the "*likeness*" of two strings.

The present article describes the measurement of similarity between strings; this measurement is more sensitive than that of edit distance and is thus suitable for use in the repair of lexical errors in chemical names. It is obtained from an approximate matching algorithm based on the philosophy of the algorithms proposed by Ukkonen,[1,10−12] but which yields fuller information on the matching process.

2. EXAMINING THE PROBLEM

In the translation of knowledge between different representations, the input information is subjected to successive stages of analysis which check: content (lexical analysis), structure (syntactic analysis), and meaning (semantical analysis); a new representation of the knowledge is then built during the second (synthesis) stage.[13−16]

In the case of chemical names, when the strings representing chemical substances in any of the established nomenclatures are subjected to a translation process in order to generate a different representation of the knowledge (be this representation abstract, symbolic or natural), the process may fail due to the appearance of lexical errors, errors due to the presence in the input string (the substance's name) of substrings (morphemes) that do not coincide with any of the terminal symbols or vocabulary of the source language (the nomenclature).[8]

The substitution of incorrect substrings by morphemes belonging to the vocabulary is known as *repair*, and in the case of chemical names should produce a new string which is free of lexicographic and syntactic errors. The morphemes which repair the incorrect substrings must be a subset of the language's vocabulary, that is, they must produce a new string whose structure satisfies the grammatical rules defining the chemical language (the nomenclature).

Take the following input string, composed of morphemes belonging to the terminal vocabulary of a language L and which contains incorrect substrings

$$C \equiv \{t_1 t_2 e_1 t_3 e_2 t_4\}$$

where t_i are the substrings that have been recognized by the lexical analyzer and for each of which a lexical component (a morpheme defined in the syntactic analyzer) has been built, and where e_i are error substrings for which the lexical analyzer is unable to find a grammar-defined morpheme.

The process of repairing the string C consists of finding, for each incorrect substring e_i, a set of substrings r_{i_j} composed of correct repairs. Thus, for the substring e_1, it will find a

† Science Faculty. e-mail: jlcruz@sun630.uco.es.
‡ E.U.P. e-mail: ma1gonim@lucano.uco.es:: ma1lurui@sun630.uco.es.
⊗ Abstract published in *Advance ACS Abstracts,* November 1, 1995.

0095-2338/96/1636-0016$12.00/0

set R_1 of repairs formed by r_{1_j} elements, and, for the substring e_2, another set R_2 formed composed of r_{2_k} elements. Both sets, R_1 and R_2, correctly repair the substrings e_1 and e_2. However, in the set $R = R_1 \times R_2$ of all possible "global" repairs of the input string, C, only one subset $R^* \subseteq R$ will produce the new, lexically and syntactically correct strings $C^* \equiv \{t_1t_2t_3r^*_{1_j}t_3r^*_{2_k}t_4\}$.

Hence, the problem of repair may be comceptually divided into three different problems.

Problem 1. Find, for each incorrect substring e_i, the substring r that is most "*similar*" to it. Here it can be clearly seen that we have a classical approximate matching problem where a pattern set Σ_T has to be searched for the substring $r \in \Sigma_T$ which is most similar to the error string. The set Σ_T is the lexicon, and in the present case comprises the terminal symbols or morphemes defined for the grammars of the nomenclatures, such that the string r is lexically correct.[8]

If instead of seeking just one repair we wished to find the set of strings $r_j \in \Sigma_T$ whose similarity to the error string was greater than a fixed threshold, we would have as a result the subset $R_i \equiv \{r_1,r_2,...,r_j\}$, $(R_i \subset \Sigma_T)$ of repair strings for the error string e_i.

Problem 2. Eliminate, from each subset R_i of lexically correct repairs, all those repairs which are not syntactically correct. For a repair to be syntactically correct, the morpheme must be of a type or category that results in the corresponding fragment or "part" of the name fitting in with the grammatical structure of inorganic names.

As the fragments making up inorganic compound names are rarely differentiated by delimiters, and since a fragment may appear in more than one error string e_i, the decision of whether a repair is "correct" with respect to the fragment will often need to be based on whether the repair is syntactically correct in the context of the entire string. The solution to this problem would be simpler if there was only one error string e_i in the input string C but becomes increasingly complicated with the appearance of additional errors. Note that to regard a repair as syntactically correct it is necessary to assume that the remaining morphemes in the string are also correct, obviously impossible when $i > 1$.

In order to simplify the problem we must then consider that the solution to *Problem 2* for a set of repairs R_i is another set $R'_i \subseteq R_i$, such that each element r'_{i_j} is a *morpheme that is syntactically correct in its particular environment*.

Let us say that the environment of a morpheme, m, in a string is composed of two morphemes x, y, possibly null, situated to the left and right, respectively, of the morpheme in question. Let us call these *leading* and *trailing* morphemes, using these terms in preference to *prefix* and *suffix* in order to avoid confusion with the chemical meaning of the latter terms.

The set of repairs R'_i for an error string e_i will thus be composed of r'_{ij} morphemes which, on being substituted for the error string in the input string C, will produce a new string C' that is both lexically and semantically correct in the repair environment.

Problem 3. Given the sets of R'_i, corresponding to the c incorrect substrings present in the string C, the problem lies in determining the subset of global solutions that is syntactically correct. This set of global solutions or repairs will be equal to

$$R^* \equiv R'_1 \times R'_2 \times ... R'_c$$

whose elements are combined thus $r^*_i = (r'_{1_i},r'_{2j},...,r'_{c_k})$ and where r'_{i_j} is an element of the set R'_i of repairs which are correct in this environment.

To determine the elements of the set R^* means working out which elements of the set $R'_1 \times R'_2 \times ... \times R'_c$ are syntactically correct within the context or, in other words, conform to the structure of the language.

3. DETERMINING FACTORS IN THE REPAIR PROCESS

It is clear that this model presents a series of problems to be solved in the process of lexical error repair. These shall be now be discussed.

3.1. Complexity of Repair. Let us again take the input string $C \equiv \{t_1t_2e_1t_3e_2t_4\}$, which requires the error substrings e_1 and e_2 to be correctly repaired.

As described in the proposed model, the first step is to find, for each string e_i, a set R_i of lexically correct repairs; this is achieved by performing an approximate matching process on the set of terminal symbols Σ_T.

Here, of course, we are supposing that the error strings e_i are attributable to only one morpheme, and this is obviously oversimplistic. If we take it that the strings e_i may be composed of or due to more than one morpheme, then two procedures would seem to be appropriate for finding the repair set R_i.

1. Perform the matching process on the terminal symbol set Σ_T and on the nonterminal symbol set Σ_N. To achieve this in the present context, all possible chemical names would have to have been previously defined in a lexicon to be used for matching. This process would not be possible in the case of nomenclatures or, indeed, in any other type of artificial languages, be they scientific or natural.

2. Carry out the matching process in stages, over the terminal set Σ_T. At each matching step the elements Σ_T are compared with the error string e_i, obtaining the set U_i of morphemes whose similarity to "all" or "part" of the error string e_i is greater than a previously-established threshold.

The set U_i is made up of two disjoint subsets, $U_i = T_i \cup P_i$, $T_i \cap P_i = \{0\}$, whose elements are morphemes that either completely repair the error string (T_i) or which only partially repair it (P_i).

Every repair $p_{i_j} \in P_i$ will produce two new error substrings, $e^L_{i_j}$ and $e^R_{i_j}$, one of which might be empty, from the erroneous substring e_i. So

$$e_i \equiv \{e^L_{i_j}p_{i_j}e^R_{i_j}\}$$

$e^L_{i_j}$ is the substring of the string e_i lying to the left of the string p_{i_j} that partially repairs the string e_i, and $e^R_{i_j}$ is the equivalent substring to the right.

The second stage of matching is now performed on the new substrings $e^L_{i_j}$ and $e^R_{i_j}$, generating the new subsets $U^L_{i_j}$ and $U^R_{i_j}$ whose elements may be total or partial repairs of these new error substrings. The process is repeated until no new error substring $e^{L...}_{i_{j...n}}$ or $e^{R...}_{i_{j...m}}$ is generated by a repair. In other words, when no partial repairs of the nature $P^{L...}_{i_{j...n}} = P^{R...}_{i_{j...n}} = \{0\}$ remain for the sets $U^{L...}_{i_{j...n}}$ and $U^{R...}_{i_{j...m}}$.

The set of lexically-correct repairs will then comprise

(a) The total repair set T_i of the error substring e_i.

(b) The set of repairs calculated by the concatenation of the partial repairs successively obtained from the repair of new error strings generated from a string e_i. Strings such as the following

$$t^{LL...}_{i_{mnp}} p^{L...}_{i_{mnp}} t^{LR...}_{i_{mnp}} \ldots p_{i_j} \ldots t^{RL...}_{i_{uvw}} p^{R...}_{i_{uvw}} t^{RR...}_{i_{uvw}}$$

or the sum of the partial and total solutions for the new strings $e^{L...}_{i_{j...n}}$ and $e^{R...}_{i_{j...m}}$ produced by the successive partial solutions.

It can be seen that when the error string e_i is due to more than one morpheme, the problem of finding the correct lexical solution becomes increasingly complex since

•An approximate string matching algorithm must be used, by which all or part of the error string may be repaired.

•The number of incorrect fragments or morphemes in the string e_i increases according to the number of matchings needed.

•The stage-by-stage process can produce redundant solutions for e_i.

Suppose that a string e_i is made up of two morphemes e_1 and e_2 and we carry out the procedure described above (although in this case we shall considerably simplify the process in order to clarify the explanation):

First step: $T = \{0\}$, where no total solutions are obtained since no terminal symbol is capable of completely repairing the string. However, for each morpheme, a set of partial repairs is obtained. Thus

For e_1 we get: $p_{11}, p_{12}, \ldots p_{1m}$

For e_2 we get: $p_{21}, p_{22}, \ldots p_{2n}$

Second step: Partial repairs obtained from the previous step produce new error substrings. Let us say that the partial solution p_{11} for e_1 produce a new substring $e^L_{11} p_{11} e^R_{11}$, where, for example, $e^L_{11} = \{0\}$ and e^R_{11} may be equal to or different from e_2.

For p_{21}, we get a new string such as $e^L_{21} p_{21} e^R_{21}$, where $e^R_{21} = \{0\}$ and e^L_{21} may be equal to or different from e_1.

Third step: carrying out matching on the newly-obtained strings we might get:

$$e^R_{11} \equiv \{t^R_{111}, t^R_{112}, \ldots t^R_{11m}\}$$

$$e^L_{21} \equiv \{t^L_{211}, t^L_{212}, \ldots t^L_{21n}\}$$

where, to simplify, we will suppose that all repairs are total.

Thus, global solutions for the error string e_1 would be elements of the set R'_1 of the nature $(p_{1i} e^R_{11})$, $(e^L_{2i} p_{2i})$, in which there would be redundancy since $p_{1i} \cap e^L_{2i} \neq \{0\}$, y $e^R_{1i} \cap p_{2i} \neq \{0\}$. This redundancy means that more matching operations will be made than are strictly necessary.

Note that, although for the total set of repairs for a substring e_i there may be redundant solutions formed by the concatenation of successive partial solutions, during the derivation of these partial solutions the similarities produced, although redundant (the same morpheme or partial repair string), need not produce the same similarity value (e.g., $S(p_{11}) \neq S(t^L_{211})$, although $p_{11} = t^L_{211}$).

•The stepwise process may introduce new—but distinct—errors into the error string. Indeed, when the error substring e_i is composed of more than one morpheme, the repair itself will comprise more than one morpheme, and the process of matching by stages may produce elements of the set R'_i, but made up of morpheme elements whose position in the string may be syntactically incorrect. This would introduce errors of the type described in *Problem 2*.

It can now be seen that *Problem 2* and *Problem 3* need to be dealt with in different ways. If, in the repair process described in *Problem 1* we accomplish our objective, where no new errors are introduced, then in the stepwise matching process it becomes necessary to solve both *Problem 1* and *Problem 2*. In other words, any repairs made must be both syntactically and lexically correct in their environment.

3.2. Incompatibility between Morphemes. As mentioned above, not all morpheme types are valid for the repair of any given error string. For a repair to be correct, the newly-generated string of lexical components has to satisfy the basic structure of inorganic chemical names as recognized by the working nomenclature. The morpheme or morpheme set employed in the repair of an error string must be of a type that guarantees the (syntactic) structural integrity of the new string.

Take the string $C \equiv \{t_1 t_2 e t_3 t_4\}$, in which e is an incorrect substring; let us suppose that r is a repair morpheme for the substring e and produces the new string $C' \equiv \{t_1 t_2 r t_3 t_4\}$, which is lexically error-free and syntactically correct.

For r to be a repair that is syntactically correct for its environment, the following must be true:

1. the combination of the leading morpheme t_2 and the repair morpheme r must be syntactically correct, as must
2. the combination of the repair morpheme r and its trailing morpheme t_3.

At this point it becomes necessary to introduce the following concept of incompatibility between morphemes: it may be said that two morphemes $t_1, t_2 \in \Sigma_T$ are incompatible if the string produced by the concatenation of the two corresponding lexemes violates the structure of inorganic names; in other words, if the combination of the two violates the syntax of the defined grammar.

As a language's syntactic structure is generally based on morpheme type or category, it would be more correct to redefine morpheme incompatibility as a function of morpheme type.

Let us define incompatibility as a function of the form

$$I(t_1,t_2) = \begin{cases} 1 & \text{if there is incompatibility} \\ 0 & \text{is there is no incompatibility} \end{cases}$$

which may be defined thus as two morphemes $t_1, t_2 \in \Sigma_T$ are said to be incompatible and $I(t_2,t_2) = 1$ if concatenation of the two gives a syntactically incorrect string $t_1 t_2$.

Analysis of the above definition and of the syntactic structure of chemical names shows that incompatibility between morpheme types is not symmetrical, that is $I(t_1,t_2) \neq I(t_2,t_1)$, and neither is the concatenation of the strings.

The concept of incompatibility could now be generalized for the entire string representing an inorganic name: a string C representing an inorganic substance and composed of n substrings $C \equiv \{c_1 c_2 \ldots c_n\}$ corresponding to morphemes defined in the grammar $c_1, c_2, \ldots, c_n \in \Sigma_T$ may be described as incompatible if there is any incompatibility between its component substrings, or the following:

Given a string $C \equiv \{c_1 c_2 c_3 \ldots c_n\}$, its global incompatibility $GI(C)$ is given by

$$GI(C) = I(\lambda,c_1) \vee I(c_1,c_2) \vee \ldots \vee I(c_{n-1},c_n) \vee I(c_n,\lambda) \quad (1)$$

where λ represents the empty string and where it can be seen that for a string composed of n morphemes, $n + 1$ calculations would be necessary to discover all possible morpheme incompatibilities. This is because of the need to check for the presence of a specific morpheme at the start and the end of the string. For example, suffix-morphemes should never appear at the beginning of a string, just as prefix-morphemes ought never to appear at the end of a string representing an inorganic substance.

From the above equation, it can be seen that the problem of making a repair that produces a new, syntactically-correct string where $GI(C) = 0$ (*Problem 3*) is easily reducible to obtain syntactically-correct repairs in a given context, $\sum_{i=1}^{i=n+1} I(c_{i-1},c_i)$ (*Problem 2*), as discussed in the previous section. In short, the problem is reduced to achieving a repair of the nature $I(t_2,r) = I(r,t_3) = 0$, where r is the repair string and t_2, t_3 are the leading and trailing morphemes.

3.3. Error String Length. The size of the error string is another factor that affects the number of matching operations required to achieve repair. According to the literature,[17−20] over 80% of lexical errors are due to typing and/or phonetic mistakes (ignorance of the language), where the error is caused by insertion, deletion, or transposition of a character, so we might propose, given an incorrect substring e of length $|e|$, the repair strings r_i will be those of length $|r_i|$ which satisfy

$$|e| - 1 \leq |r_i| \leq |e| + 1 \qquad (2)$$

thus enabling us to limit the number of comparisons to be made.

The limit proposed in the above equation would only be valid for a simplification of the problem in which the error string e was assumed to be composed of just one morpheme or terminal symbol. Unfortunately, this is not the case here; for instance, in the string *sodium bosulate* there is an error string *bosul*, formed by the morphemes *bo* (due to the substitution of the character "*i*" by "*o*") and *sul* (due to omission of the letter "*f*"). In the case of the first expression, the length of the repair strings would lie in the interval [4,6]; this means that the solution *bi* would never be found, thereby making it impossible to achieve the repair *bisulf*.

One could, to solve this problem, expand the range determined by eq 2 by one or more units, but we would always come across situations in which the equation would not be satisfied. Take, for instance, the string *tetarhydorgenfosfate*, where the error substring *tetarhydorgen* has a length of 13 and eq 2 would have to be modified thus $|e| - 8 \leq |r_i| \leq |e| + 8$ for the string *tetra* to be successfully matched. This would lead to an excessive increase in the number of comparisons to be made in other situations where they would not be necessary.

We can see that the problem lies in the lower limit of the equation, since an error string of length $|e|$ may be composed of much smaller error morphemes, $|c_i| \ll |e|$, for $e \equiv \{c_1c_2...c_n\}$; this would be the case for the string *sodium ortodosulate*, where the error string *ortodosul* is nine characters long, owing to the three morphemes *ortho*, *di*, and *sulf*, whose lengths are 5, 2, and 4, respectively.

This problem arises because we recognize the possible existence of more than one erroneous morpheme in the error string, as the syntactic structure of inorganic names generally lacks delimiters between significant fragments. It is also due to performing the matching process on the terminal symbol set Σ_T rather than on the nonterminal set Σ_N. For this reason, given an error substring e of length $|e|$, matching should be carried out with all morphemes that satisfy the expression

$$\min(|r_j^*|) \leq |r_i| \leq int(|e| + f \times |e| + 1) \qquad (3)$$

where $|r_j^*|$ represents the length of elements in the set of compatible morphemes that might repair the error substring, and f is a variable factor with a value in the interval [0, 1], which we shall call the *error factor*.

If we take the error substring e to be composed of just one morpheme, then f will represent the number of deletions or insertions that might be the cause of the error, for each character present.

3.4. Similarity Threshold. Another factor that can intervene in the matching process it the similarity threshold, taking similarity to mean the degree of likeness existing between two strings.

If, as a measure of similarity, we use the edit or Levenshtein distance,[7,21] k, then as the value of k rises, the similarity between two strings decreases, and vice versa. Two strings are said to be identical when $k = 0$.

In the introduction and in the previous paper in this issue,[8] the authors made reference to studies aimed at the calculation of edit distance for exact and approximate matching and the modifications proposed to "penalize", with different weightings, differences due to insertion, deletion, and substitution of characters (transposition being usually regarded as a deletion and an insertion).[22,23]

We have seen that, initially, the measurement of edit distance does not take into account the length of the error string or that of the repair pattern or string. For example, for the strings "*o*" and *drogen*, the patterns *oxo* and *hydrogen* each have an edit distance of 2, although their string lengths are very different. While from an operational point of view one may speak of equal similarity (the same number of operations needed to convert the error string into the pattern), conceptually things are different. For the measurement of similarity it would be more convenient to change this metric to

$$S = 1 - \frac{d(error,pattern)}{|repair|} \qquad (4)$$

where S is similarity value, $d(error,pattern)$ is the edit distance or number of operations required to change the error string into the pattern, and *repair* is the maximum difference between the length of the error string to be repaired by the pattern and that of the pattern itself.

Applying this to the above example, we would get values for S of $1 - {}^2/_3 = 0.33$ and $1 - {}^2/_9 = 0.77$, respectively, more in keeping with the term "similarity". Using this model to compare two strings t and p, each character coinciding with both strings would return a value of ${}^1/_{|r|}$ to the similarity figure. Similarity is equal, then, to the sum total of the figures obtained from the process of matching the two strings, for which one of the algorithms mentioned in the bibliography could be used.[22] In this calculation of similarity, errors of insertion, deletion, and substitution are all penalized, since in no event do they contribute to the overall computation of similarity.

Given the features of our problem, where the error string may include one or more incorrect morpheme, not all morphemes are compatible for the repair of the error string, etc., and given the proposed model in which the repair process deals, stage-by-stage, with significant "parts" of the error string, some redefinition of the concept of string similarity becomes necessary.

Let us examine a comparison of two strings C and T, not necessarily of the same length, where the following situations may arise:

1. $c_i = t_i$, where the character i is present in both
2. $c_i \neq t_i$, with no matching of characters
3. $c_i = t_{i+1}$, $c_i = t_{i-1}$, a shifting of characters
4. $c_i = t_{i+d}$, $c_i = t_{i-d}$, multiple shifting of characters In such situations there are instances of insertion, deletion, substitution, and multiple transposition of characters as well as cases where transposition involves characters belonging to more than one morpheme (*overlapping*). Here, for example, the string *teatr* would be classed as an incorrect version of the pattern *tetra*, with multiple transposition of the letter "*a*".

For the calculation of similarity, the following assumptions were made for the matching process:

1. coinciding characters return $+1/L$
2. absent characters return $-1/(2\times L)$
3. shifted characters return $+1/(2^d\times L)$ where d is the number of shifts involved, and L is the length of the matching process working string, namely the number of characters involved in the matching, i.e.,

$$L = \max(|repair|, |pattern|) \tag{5}$$

or the maximum difference between the length of the part of the error substring being repaired and that of the pattern.

In the calculation of similarity, it is useful to maintain a record of which part a specific pattern is capable of repairing. Due to insertion of characters, the error string may be of a different length from the repair pattern, though only comprising one morpheme. We may thus examine characters from the error string that have been shifted by more than one unit relative to their position in the pattern. The further they are shifted, the less these characters contribute to the total similarity calculation. When shifting is due to multiple character insertion, its consideration actually lowers the total similarity figure, since the denominator L (the repair length) increases with displacement. In such situations, it might be better to consider a deletion instead of multiple shifting, in order to produce greater similarity.

For instance, in the error string *oxsulafte*, for which the morpheme *oxo* would be a valid repair, it is more convenient to consider that there has been omission of the final character "*o*" of the string *oxo* than to say that there has been an extra insertion (and therefore, multiple shifting) of the string *sulaft*. However, for the error string *hydrgeon*, a valid repair for which would be *hydrogen*, it is more convenient to say that there has been a multiple shift of the character "*o*" than to deal with a deletion and an insertion.

The similarity values obtained using the proposed equation range from -0.5 to 1, since the deletion of characters from the pattern is "penalized" (decreased) by half the value with which the presence of matching characters is "rewarded" (increased).

4. APPROXIMATE MATCHING ALGORITHM

The proposed algorithm is based on a stepwise method to compute the best repair that a pattern may perform on all or part of an error string. The process is carried out by each of the patterns that satisfies the working restrictions; only those repairs which satisfy a previously-established similarity threshold will be considered acceptable.

Given an error string $C \equiv \{c_1\ c_2\ c_3\ ...,\ c_n\}$ and a pattern or morpheme set $T \equiv \{t_1t_2t_3\ ...,\ t_m\}$, the matching algorithm will take the following steps:

Preprocessing. In this step, each character t_i from the pattern is compared with each of the characters c_j from the error string, giving the working matrix M. We have called this the *adjacency matrix* and it is of dimension $m \times n$. Each element $M(i,j)$ is set to 1 if the character i from the pattern is the same as j from the error string. If we say that the characters forming the error string and the pattern are nodes in a tree where the arcs represent the relative positions between characters in the string, then the values $M(i,j)$ of the adjacency matrix will represent arcs between the nodes of the subtree of the pattern and coinciding nodes of the error string subtree (see Figure 1).[10,24]

Processing. The matching algorithm works with the information held in the adjacency matrix in the following way:

Step 1. The adjacency matrix M is scanned row by row, and for each occurrence of $M(i,j) = 1$, the pattern character i is "*anchored*" at the position j of the error string. This means that the pattern character $T(i)$ is now positioned over the character $T^*(i) = i + j - 1$ of the error string. Thus, $T^*(i)$ represents the point j on the error string at which the pattern character i is positioned during the repair process. The value of $T^*(i)$ may be negative, zero, or positive, that is, greater than the size of the error string.

Step 2. The array T^* is compared with the error string at the interval $[C(T^*(1)), C(T^*(m))]$, that is, along the error string over the interval delimited by the length of the pattern, and the following are recorded:

1. Matching characters, where $T^*(i) = C(T^*(i))$.
2. Characters from the pattern not present in the error string, namely all those where $T^*(i)$ does not appear in the interval $[C(T^*(1)), C(T^*(m))]$.
3. Characters shifted within the interval determined by the pattern length, that is, those values of $T^*(i)$ found in the interval $[C(T^*(1)), C(T^*(m))]$ and which have not been previously considered. Single and multiple shifts are dealt with in this step.

In the course of this step a new array is built, of size m and called *Image*. Each element of *Image* stores the position j of the error string in which the pattern character i is found, or zero if none is found.

Step 3. Now initial similarity (S_p^1) is calculated. It is assumed that the repair length will be the same as that of the pattern, so the denominator employed to compute the figures will be m.

Step 4. In this step, the following is performed:

1. If the number of missing characters calculated in *step 2* is zero, skip to *step 5.1*.
2. Otherwise, increment the working interval by one unit at either end and examine the pattern characters shifted one place left and right of the area of the error string checked in the previous step. In other words, perform a matching

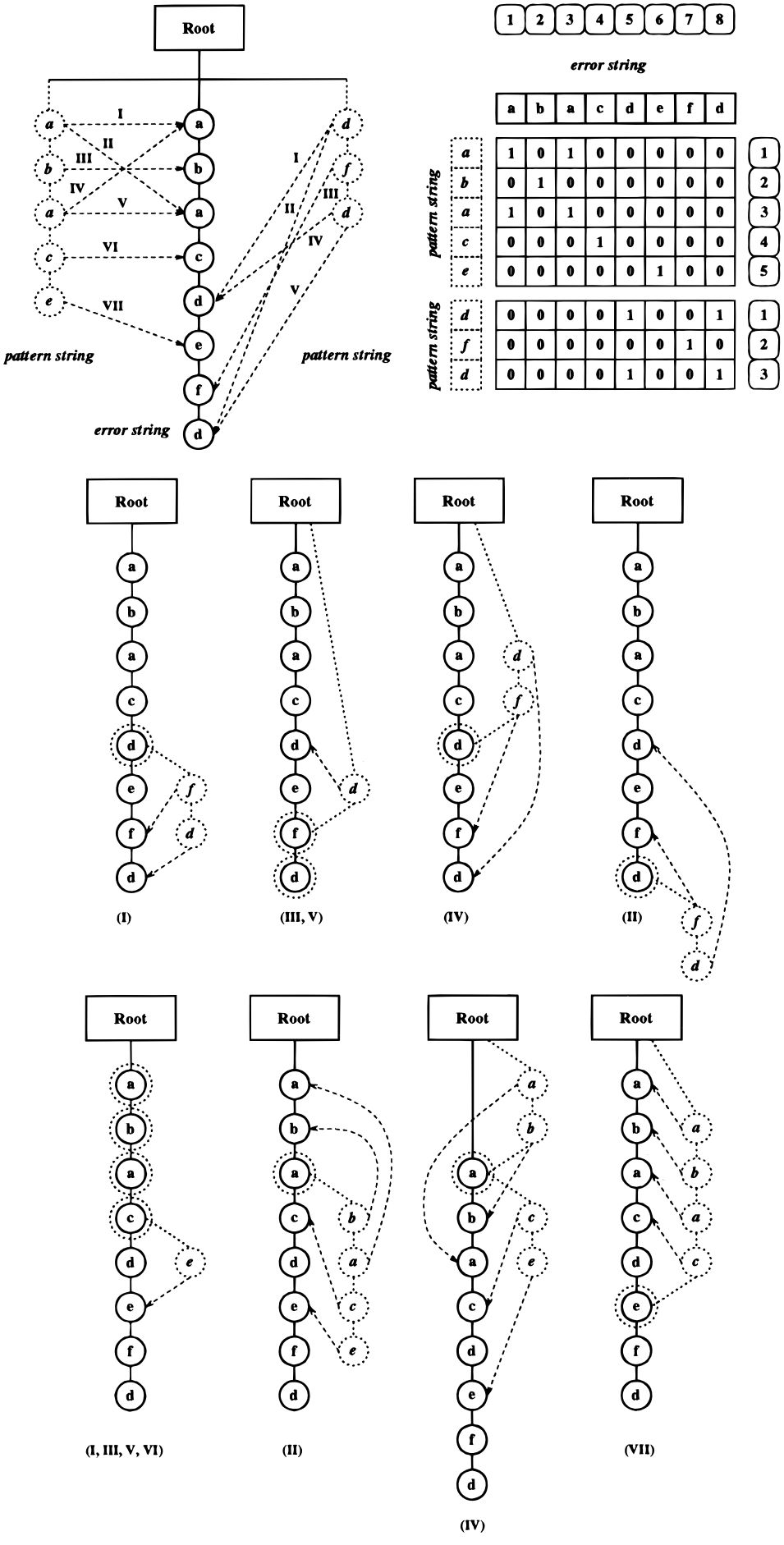

Figure 1. An illustration of the matching process.

Table 1. Example of the Repair of the String *abacdefd*

	anchor point	matching (T^*)	figures	similarity	image	tree
		Repair Process for the Pattern *dfd*				
yes	$M(1, 5)$	{5,6,7}	$\frac{1}{4}, \frac{1}{8}, \frac{1}{8}$	0.500	{5,7,8}	I
yes	$M(1, 8)$	{8,9,10}	$\frac{1}{6}, \frac{1}{24}, \frac{1}{192}$	0.214	{8,7,5}	II
yes	$M(2, 7)$	{**6,7,8**}	$\frac{1}{8}, \frac{1}{4}, \frac{1}{4}$	0.625	{**5,7,8**}	**III**
yes	$M(3, 5)$	{3,4,5}	$\frac{1}{6}, \frac{1}{48}, \frac{1}{192}$	0.193	{8,7,5}	IV
no	$M(3, 8)$	{6,7,8}	$\frac{1}{8}, \frac{1}{4}, \frac{1}{4}$	0.625	{5,7,8}	V
		Repair Process for the Pattern *abace*				
yes	$M(1, 1)$	{**1,2,3,4,5**}	$\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{12}$	0.750	{**1,2,3,4,6**}	**I**
yes	$M(1, 3)$	{3,4,5,6,7}	$\frac{1}{7}, \frac{1}{28}, \frac{1}{56}, \frac{1}{28}, \frac{1}{14}$	0.304	{3,2,1,4,6}	II
no	$M(2, 2)$	{1,2,3,4,5}	$\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{12}$	0.750	{1,2,3,4,6}	III
yes	$M\{3, 1)$	$\{-1,0,1,2,3\}$	$\frac{1}{128}, \frac{1}{32}, \frac{1}{8}, \frac{1}{32}, \frac{1}{64}$	0.211	{3,2,1,4,6}	IV
no	$M(3, 3)$	{1,2,3,4,5}	$\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{12}$	0.750	{1,2,3,4,6}	V
no	$M(4, 4)$	{1,2,3,4,5}	$\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{12}$	0.750	{1,2,3,4,6}	VI
yes	$M(5, 6)$	{2,3,4,5,6}	$\frac{1}{12}, \frac{1}{12}, \frac{1}{12}, \frac{1}{12}, \frac{1}{6}$	0.500	{1,2,3,4,6}	VII

operation on the interval $[C(T^*(1) - 1), C(T^*(m) + 1)]$ and calculate a new similarity value S_p^2. In this calculation we assume that the size of the repair is now equal to $m + 1$ or $m + 2$, depending on whether similarity has been observed at one or both ends of the new interval.

In this process, only pattern characters which have not previously checked out as similar are examined, that is those elements whose $Image(i) = 0$. So only the rows i of the adjacency matrix in the columns where $T^*(1) - 1 \geq 1$ and $T^*(m) + 1 \leq n$ are examined.

Step 5. Values for S_p^1 and S_p^2, are compared and

1. If $S_p^1 > S_p^2$, then:

(a) The final similarity is $S_t = S_p^1$.

(b) Values of S_t and *Image* are stored temporarily.

(c) Skip to *step 7*.

2. If $S_p^1 \leq S_p^2$, then:

(a) $S_p^1 = S_p^2$.

(b) Move on to *step 6*.

Step 6. As long as there is a value of $Image(i) = 0$, that is, if there is a character in the pattern that has not yet been matched against the error string, *step 4* is repeated, each time widening the interval by one unit. Consequently, in the recalculation of S_p^2 the size of the denominator (the repair length) will increase by one or two units.

Step 7. At the end of this process we should have the highest similarity value S_t along with its *Image*, corresponding to the best repair of the error string with respect to its anchor point or positioning on the pattern. It should be borne in mind that different anchor points call for the consideration of different error types.

The value of S_t is then compared with that of the previous anchored value, and the higher of the two values is kept, along with its corresponding *Image*.

Step 8. Jump back to *step 1*, choosing a new adjacency matrix value of $M(i,j) = 1$ in order to introduce a new anchor point and proceed from *step 2* through *step 7*; this process is repeated until no further anchoring is possible (all elements where $M(i,j) = 1$ have been examined).

It should be pointed out that not all values for $M(i,j) = 1$ are examined when computing similarity, since that would give rise to redundant calculations and adversely affect the efficiency of the repair process. In order to perform only the essential matching operations, we proceed thus:

When an anchor point has been established for a value of $M(i,j) = 1$ from the adjacency matrix, a value of *zero* is placed in

1. The adjacency matrix element $M(i,j)$ chosen for anchoring.

2. Every element $M(h,k)$ where the value of $T^*(h) = k$ in the positioning array T^* obtained for this position.

In this way, the repair in which the pattern character i is positioned over the error string character j is only considered for one matching operation.

Termination. At the end of the *processing stage*, the maximum value for similarity S_t has been found, together with the corresponding *Image* for the best matching of a pattern T with an error string C. During this stage we have obtained

- The repair interval within which the pattern can repair the error string; the interval is defined thus

$$[\min(Image(i)), \max(Image(i))]$$

- The similarity value for the repair.
- Based on the information stored in the *Image* data structure, the deleted, inserted and/or, transposed characters responsible for the lexical error.

A value of $Image(i) = 0$ reports the omission of the character i from the text pattern; when $Image(i) = i + \min(Image(i)) - 1$ the process has found a character which coincides with the text pattern; a value of $Image(i) = i + \min(Image(i)) - 1 + d$ reports on the transposition of size d of a character from the text pattern.

- New (and possibly empty) error substrings situated to the left and right of the repair interval for the error string. That is, the strings

$$e^L \equiv \{c_1\, c_2 \ldots c_{\min(Image(i))-1}\}, \text{ and}$$

$$e^R \equiv \{c_{\max(Image(i))+1}\, c_{\max(Image(i))+2} \ldots c_n\}$$

on which it would be ncessary to perform new matching operations to find total repairs for the error strings.

5. AN EXAMPLE OF A REPAIR

Let us examine a generic example of the use of the proposed algorithm, one which allows us to observe its application to any approximate string matching problem and which illustrates total or partial repair of the error string.

Suppose we have a string composed of the substrings *abace* and *dfd* and which contains an error which the first *d*

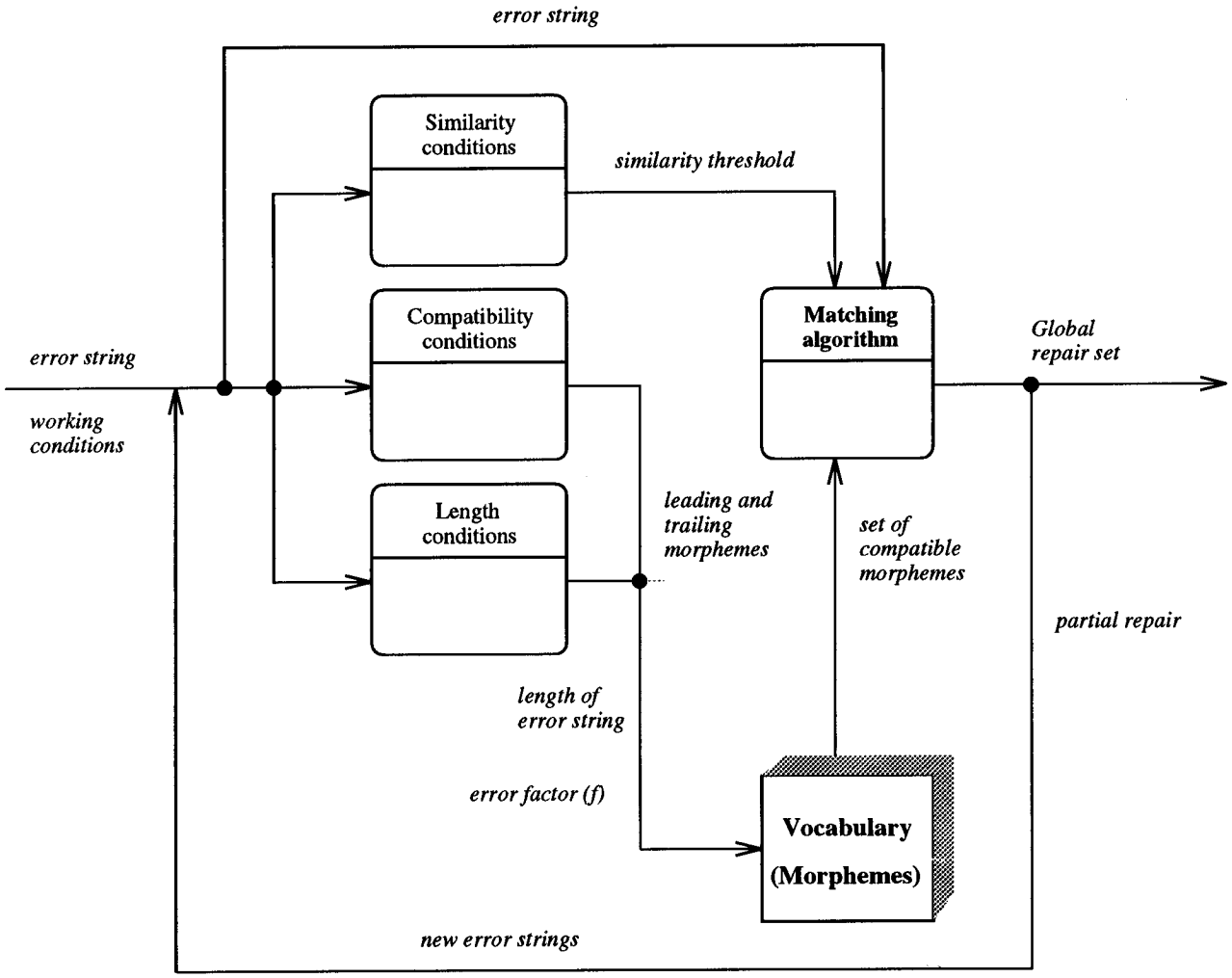

Figure 2. Model for the repair of lexical errors.

and the character *e* have been transposed, giving an error string *abacdefd*. There are now two incorrect substrings, so two repairs are required.

Let us follow the algorithm step-by-step and, for the sake of simplicity, only carry out comparisons with the strings *abace* and *dfd*.

Preprocessing. Each error string character is compared with each of the characters in the patterns which might successfully effect repair; that is, with the set of morphemes which satisfies the working restrictions, the adjacency matrix *M* is thus obtained.

Figure 1 shows the adjacency matrix for the patterns in this example: the string *abace* and the string *dfd*. This matrix is a simple form of representing the tree for suffixes or relationships between the error string and the pattern, also shown in Figure 1.

Each of the trees displayed in the figure shows a different positioning of characters from the pattern on those of the error string and the matching necessary for the calculation of similarity between the two strings. Each tree reports on the error type (deletion, insertion, or transposition) responsible for the lexical error.

Processing. Pattern by pattern, we choose each value of the adjacency matrix where $M(i,j) = 1$. For each anchor point $M(i,j)$, a different positioning is obtained for the pattern relative to the error string; this is defined by the array T^*, and from it we can successively calculate the best figure for similarity to the pattern characters.

Table 1 contains each of the anchor points set for the two example patterns; the Table shows the chosen position $M(i,j)$, the positioning array T^*, the similarity figures for each character, the similarity total, the *Image* array obtained, and finally, the tree corresponding to the matching operation shown in Figure 1.

It may be observed that, as previously mentioned, not all anchor points are established, since they are unnecessary and would produce redundant results. It may also be seen that the *Image* data structure holds information about the error type which the matching algorithm considers responsible for the lexical error. For example the *Image* data structure obtained for the pattern {*dfd*}, {*5,7,8*}, reports that the pattern can repair the text characters in the interval [5,8] and that there has been an insertion error at position 6 of the error string. If matching had been carried out using the pattern {*bfd*}, the best repair would have been *Image* $\equiv$ {0,7,8}, reporting that the pattern could repair the text string in the interval [7,8], and that there was an error of deletion of the character *b* in the text.

It is clear from this last example that, because of the proposed model for the computing of similarity, it was more convenient to consider the omission of the character *b* from the text pattern than to consider that there had been an error of multiple transposition of that character, which would have led to a lower similarity value. It is not then seen as wise to work with over-large repair patterns, which in turn means

that not all possible repairs will be found for a text string containing more than one incorrect morpheme.

6. DISCUSSION

This article has analyzed some of the problems that may arise in the repair of lexical errors in inorganic chemical names, and a strategy has been proposed, as shown in Figure 2. This strategy is based on a series of preestablished working conditions where the following are specified:

•*Length*. The percentage of error to be dealt with in the error string, that is, the maximum number of characters omitted or inserted in the error string.

•*Compatibility*. The definition of the syntax of the grammar defining the language. This is a simple definition based on morpheme category or type and defines how these may be introduced into a string representing an inorganic chemical name, as will be described for inorganic nomenclatures in the next article in this series.[25]

•*Similarity*. A minimum acceptable threshold for a repair to be valid. This threshold will be limited to the cardinality of the set of possible repairs (R^*).

Under these conditions, the proposed approximate matching algorithm would be capable of finding repairs for all or part of an error string; in the case of partial repairs, the newly-generated error strings would be sent back through the repair process until total repair was achieved (Figure 2).

The matching algorithm on which the proposed repair model is based could be applied to any approximate string matching problem, particularly in cases where the string is a sentence from a language which normally contains no delimiters to separate different morphemes or words, such as is true of organic and inorganic chemical nomenclatures.

A new schema has been proposed for the calculation of similarity between strings. Using different weights, this schema "penalizes" the deletion, insertion, and transposition of characters, taking into account the possibility of multiple transpositions of characters and the size of the repair that the pattern may perform on the error string. It is extremely sensitive, producing similarity values that provide a high degree of discrimination between different repairs.

The repair process generates a data structure, *Image*, that provides a great deal of information on what the algorithm has decided are the causes of the lexical error, and this could be a powerful tool for use in systems demanding a high level of man-machine communication and where the control and repair of lexical errors is required.

The next article in this series describes the application of the proposed model to the repair of lexical errors that may arise in the recognition of inorganic chemical names in the most commonly-used nomenclatures (IUPAC, Stock, and Conventional).[25]

REFERENCES AND NOTES

(1) Tarhio, J.; Ukkonen, E. Approximate Boyer-Moore String Matching. *SIAM J. Comput.* **1993**, *22*(*2*), 243−260.
(2) Cole, R. Tight Bounds on the Complexity of the Boyer-Moore String Molecular Algorithm. Courant Institute: New York University, Technical Report, New York, 1991.
(3) Apostolico, A.; Giancarlo, R. The Boyer-Moore-Galil String Searching Strategies Revisited." *SIAM J. Comput.* **1986**, *15*(*1*), 98−105.
(4) Boyer, R.; Moore, S. A Fast String Matching Algorithm. *J. CACM* **1977**, *20*, 762−777.
(5) Gali., Z. On Improving the Worst Case Running Time of the Boyer-Moore String Matching Algorithm. *J. CACM* **1979**, *22*, 505−508.
(6) Marzal, A.; Vidal, E. Computation of Normalized Edit Distance and Applications. *IEEE Trans. Pattern Anal. Machine.* **1993**, *15*(*9*), 926−932.
(7) Levenshtein, V. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *soDiet. Phys. Dokl.* **1966**, *6*, 126−136.
(8) Luque Ruiz, I.; Cruz Soto, J. L.; Gómez-Nieto, M. A. Error detection, recovery and repair in the translation of inorganic nomenclatures. 1. A study of the problem. *J. Chem. Inf. Comput. Sci*., **1995**, *35*, xxxx.
(9) Hippe, Z. *Artificial Intelligence in Chemistry*; Elsevier: Amsterdam, 1991.
(10) Ukkonen, E.; Wood, D. Approximate String Matching with Suffix Automata. *Algorithmica* **1993**, *10*(*5*), 353−364.
(11) Jokinen, P.; Tarhio, J.; Ukkonen, E. Comparison of Approximate String Matching Algorithms. Helsinki University, Department of Computer Science, Technical Report: A-1991-7, 1991.
(12) Ukkonen, E. Finding Approximate Patterns in Strings. *J. Algorithms* **1985**, *6*, 132−137.
(13) Fischer, C. N.; Leblanc, R. J. *Grafting a Compiler*; The Benjamin Cummings Publisher: 1988.
(14) Lewis, P. M.; Rosenkrantz, D. J.; Stearns, R. E. *Compiler Design Theory*; Addison-Wesley: 1976.
(15) Trembay, J. P.; Sonrenson, P. G. *The Theory and Practice of Compiler Writing*; McGraw-Hill: 1985.
(16) Aho, A. V.; Ullman, J. D. *The Theory of Parsing, Translation, and Compiling. Volumen I: Parsing*; Prentice-Hall: 1972.
(17) Bratchikov, I. L.; Diallo, M. M. Elementary and Structural Language Errors and Methods for their Handling. *Automatic Control Comput. Sci*. **1991**, *25*(*4*), 81−84.
(18) Segal, J.; Ahmad, K.; Rogers, M. The Role of Systematic Errors in Developmental Studies of Programming Languages Learners. *J. Educ. Comput. Res*. **1992**, *8*(*2*) 129−153.
(19) Pollock, J. J. Spelling Error Detection and Correction by Computer: Some Notes and Bibliography. *J. Documentation* **1982**, *38*(*4*), 282−291.
(20) Pollock, J. J.; Zamora, A. Collection and Characterization of Spelling Errors in Scientific and Scholarly Text. *J. Am. Soc. Inf. Sci*. **1983**, *34*(*1*), 51−58.
(21) Chang, W. I.; Lampe, J. Theoretical and Empirical Comparison of Approximate String Matching Algorithms. *Technical Report*: *91*−*653*, CA, 1991.
(22) Kukich, K. Techniques for Automatically Correcting Words in Text. *ACM Comput. Surveys* **1992**, *24*(*4*), 377−439.
(23) Bertossi, A. A.; Luccio, F.; Lodi, E.; Pagli, L. String Matching with Weighted Errors. *Theoretical Comput. Sci*. **1990**, *73*(*3*), 319−328.
(24) Jacquet, P.; Szpankowski, W. Autocorrelation on Words and Its Applications. Analysis of Suffix Trees by String Ruler Approach. *J. Combinatorial Theory* **1994**, *66*(*2*), 237−269.
(25) Luque Ruiz, I.; Cruz Soto, J. L.; Gómez-Nieto, M. A. Error detection, recovery and repair in the translation of inorganic nomenclatures. 3. An Error Handler. *J. Chem. Inf. Comput. Sci*. Submitted for publication, 1995.

CI950222U