# FURY : Fuzzy unification and resolution based on edit distance

David Gilbert and Michael Schroeder
School of Informatics, City University
Northampton Square, London, EC1V 0HB, UK
{drg,msch}@soi.city.ac.uk

## Abstract

*We present a theoretically founded framework for fuzzy unification and resolution based on edit distance over trees. Our framework extends classical unification and resolution conservatively. We prove important properties of the framework and develop the FURY system, which implements the framework efficiently using dynamic programming. We evaluate the framework and system on a large problem in the bioinformatics domain, that of detecting typographical errors in an enzyme name database.*

## 1. INTRODUCTION

Uncertainty and fuzziness play an important role in reasoning and occur on various levels: Uncertainty and fuzziness of the concepts in question, of the rules and domain knowledge, or of the reasoning process itself. In contrast to fuzzy logic and uncertainty in AI which deal mainly with the first two aspects, we address with the last and focus on fuzzy unification. In classical unification predicates unify or they do not; we introduce a degree of unification ranging from a complete match (degree 0) as in classical unification to a complete mismatch (degree 1). Arcelli, Formato, Gerla have developed a general abstract framework for fuzzy unification, quotient unification and unification as negotiation [3]. In this paper, we develop an alternative approach for fuzzy unification and resolution based on edit distance. The concept of edit distance has a well established history dating back to the 60s and 70s [9, 15, 1] and is still widely used, for example, in bioinformatics to comparing genomic sequences. The edit distance between two strings $A$ and $B$ is defined as the minimal number of delete, add, and replace operations to convert $A$ into $B$. The basic principle of edit distance is well-understood, but to employ it for fuzzy unification there are three requirements: First, a normalisation is required to be able to compare strings independent of their size, otherwise relative many mismatches of two small strings is possibly rated better than only a few on long strings, which is counter-intuitive. Second, the definition of edit distance has to be extended to deal with general tree structures representing the predicates and terms to be compared. Third, for compatibility reasons fuzzy unification should be an extension of classical unification.

Once such a fuzzy unification mechanism is in place, resolution needs to be extended. In most approaches to fuzzy resolution the conjunction of body literals is mapped to the minimum operation on the corresponding fuzzy values. However, since our approach does not attempt to capture fuzzy concepts, but instead fuzzy unification, a set of different operations for resolution is useful. An alternative to the minimum operation is multiplication or average. Both have the advantage of being accumulative integrating all values in the rule body under consideration. Furthermore, minimum and multiplication exhibit the same intuitive property that failing goals are translated to a fuzzy value of 0, which is propagated.

Potential applications of our method include data cleaning (for example detecting typographical errors in database entries), and debugging logic programs (e.g. detecting typological errors in predicate names and arguments, missing or exchanged arguments). Embury et al. [2] for example tackle the problem of conflict detection for integration of taxonomic data sources. They specify domain knowledge and consistency rules in Prolog and search for inconsistencies, which are often based on different spellings and naming conventions. For such problems, our meta-interpreter FURY implementing fuzzy unification and resolution may be used directly. Another general problem, where our approach is useful, concerns data cleaning: Given two databases, one with accurate data and another one with less reliable data, our system can compare the reliable entries to the best hits of the unreliable source and subsequently clean up the latter.

The paper is organised as follows: First, we introduce and review some basic definitions. Next, we introduce fuzzy unification and resolution and show important properties of our approach. Finally, we describe an efficient implementation FURY , discuss its complexity and evaluate it on a problem in bioinformatics.

## 2. DEFINITIONS AND BACKGROUND

Symbols are strings, where a string is either the empty string $\epsilon$ or a string $a.A$, where $a$ is a character and $A$ is a string. $|A|$ denotes the length of $A$.

Let $V$ be a set of variable names, $F$ a set of function symbols and $P$ a set of predicate symbols. The set of terms is defined inductively. Every variable $x \in V$ is a term. Let $f \in F$ be a function symbol of arity $n$ (if $n = 0$, $f$ is also called a constant) and $t_1, \ldots, t_n$ terms, then $f(t_1, \ldots, t_n)$ is a term. Additionally, we introduce a unique, empty term $\epsilon$. Nothing else is a term. Let $p \in P$ be a predicate symbol of arity $n$ and $t_1, \ldots, t_n$ terms, then $p(t_1, \ldots, t_n)$ is an atomic formula. Literals are all atomic formula $A$ and their negation $\neg A$. For convenience, we define $\overline{L} = \neg A$ if $L = A$ and $\overline{L} = A$ if $L = \neg A$ for an atom $A$. The set of formulae is defined recursively. Every atomic formula is formula. If $F$ and $G$ are formulae, and $x$ is a variable, then $F \wedge G$, $F \vee G$, $\neg F$, $\forall x\ F$, $\exists x\ F$ are formulae. Nothing else is a formula.

Any formula can be equivalently transformed into Skolem normal form, where all existential quantifiers are followed by universal quantifiers followed by a quantifier free formula in conjunctive normal form (CNF). A formula is in CNF if it has the form $(L_{11} \vee \ldots \vee L_{mn_1}) \wedge \ldots \wedge (L_{n1} \vee \ldots \vee L_{nm_n})$, where the $L_{ij}$ are literals. Each of the conjuncts in a CNF formula is called clause and usually written as a set omitting the $\vee$ connective. A clause is called Horn or definite if it contains exactly one positive literal. A clause is negative if it contains only negated atoms.

In order to check whether a goal is entailed by a Horn formula, we employ unification and resolution.

**Definition 1** *Unificator [12]. A substitution is a replacement of variables by terms. A substitution $sub$ is a unificator of two literals $L, L'$ if $Lsub = L'sub$. A unificator $sub$ of $L, L'$ is the most general unificator (MGU) of $L, L'$ if for every other unificator $sub'$ of $L, L'$ there is a substitution $s$, such that $sub' = subs$.*

The MGU can be computed using Robinson's unification algorithm [12].

**Example 1** *The predicates $address(x)$ and $address(Northampton)$ unify and the MGU is $[x/Northampton]$. For various reasons all of the following predicates do not unify: $address(x)$ and $address(f(x))$, because $x$ occurs in $f(x)$, which would lead to a circular substitution; $address(Northampton)$ and $address(10, Northampton)$ as the predicates do not have the same number of parameters; $address(Northampton)$ and $adresse(Northampton)$ as the predicate names slightly mismatch; $address(Northampton)$ and $address(Northhampton)$ as the terms slightly mismatch.*

**Definition 2** *SLD Resolution [10]. Let $N$ be a negative clause and $\neg A \in N$. Let $D$ be definite clause and $A' \in D$. Let $sub$ be the MGU of $A$ and $A'$. Then $((N - \{\neg A\}) \cup (D - \{A'\}))sub$ is called resolvent of $N$ and $D$. Let $N$ be a negative clause and $\mathcal{D}$ be a set of definite clauses. An SLD-derivation of $\mathcal{D} \cup \{N\}$ consists of a sequence $N = N_0, N_1, \ldots$ of negative clauses, a sequence $D_1, D_2, \ldots$ of clauses in $\mathcal{D}$ and a sequence of MGUs $sub_1, sub_2, \ldots$ such that $N_{i+1}$ is a resolvent of $N_i$ and $D_{i+1}$ using $sub_{i+1}$. An SLD-refutation is an SLD-derivation, where the last negative clause is empty.*

Resolution is fundamental, as it allows us to efficiently check satisfiability of a formula:

**Theorem 1** *A formula is satisfiable iff there is an SLD-refutation [10].*

**Example 2** *Consider the clauses $\{address(x, y), \neg id(x, z), \neg street(z, y)\}$, $\{id(007, Bond)\}$, $\{street(007, Bondstreet)\}$ and the goal $address(Bond, x)$ or expressed as a clause $\{\neg address(Bond, x)\}$. There is an SLD-refutation for the definite clauses and the goal clause with the variable substitution $[x/Bondstreet]$.*

## 3. FUZZY UNIFICATION AND RESOLTUION

In this section, we set out to broaden the principles of unification and resolution to encompass fuzzy matches of predicate and function symbols and to deal with missing arguments. We need a comparison measure to define how similar two symbols are. An established measure for this purpose is edit distance [9], which is the minimal number of add, delete, and replace operations to transform one string into another. An equivalent, more operational definition of edit distance recursively compares two strings by either dropping one of the two or both first characters of the strings at a penalty of 1 or to drop the two with no penalty if they match.

### 3.1. Edit distance

**Definition 3** *Edit distance*
*Let $A$, $B$ be strings and $a, b$ characters, then*

$$\begin{aligned} e(A, \epsilon) &= e(\epsilon, A) = |A| \\ e(a.A, b.B) &= min\{e(A, b.B) + 1,\ \ e(a.A, B) + 1, \\ &\qquad e(A, B) + 1,\ \ e(A, B) \text{ if } a = b\ \} \end{aligned}$$

**Example 3** $e(address, adresse) = 2$ *and* $e(007, aa7) = 2$.

Although the first example has six letters in common (adress) and the second only one (7), both edit distances amount to 2. Therefore, there is a need to normalise edit distance to judge the penalties for mismatches relative to the size of the strings. Such a normalised edit distance should range between 0 (no matches) to 1 (no mismatches).

**Definition 4** *Let* $A$, $B$ *be strings and at least one of them non-empty, then* $ne(A, B) = \frac{e(A,B)}{max(|A|,|B|)}$ *is the normalised edit distance.*

**Example 4** *With normalisation we obtain* $ne(address, adresse) = \frac{2}{7}$ *and* $ne(007, aa7) = \frac{2}{3}$.

As the name suggests edit distance $e$ and normalised edit distance $ne$ are distance metrics, i.e. they are (i) symmetric, (ii) the distance from $A$ to $B$ is 0 iff $A = B$ and greater otherwise, and (iii) they satisfy the triangle inequality stating that the direct distance between two strings is the shortest.

## 3.2. Edit Distance over Trees and Fuzzy Unification

While normalised edit distance is well suited to compare symbols, we want to deal with predicates and terms, which have a tree structure. Therefore, we have to extend our definition. It is very important that for the purpose of comparison there is no difference between a tree structure of a predicate and of terms. Hence, we do not distinguish between predicate and function symbols, and in the remainder $t$ often denotes both a predicate or a term. Please note also that we include the empty symbol $\epsilon$ as predicate or function symbol and we do not distinguish between $\epsilon(t)$ and $t$ for a term $t$.

To define fuzzy unification, we have to recursively traverse the tree representing the predicates and terms. In definition 5 of edit distance over trees $et$, the first returned parameter is the number of mismatches, the penalty; the second is the accumulated substitution; the third is a factor for normalisation: the sum of the maximal nodes of the pairwise node comparisons in the recursive traversal. But let's consider this recursive definition in detail: Any term perfectly mismatches the empty symbol, which is penalised with the maximum value - the size of the term. Two variables as well as a variable and a term perfectly match, which is captured by a fuzzy factor of 0 and the corresponding substitutions. Note that for the latter an occurs check is performed. Predicate or function symbols do not contain any further structure and therefore their fuzzy unification factor is given by the edit distance $e$. For the purpose of normalisation we use here the maximum length of the two symbols. In the core of the definition, we reduce two predicates or terms $t, t'$ and call the edit distance over tree recursively. To the edit distance of the leading predicate or function symbol we add the minimum distance after dropping the first term(s) and adding the penalty of the dropped term(s). Thus, the definition compares terms from left to right dropping terms of either the $t$, $t'$, or both $t$ and $t'$. The result of this decompositions are added up using $\oplus$, which adds numbers and concatenates substitutions.

**Definition 5** *Fuzzy Unification*
*Let* $t = f(t_1, \ldots, t_n)$ *and* $t' = f'(t'_1, \ldots, t'_m)$ *be two terms or predicates, and let* $x, y \in V$ *be variables. The size of a term or predicate is defined as:* $size(x) = size(\epsilon) = 0$, $size(f) = |f|$, *and* $size(f(t_1, \ldots, t_n)) = |f| + \sum_{i=1,\ldots,n} size(t_i)$.

*The edit distance over trees et maps two terms or predicate to a tuple of the number of mismatches, a unificator, and a normalisation factor*

$$\begin{array}{l}
et(t, \epsilon) = (size(t), [], size(t)) \\
et(x, y) = (0, [x/y], 0) \\
et(x, t) = (0, [x/t], 0) \text{ if } x \text{ not in } t \text{ and } t \notin V \\
et(f, f') = (e(f, f'), [], max\{|f|, |f'|\}) \\
et(t, t') = et(f, f') \oplus min_v\{ \\
\quad et((t_2, \ldots, t_n), (t'_1, \ldots, t'_m)) \oplus et(t_1, \epsilon), \\
\quad et((t_1, \ldots, t_n), (t'_2, \ldots, t'_m)) \oplus et(t'_1, \epsilon), \\
\quad et((t_2, \ldots, t_n), (t'_2, \ldots, t'_m)) \oplus et(t_1, t'_1)\}
\end{array}$$

*where* $(v, s, n) \oplus (v', s', n') = (v + v', s\, s', n + n')$ *and* $min_v$ *returns the triple with minimal first component.*

*et is called edit distance over trees. The normalised edit distance over trees* $net(t, t') = (\frac{v}{n}, s)$ *with* $(v, s, n) = et(t, t')$ *is called fuzzy unification. For convenience, we often use net to refer only to its first component.*

**Example 5** *Consider example 1, where unification failed because of mismatching predicate and function symbols or missing parameters. With fuzzy unification, we obtain* $net(address(Northampton), address(9b, Northampton)$ $= \frac{2}{7+2+11} = \frac{1}{10}$ *as the argument 9b cannot be matched,* $net(address(Northampton), adresse(Northampton)$ $= \frac{2}{7+11} = \frac{1}{9}$ *as the predicate names mismatch;* $net(address(Northampton), address(Northhampton) = \frac{1}{7+12}$ *as the terms slightly mismatch.*

Fuzzy unification lifts the normalisation by maximum size of the compared strings as introduced for the simple edit distance to the level of terms and predicates with a tree representation. An alternative to adding all mismatches and then normalising by the pairwise maximum length of the compared nodes is a direct normalisation of compared nodes using $ne$ and then redefining $\oplus$ to take the average. This has however the disadvantage of favouring short mismatches of parameters (see e.g. example 3, 4, which our above definition does not suffer from.

With the definition of $net$ in place we can prove some of its properties.

**Lemma 1** *Fuzzy unification is a conservative extension of unification, i.e. if $s$ is an MGU for literals $L, L'$, then $(0, s)$ is a fuzzy unificator for $L, L'$.*

**Proof sketch:** The proof is by induction on the predicate/term structure. Let's consider $et$ directly. The case $et(t, \epsilon)$ cannot occur, as they would not unify. For variables we have $et(x, y) = et(x, t) = (0, s, 0)$, where $s$ is the MGU. For the general case of $t, t'$, we know by the induction hypotheses that all subterms unify with factor 0 and according MGU. □

**Theorem 2** *Let $t, t'$ be terms or predicates. $net$ is normalised, i.e. $0 \leq net(t, t') \leq 1$.*

**Proof sketch:** To see that $net$ is normalised, consider that $net(t, t') \geq 0$ as all factors involved are positive and that $net(t, t') \leq 1$ as $net(t, t') = \frac{v}{n} = \frac{v_1+v_2+\ldots}{n_1+n_2+\ldots}$ with $(v, s, n) = et(t, t')$ and $v = v_1 + v_2 + \ldots$ and $n = n_1 + n_2 + \ldots$ and $v_i \leq n_i$ by definition of $et$. □

For the purpose of relating our edit distance over trees to classical edit distance, let us not distinguish between terms and their canonical string representation. Then edit distance over trees $net$ is "cruder" than $ne$, as the latter can compare character by character, whereas the former has to drop, add, or replace whole terms.

**Lemma 2** *Given two terms or predicates $t, t'$ without variables and let us not distinguish $t, t'$ from their canonical string representation, then $net(t, t') \geq ne(t, t')$.*

Before we formally prove this lemma let us introduce a helpful lemma:

**Lemma 3** *Let $A = CD$ and $B = EF$ be strings. Then $e(A, B) \leq e(C, E) + e(D, F)$*

**Proof sketch of lemma 3:** By definition of edit distance $e(C, E)$ and $e(D, F)$ are the minimal number of add-, delete-, replace-operations required to transform $C$ to $E$ and $D$ to $F$, respectively. Therefore $e(A, B) = e(CD, EF) \leq e(C, E) + e(D, F)$. □

**Proof of Lemma 2:** To prove $net(t, t') \geq ne(t, t')$ it is sufficient to consider the simplified case $t = p(c)$ and $t' = p'(c')$ for predicate symbols $p, p'$ and constants $c, c'$. If $t = t'$, then $net(t, t') = 0 = ne(t, t')$ by definition. If $t \neq t'$, we assume without loss of generality that $size(t) > size(t')$. If $|p| \geq |p'|, |c| \geq |c'|$, then $net(t, t') = \frac{e(p,p')+e(c,c')}{|p|+|c|} \geq \frac{e(p(c),p'(c'))}{|p(c)|} = ne(t, t')$ according to lemma 3. In the final case ((that $|p| \geq |p'|, |c| \geq |c'|$ does not hold), we assume without loss of generality that $|p| \leq |p'|, |c| \geq |c'|$. Now let us assume that $net(t, t') < ne(t, t')$ and thus $net(t, t') = \frac{e(p,p')+e(c,c')}{|p'|+|c|} = \frac{e(p,p')+e(c,c')}{|p(c)|-(|p'|-|p|)} < \frac{e(p(c),p'(c'))}{|p(c)|} = ne(t, t')$ by definition. This is however contradictory, as $|p(c)| > |p(c)| - (|p'| - |p|)$ by the choice of $p, p', c, c'$ and $e(p, p') + e(c, c') \geq e(p(c), p'(c'))$ by lemma 3. All in all, we have established that $net(t, t') \geq ne(t, t')$. □

### 3.3. Fuzzy SLD Resolution

Having defined fuzzy unification, we can turn to fuzzy SLD resolution. The extension is straight forward in that we only replace unification by fuzzy unification. The tricky part is the definition of how to add fuzzy unificators. I.e. given two fuzzy unification factors, we have to define a function $\otimes$ combining both and returning the desired value for the resolvent. In a fuzzy setting, the natural choice for $\otimes$ is the maximum, but as we will see below there are other options. For the general setting, we just require that $\otimes$ operates on the interval of $[0, 1]$ and has a neutral element 0, i.e. $0 \otimes x = x \otimes 0 = x$.

**Definition 6** *Fuzzy SLD Resolution*
*Let $N$ be a negative clause and $\neg A \in N$. Let $D$ be definite clause and $A' \in D$. Let $(v, sub) = net(A, A')$ be the fuzzy unification of $A$ and $A'$. Then $N' = ((N - \{\neg A\}) \cup (D - \{A'\}))sub$ is called fuzzy resolvent of $N$ and $D$ with fuzzy unificator $(v, sub)$.*

*Let $\otimes : [0, 1] \times [0, 1] \mapsto [0, 1]$ be a function with $x \otimes 0 = x \otimes 0 = x$. Let $N$ be a negative clause and $\mathcal{D}$ be a set of definite clauses. An fuzzy SLD-derivation of $\mathcal{D} \cup \{N\}$ consists of sequence $N = N_0, N_1, \ldots$ of negative clauses, a sequence $D_1, D_2, \ldots$ of clauses in $\mathcal{D}$, a sequence of fuzzy unificators $(v_1, sub_1), (v_2, sub_2), \ldots$, and a sequence of fuzzy factors $0 = v'_0, v'_1, \ldots$ such that $N_{i+1}$ is a fuzzy resolvent of $N_i$ and $D_{i+1}$ with fuzzy unificator $(v_{i+1}, sub_{i+1})$ and $v'_{i+1} = v'_i \otimes v_{i+1}$.*

*A fuzzy SLD-refutation is a fuzzy SLD-derivation, where the last negative clause is empty.*

**Lemma 4** *Fuzzy resolution is a conservative extension of resolution, i.e. if there is an SLD refutation, then there is a fuzzy SLD refutation with fuzzy unificators $(0, v_i)$.*

**Proof sketch:** The lemma holds since fuzzy unification conservatively extends unification and since the initial fuzzy factors $v'_0 = 0$ and clauses to be resolved perfectly match and combining these factors always maintains 0 by definition of $\otimes$, i.e. $0 \otimes 0 = 0$. □

### 3.4. Accumulation Functions

There is a fundamental difference between the operators $\oplus$ for unification and $\otimes$ for resolution. During unification we add up mismatches of symbols supposed to be similar; during resolution we add up mismatches of different predicates, which are not related to each other apart from occurring in the same clause. The operation $\otimes$ caters for the computation of the fuzzy values during inference. Hence, one natural interpretation of $\otimes$ would be the maximum value (note, a fuzzy "and" is usually associated with the minimum function, but in our case 0 represents no mismatch and 1 complete mismatch, so that we use maximum). No matter how $\otimes$ is instantiated, it is important that it respects the neutral element 0, so that $x \otimes 0 = 0 \otimes x = x$. This is satisfied by the max function. The intuitive idea of the max-function for $\otimes$ is that max propagates the worst match from a body of a rule to the head. If for example nine out of ten predicates in a rule perfectly match, but one does not match at all, the overall result would be a complete mismatch. Although this is natural bearing classical inference in mind, some applications may require a "softer" function such as multiplication: $x \otimes y = 1 - (1-x)(1-y) = x + y - xy$. Again, $x \otimes 0 = 0 \otimes x = x$ is satisfied, but in contrast to max, the inputs are accumulated, so that all contribute to the final result. Two other alternatives worth mentioning are the average, which is interesting for the same reasons as $x+y-xy$, but which does not treat 0 as the neutral element and second a hierarchical approach, which emphasises the mismatches on higher levels of the tree structure more than on lower levels. This corresponds to the intuition that mismatches in the predicate symbol are more important than mismatches in some nested argument to a predicate.

# 4. META-INTERPRETER

We have implemented a meta-interpreter for FURY in Sicstus Prolog based on a standard vanilla interpreter, as found in [7], replacing Kowalski's "match" by fuzzy unification. In addition in the interpreter below, $\otimes$ is implemented as multiplication and the result is stored in the variable `AccVal`. As theorem 2 states, any terms $t, t'$ match at least with $net(t,t') \leq 1$, where $net(t,t') = 1$ for complete mismatches. In order to prune the search space to a reasonable size, we introduce a cut off value (`Cutoff`), which terminates the exploration of the current branch, if `AccVal` is less than the cut-off. In the code below, `P` is the programs, `G, Gs` goals, `Subs` substitutions, `R` rules

```
solve(P,Gs,Cutoff,AccVal)<-empty(Gs).
solve(P,Gs,Cutoff,AccVal))<-AccVal<Cutoff.
solve(P,Gs,Cutoff,AccVal))<-
  AccVal >= Cutoff, select(Gs,G,Rest),
  member(R,P), renamevars(R,Gs,R'),
  parts(R',Head,Body), net(Head,G,Net,Subs),
  AccVal'=Net*AccVal, add(Body,Rest,NewGs),
  apply(NewGs,Subs,NewGs'),
  solve(P,NewGs',Cutoff,AccVal).
```

| $net < n$ | % |
|---|---|
| 0.1 | 0.009 |
| 0.2 | 0.015 |
| 0.3 | 0.031 |
| 0.4 | 0.081 |
| 0.5 | 0.24 |
| 0.6 | 1.0 |
| 0.7 | 5.0 |
| 0.8 | 26 |
| 0.9 | 79 |
| 1.0 | 96 |

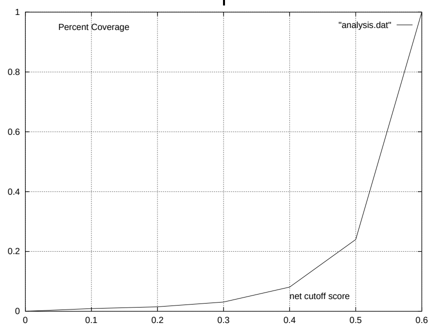


**Figure 1. Coverage vs net cutoff $n$ and distribution of cumulative scores for pairwise comparison of entries in Brenda database.**

Edit distance [9] (see Def 3.1) for two strings length $n$ and $m$ when implemented naively has complexity order exponential in $n$ and $m$, hence the complexity of edit distance over trees is even worse. However, using a dynamic programming approach [15, 6] the complexity of edit distance can be reduced to $O(mn)$. Based on dynamic programming, our algorithm computing edit distance over trees has $O(n^{2d})$ where $n$ is the maximum number of children and $d$ is the minimum depth of the two trees.

## 4.1. Analysis of the BRENDA enzyme function database

Biologists make heavy use of database in flat-file, which are often unclean as they are compiled by hand. However, the correct spelling of entries is crucial, when performing *automated* analysis of the data. As a first step to clean data, we performed an analysis of the BRENDA enzyme function database [13]. There are 15915 entries (enzyme names), with an average string length of 26. We performed an all against all pairwise comparison of the entries; the average time for performing a normalised edit tree comparison between any two enzyme names is 3.9ms, i.e. on average a minute to compare one enzyme name against all the others in Brenda.

As an example, we performed a normalised edit tree comparison between the string 'H2O' and all 15915 entries.

| $net$ | A | B | C | D |
|---|---|---|---|---|
| $[0.0 - 0.1)$ | 1 | 1 | 1 | 1 |
| $[0.1 - 0.2)$ | 0 | 0 | 0 | 0 |
| $[0.2 - 0.3)$ | 2 | 4 | 4 | 4 |
| $[0.3 - 0.4)$ | 0 | 0 | 0 | 4 |
| $[0.4 - 0.5)$ | 2 | 4 | 4 | 4 |
| $[0.5 - 0.6)$ | 4 | 4 | 4 | 7 |
| $[0.6 - 0.7)$ | 9 | 10 | 10 | 39 |
| $[0.7 - 0.8)$ | 5 | 7 | 7 | 25 |
| $[0.8 - 0.9)$ | 13 | 20 | 21 | 97 |
| $[0.9 - 1.0)$ | 17 | 22 | 676 | 3346 |
| 1.0 | 0 | 0 | 0 | 12388 |

**Figure 2. Analysis of Brenda database listed by $net$-intervals. Distribution of scores for all 15915 for exact matching of 'H2O' (A), matching of the regular expression 'H.*2.*O' (case sensitive (B) and insensitive (C)), and $net$-matching with 'H20' (D).**

| $net$ | String | $net$ | String | $net$ | String |
|---|---|---|---|---|---|
| 0.0 | 'H2O' | 0.40 | 'H2CO3' | 0.60 | 'H2N-R' |
| 0.25 | **'H 2O'** | 0.40 | 'H2SO4' | 0.60 | 'HCO3-' |
| 0.25 | **'H2 O'** | 0.40 | 'NH2OH' | 0.60 | 'HCOO-' |
| 0.25 | 'H2O2' | 0.50 | 'H2Se' | 0.60 | 'NH4OH' |
| 0.25 | **'H2Oo'** | 0.50 | 'HNO2' | 0.60 | 'RHg2+' |
| 0.33 | 'H2' | 0.50 | 'Hg2+' | 0.63 | 'ADP+ H2O' |
| 0.33 | **'H20'** | 0.50 | 'RCH2OH' | 0.63 | 'AMP+ H2O' |
| 0.33 | 'H2S' | 0.57 | 'C3(H2O)' | 0.63 | 'CO2 +H2O' |
| 0.33 | 'N2O' | 0.57 | 'H+ H2O2' | 0.63 | 'GDP+ H2O' |
| 0.40 | **'+ H2O'** | 0.57 | **'H2O (r)'** | 0.63 | **'H2O <1>;'** |
| | | 0.60 | 'CH3OH' | 0.63 | 'IDP+ H2O' |

**Figure 3. Absolute scores of closest fuzzy matches to 'H20' in the Brenda database. Correct matches are bold.**

Most typographical errors were found with $net < 0.4$ (8 mismatches out of 15915 entries, see D in Fig. 2). In the general case, $1\%$ comparisons result in a score less than 0.6 and less than $0.1\%$ result in a score of less than 0.4 (see Fig. 1). Thus, extrapolating from our results for the 'H2O' comparison with a cutoff $net < 0.6$ then on average there will be only 160 potential errors to check out of the database of 15915 entries, which is a realistic task.

The distribution of scores in shown in the last column (D) of Fig. 2. The other columns show the number of hits for matching of 'H20' (A) and 'H*2*O' (B (case sensitive), C (case insensitive)) listed by intervals of $net$. Fig. 3 shows a detailed list of the top hits of $net$ with score less than 0.625. It indicates that $net$ filters entries well and illustrates the kind of false positives detected by our method.

We have also adapted $net$ to be used to compute the distance between the topological representation of two aligned protein structures [5].

## 5. COMPARISON AND CONCLUSION

Recently, there has been much interest in fuzzy logic programming. Our work is closely related to Arcelli, Formato, Gerla [3], who develop an abstract framework for fuzzy unification and resolution. There are important differences: First, [3] does not allow unification of predicates of different arity, which is however a problem often occurring in Prolog programming [4]. Second, [3] is not an extension of classical unification, which is important for compatibility reasons. Third, our work is based on a specific similarity measure, namely edit distance. Much work on edit distance is quiet mature [9, 15, 1], but still widely used, especially in bioinformatics. For our interpreter we needed a normalised edit distance over trees. Although there has been some work on normalised edit distance [14], this work does not deal with tree structures and the normalisation is defined different from ours. Basically, the authors assign a weight for each edit operation and minimise the sum of all required edit operations and then divide by the length of the path. In our context this is not applicable, as we normalise by the maximum string length. Some other interesting work on edit distance introduces swapping of characters as additional edit operation besides replace, add, and delete [11]. This idea could be integrated into our fuzzy unification definition, and would make sense as [4] points out that this is a common mistake in logic programming. Besides our purely syntactical string comparisons, it may be desirable to consider semantical similarity. To this end, [8] introduces semantic comparisons, which our current framework and system do not deal with. One reason, why this is difficult to achieve, is the definition of semantical equality, which may be based on thesaurus entries, but which is quite vulnerable.

To summarise, the main contributions of this paper are three-fold: First, we define and proof important properties of normalised edit distance over trees, which is general in nature and applicable to other problems; Second, we define fuzzy unification and resolution in terms of this normalised edit distance over trees and show that it is a conservative extension of classical unification and resolution; Third, we efficiently implement our system using dynamic programming and evaluate its performance and behaviour on the Brenda database [13]. To the best of our knowledge there is no other similar framework and system The system is available over the web at `www.soi.city.ac.uk/~drg/systems/fury`

## Acknowledgements

We would like to thank Lorenz Wernisch for the complexity results.

## References

[1] A. V. Aho and T. G. Peterson. A minimum distance error-correcting parser for context-free languages. *SIAM Journal on Computing*, 1(4):305–312, Dec. 1972.

[2] S. Embury, A. Jones, I. Sutherland, W. Gray, R. White, J. Robinson, F. Bisby, and S. Brandt. Conflict detection for integration of taxonomic data sources. In *Proceedings of the 11th International Conference on Scientific and Statistical Database Management*, pages 204–213, 1999.

[3] F. A. Fontana, F. formato, and G. Gerla. Fuzzy unification as a foundation of fuzzy logic programming. In *Logic Programming and Soft Computing*, pages 51–68. Research Studies Press, 1998.

[4] P. Fung, M. Brayshaw, B. du Boulay, and M. Elsom-Cook. Towards a taxonomy of misconceptions of the prolog interpreter. In P. Brna, B. du Boulay, and H. Pain, editors, *Learning to Build and Comprehend Complex Information Structures: Prolog as a Case Study*. Ablex, 1999.

[5] D. Gilbert, D. Westhead, J. Viksna, and J. Thornton. Topology–based protein structure comparison using a pattern discovery technique. In *submitted to AISB2000 (Artificial Intelligence in Bioinformatics)*, 2000.

[6] D. Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology.* CUP, 1997.

[7] R. A. Kowalski. *Logic for Problem Solving*. Elsevier, 1979.

[8] P. Krisko, P. Marcincak, P. Mihok, and J. Sabol. Low retrieval remote querying dialogue with fuzzy conceptual, syntactical and linguistical unification. *Lecture Notes in Computer Science*, 1495:215–??, 1998.

[9] V. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Akademii nauk SSSR (in Russian)*, 163(4):845–848, 1965. Also in Cybernetics and Control Theory, vol 10, no. 8, pp 707–710, 1996.

[10] J. Lloyd. *Foundations of Logic Programming*. Springer–Verlag, second edition, 1987.

[11] R. Lowrance and R. A. Wagner. An extension of the string-to-string correction problem. *Journal of the ACM*, 22(2):177–183, Apr. 1975.

[12] J. A. Robinson. A machine oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–42, 1965.

[13] D. Schomburg, D. Salzmann, and D. Stephan. *Enzyme Handbook, Classes 1–6*. Springer, 1995. http://www.brenda.uni-koeln.de.

[14] E. Vidal, A. Marzal, and P. Aibar. Fast computation of normalized edit distances. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(9):899–902, Sept. 1995.

[15] R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, Jan. 1974.