European Journal of Operational Research 198 (2009) 706–714

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

An integer linear programming approach for approximate string comparison

Marcus Ritt [a], Alysson M. Costa [b,*], Sergio Mergen [a], Viviane M. Orengo [a]

[a] Instituto de Informática, Universidade Federal do Rio Grande do Sul, UFRGS, Brazil
[b] Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, USP, 13560-970 Sao Carlos, Sao Paulo, Brazil

ARTICLE INFO

Article history:
Received 26 April 2008
Accepted 11 October 2008
Available online 26 October 2008

Keywords:
Approximate string matching
Distance metric
Block edits
Block moves
Integer programming
Hop constraints

ABSTRACT

We introduce a problem called *maximum common characters in blocks* (MCCB), which arises in applications of approximate string comparison, particularly in the unification of possibly erroneous textual data coming from different sources. We show that this problem is NP-complete, but can nevertheless be solved satisfactorily using integer linear programming for instances of practical interest. Two integer linear formulations are proposed and compared in terms of their linear relaxations. We also compare the results of the approximate matching with other known measures such as the Levenshtein (edit) distance.
© 2008 Elsevier B.V. All rights reserved.

1. Introduction

There has been continuous interest in string comparison algorithms due to their wide applicability in database applications, data mining, text processing and bioinformatics, to give some examples. Due to errors in the data, we are interested in approximate comparisons, which determine a usually symmetric and positive distance (or equivalently a similarity) between two given strings. The underlying model of differences for measuring this distance often depends on the application domain and determines its applicability. The edit distance (Levenshtein, 1966), for example, defines the distance of two strings as the minimal number of insertions, deletions or substitutions of a single character and therefore is likely to be an adequate measure, when the error source produces this kind of modification.

Our application is motivated by the task of unifying textual data coming from different data sources. We are interested in an approximate comparison which tolerates a small number of errors of the Levenshtein type (Levenshtein, 1966), but additionally allows the inversion of complete substrings (polygrams) above a given minimum length. This kind of inversion frequently occurs in practice, for example in person names or names of attributes in database schemes, as shown by the two examples in Table 1. This idea leads to the following definition of the similarity between two strings s and t: Identify all occurrences of common substrings longer than a minimum length $k > 0$, which do not overlap, neither in s nor in t. Let L be the total number of characters occurring in these common substrings and define the similarity of s and t as

$$\sigma(s,t) = \frac{L}{\alpha|s| + (1-\alpha)|t|},$$

where $|\cdot|$ denotes the length of a string. This similarity measure is called Carla and has been introduced in (Mergen and Heuser, 2005). The parameter α permits different applications of the similarity measure. A parameter of $\alpha = 0.5$ can be used to compare two strings, while $\alpha = 1$ is more appropriate for deciding if s is contained in t.

The problem of determining the maximal number of common characters L leads to an optimization problem that we call *maximum common characters in blocks* (MCCB):

* Corresponding author. Tel.: +55 16 33738164; fax: +55 16 33739 175.
E-mail addresses: marcus.ritt@inf.ufrgs.br (M. Ritt), alysson@icmc.usp.br, alysson@gmail.com (A.M. Costa), mergen@inf.ufrgs.br (S. Mergen), viviane@inf.ufrgs.br (V. Orengo).

0377-2217/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2008.10.013

Table 1
Typical inversions in textual data and their similarities in the proposed measure to the reference string in the first line for a minimum common substring size $k = 4$ and $\alpha = 0.5$.

Example 1	Similarity	Example 2	Similarity
authorname	1.00	European journal of operational research	1.00
nameauthor	1.00	Operational research, European journal of	0.96
authorsname	0.95	European operations research journal	0.92
author's name	0.87	Operational research journal	0.82
name of author	0.83		

Table 2
Example of instances for MCCB and their corresponding optimal solutions for different minimal substring sizes.

String a	String b	k	Common substrings	L
authorname	nameauthor	4	{*author, name*}	10
semrua	ruacharrua	3	{*rua*}	3
myfasterfoxtrains	myfoxtofastrains	1	{*a,a,f,f,i,m,n,o,r,s,s,t,t,x,y*}	15
myfasterfoxtrains	myfoxtofastrains	3	{*myf, ast, oxt, rains*}	14
myfasterfoxtrains	myfoxtofastrains	4	{*fast, foxt, rains*}	13
myfasterfoxtrains	myfoxtofastrains	5	{*trains*}	6

Instance Two strings $s, t \in \Sigma^*$ over an alphabet Σ, a minimum substring size $k > 0$.
Solution A sequence of substrings $u_1, \ldots, u_n$, each of length at least k such that for $1 \leqslant i \leqslant n$ each u_i occurs in s and in t and u_i does not overlap u_j (in s or t), if $i \neq j$.
Objective Maximize the number of common characters $L = \sum_{1 \leqslant i \leqslant n} |u_i|$.

Table 2 gives some example instances of MCCB.

2. Related work

There is an abundant list of proposals for approximate string comparison. The survey of Navarro (2001) provides a good overview over distances based on models of substring substitution with varying costs in the context of approximate string matching (which applies to approximate string comparison as well). These include the edit distance, the hamming distance, episode distance, and the longest common subsequence. All these measures permit at most one transposition of individual characters.

An early paper about edit distances allowing block edits of Tichy (1984) seeks to minimize the number of common substrings covering one of the input strings, but does not require the common substrings to be non-overlapping, which permits the problem to be solved in polynomial time.

Buss and Yianilos (1995) compare strings based on bipartite matchings. They allow arbitrary matchings between the characters of the two strings and attribute to each matching a cost proportional to the distance of the characters' indices.

Lopresti and Tomkins (1997) study a complete family of models for block edits. Besides matching any number of blocks in both strings, they allow for arbitrary cost functions when comparing the individual blocks. They show the associated problems to be NP-complete for quite general conditions, including the case where the matching blocks are required to be non-overlapping.

The problem studied in this paper is an easy subproblem of the block edit model of Lopresti and Tomkins when the minimal common substring size is $k = 1$. For a general k, we prove in Section 3 that this problem remains NP-complete. Our method allows polygram permutations without additional cost, but differently from all other approaches, it restricts them to polygrams of a minimum length.

The remainder of this paper is organized as follows. Section 4 gives two different formulations for the problem as integer linear programs (ILP) to find the optimal value of MCCB, which is used by Carla. In both cases, the lower bound on the common substring length is modeled with the use of the so-called hop constraints (Gouveia, 1999, Costa et al., forthcoming). In Section 6, we study the relative performance of the ILP formulations. We evaluate the quality of Carla in comparison to other approximate string matching functions in Section 7.

3. NP-completeness of MCCB

In this section, we consider the decision version of MCCB, where an instance has an additional parameter L and we want to decide if there exists a set of common substrings of total size at least L. The following theorem shows, that this problem is NP-complete.

Theorem 1. *MCCB is* NP*-complete.*

Proof. To establish NP-completeness, we have to show that MCCB is in NP and is NP-hard. We first show that MCCB is in NP. Given an instance with two strings s, t, and a minimum common substring size k, let $M = \max(|s|, |t|)$. Since the common substrings are non-overlapping in both s and t, there are at most $N = \lfloor \min(|s|, |t|)/k \rfloor \leqslant M$ of them, each of which can be described by a triple (i,j,l) where i is the start of the substring in s, j is the start of the substring in t and l is the length of the substring. Each number has length $O(\log M)$. We can guess a collection of at most N triples (there are at most M^{3M} such collections) and verify in polynomial time that (i) all the intervals are non-overlapping, (ii) the selected substrings in s and t are the same and of length at least k and (iii) that $\sum_{1 \leqslant i \leqslant N} l_i \geqslant L$.

To prove NP-hardness, we will reduce 3-SAT to MCCB. Let

$$\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m,$$

be an instance of 3-SAT with variables $X = \{x_1, \ldots, x_n\}$ and m clauses $C_j = (c_{j1} \vee c_{j2} \vee c_{j3})$, for literals c_{jk}. In the following, we will assume that no clause contains a variable and its negation. Such clauses are trivially true and could be filtered out in an additional step.

We will construct two strings s and t, such that for $k = 3$ we have that $C \geqslant 3n(2m+1) + 3m$ iff φ is satisfiable. The underlying alphabet Σ of size $5m + 9mn + 1$ is the union of

- an alphabet $\Sigma_j = \{\sigma_{j1}, \ldots, \sigma_{j5}\}$ for each clause $j \in [1,m]$;
- an alphabet $V_{ij} = \{v_{ij1}, \ldots, v_{ij5}\}$ for each variable $i \in [1,n]$ and clause $j \in [1,m]$;
- an alphabet $W_{ij} = \{w_{ij0}, w_{ij1}, w_{ij2}, w_{ij3}\}$ for each variable $i \in [1,n]$ and clause $j \in [1,m]$; and
- an additional symbol τ.

The different alphabets serve to avoid unwanted common strings in the following construction.

First, for each variable i and clause j we construct two strings s_{ij} and t_{ij} as follows:

$$s_{ij} = \begin{cases} v_{ij1}v_{ij2}\sigma_{jp}\sigma_{jp}\sigma_{j,p+1}\sigma_{j,p+2} & \text{if } x_i = c_{jp} \text{ for a } p \in \{1,2,3\}, \\ \sigma_{j,p}\sigma_{j,p+1}\sigma_{j,p+2}\sigma_{j,p+2}v_{ij4}v_{ij5} & \text{if } \overline{x_i} = c_{jp} \text{ for a } p \in \{1,2,3\}, \\ v_{ij1}v_{ij2}v_{ij3}v_{ij3}v_{ij4}v_{ij5} & x_i \notin C_j \wedge \overline{x_i} \notin C_j, \end{cases} \tag{1}$$

$$t_{ij} = \begin{cases} v_{ij1}v_{ij2}\sigma_{jp}\sigma_{j,p+1}\sigma_{j,p+2} & \text{if } x_i = c_{jp} \text{ for a } p \in \{1,2,3\}, \\ \sigma_{j,p}\sigma_{j,p+1}\sigma_{j,p+2}v_{ij4}v_{ij5} & \text{if } \overline{x_i} = c_{jp} \text{ for a } p \in \{1,2,3\}, \\ v_{ij1}v_{ij2}v_{ij3}v_{ij4}v_{ij5} & x_i \notin C_j \wedge \overline{x_i} \notin C_j. \end{cases} \tag{2}$$

If a clause contains multiple occurrences of a variable x_i, we choose the minimum possible p in the cases above. Each pair s_{ij}, t_{ij} is constructed so that s_{ij} equals t_{ij} with the middle character repeated. Therefore either the first or the last half of s_{ij} can overlap with t_{ij} and each pair contributes at most one common substring of length three (see Fig. 1a) and represents the selection of either $x_i = t$ or $x_i = f$, for a (potential) use in clause j.

Next, for each variable i, we construct a string s_i of length $12m + 3$ by concatenating all s_{ij}, and a string t_i of length $8m + 4$ by concatenating all t_{ij}

$$s_i = g_{i0}s_{i1}g_{i1}s_{i2}g_{i2}\cdots s_{i,m-1}g_{i,m-1}s_{im}g_{im},$$
$$t_i = h_{i0}t_{i1}h_{i1}t_{i2}h_{i2}\cdots t_{i,m-1}h_{i,m-1}t_{im}h_{im},$$

where

$$\begin{aligned} g_{i0} &= w_{im0}w_{im1}w_{im2}w_{im2}w_{im3}s_{i1}[1] \\ g_{ij} &= s_{ij}[6]w_{ij1}w_{ij2}w_{ij2}w_{ij3}s_{i,j+1}[1] \quad j \in [1,m), \\ g_{im} &= s_{im}[6]w_{im0}w_{im1} \\ h_{i0} &= w_{im0}w_{im1}w_{im2}w_{im3} \\ h_{ij} &= w_{ij1}w_{ij2}w_{ij3} \qquad j \in [1,m), \\ h_{im} &= w_{im0}w_{im1}w_{im2} \end{aligned}$$

($s[i]$ denotes the ith character of string s, counting from one.)

The pair of strings s_i and t_i represents the variable i. All adjacent substrings of length three in s_i are overlapping with their left and right neighbor in t_i. The strings g_{i0} and g_{im} overlap the last and the first substring of s_i in t_i, in a way that there are exactly two ways to choose a maximal subset of $2m + 1$ common strings of length three. Either we can select all $2m + 1$ odd substrings in s_i, or we can select only the first odd substring (namely $g_{i0}[1]g_{i0}[2]g_{i0}[3]$) and all $2m$ even substrings in s_i. In the latter case, the first odd substring in s_i must correspond to h_{im} in t_i, to avoid overlapping. This guarantees a consistent choice of $x_i = f$ or $x_i = t$ in all clauses. Therefore, the pair s_i, t_i can contribute at most $3(2m + 1)$ characters in common substrings and this number can be achieved by exactly two choices of common substrings.

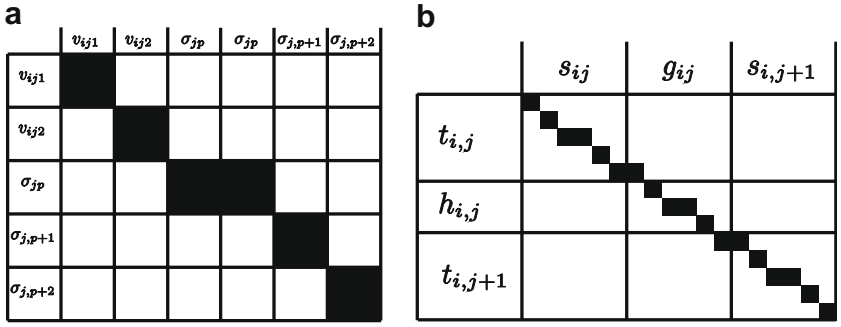


Fig. 1. (a) Left: overlapping of strings s_{ij} and t_{ij} for the case $x_i = c_{jp}$. The case $\overline{x_i} = c_{jp}$ has the same overlapping with different characters. (b) Right: overlapping strings $s_{ij}g_{ij}s_{i,j+1}$ and $t_{ij}h_{ij}t_{i,j+1}$.

Then, consider the concatenations $s = s_1 s_2 \dots s_n$ and $t' = t_1 \tau t_2 \tau \dots \tau t_n$. Since, by choice of the alphabets, for $i \neq j$, the longest common substring of s_i and t_j has length two, the pair s, t' can contribute at most $3n(2m+1)$ characters in any selection of common substrings. Finally, let

$$t = t'\tau\sigma_{11}\dots\sigma_{15}\sigma\sigma_{21}\dots\sigma_{25}\dots\tau\sigma_{m1}\dots\sigma_{m5}.$$

The length of s and t is $n(12m+6)$ and $n(8m+4)+6m+n-1$, respectively. The whole construction can be easily carried out in polynomial time.

Consider some valuation of ϕ. The characters $\sigma_{j1},\dots,\sigma_{j5}$ in the final part of t match the substrings in s_{ij} which correspond to negated literals, in a way, that all three possible substrings of length three overlap in t. Therefore, for each satisfied clause, the final part of t can contribute three characters to C. If φ is satisfiable, it is possible to find a solution of size $C \geqslant 3n(2m+1)+3m$.

Conversely, if there is a collection of common substrings of total length $C \geqslant 3n(2m+1)+3m$, by construction we must have $C = 3n(2m+1)+3m$. In this case, each of n pairs s_i, t_i must contribute its maximum of $3(2m+1)$ characters, and each of m pairs s and $\sigma_{j1}\dots\sigma_{j5}$ exactly three characters. If we attribute $x_i = t$ whenever s_{i1} is covered by the common substrings, and $x_i = f$ otherwise, φ must be satisfied. □

4. Two integer linear programs

In this section, we propose two integer linear formulations for the MCCB. We first introduce a input representation, in order to simplify the formulation presentation. Then, the two models are described.

4.1. Input representation

In order to formulate the MCCB as integer programs, let us first represent the original input as a 0–1 matrix with dimensions $I \times J$, ($I = |s|, J = |t|$), in which the element a_{ij} is equal to zero if no matching occurs between the position i in the first string and position j in the second string, and to one otherwise. An example is presented in Fig. 2a.

We call a sequence of elements $(i_j,(i+1)_(j+1),\dots,(i+q)_(j+q))$ with $q \geqslant 0$ a *maximum block* X_t if $a_{ij} = 1$, $\forall i_j \in X_t$ and elements $a_{(i-1)(j-1)}$ and $a_{(i+q+1)(j+q+1)}$ are either equal to 0 or do not exist (exceed the matrix dimensions). Let us call $X = \{X_1, X_2, \dots, X_t, \dots, X_{|X|}\}$ the set of all maximum blocks. In the matrix given in Fig. 2a, we have $X = \{(1_1,2_2,3_3,4_4),(1_7,2_8),(3_5,4_6,5_7,6_8),(5_1,6_2)\}$. For an element $i_j \in X_t$, we say that elements $(i+q)_(j+q) \in X_t$ are below i_j and elements $(i-q)_(j-q) \in X_t$ are above i_j, $q > 1$.

Based on the input table, we create a matrix called *forward* in which each element f_{ij} associated with $i_j \in X_t$ represents the number of elements $g_h \in X_t$ such that $g \leqslant i$ and $h \leqslant j$ (i.e., the number of elements in X_t that are above i_j plus one). Analogously, we create a *backward* matrix in which each element b_{ij} associated with $i_j \in X_t$ represents the number of elements $g_h \in X_t$ such that $g \geqslant i$ and $h \geqslant j$ (i.e., the number of elements in X_t that are below i_j plus one). The forward and backward matrices corresponding to the input matrix in Fig. 2a are presented in Fig. 2b and c (null entries are omitted for clarity).

Let us now define a digraph $G = (N \cup N_e, A_o \cup A_d \cup A_i)$. The sets N, N_e, A_o, A_d and A_i are defined as follows:

- $N = \{i_j | f_{ij} + b_{ij} \geqslant k+1\}$
- $N_e = \{o_o, d_d\}$, where o_o and d_d are two artificial vertices.
- $A_o = \{(o_o, i_j) | (i_j) \in N, b_{ij} \geqslant k\}$
- $A_d = \{(i_j, d_d) | (i_j) \in N, f_{ij} \geqslant k\}$
- $A_i = \{(i_j, i_j^+) | i_j, i_j^+ \in N, a_{ij} = a_{i_j^+} = 1\}$

In order to simplify the presentation, we use symbols i_j^+ standing for $(i+1)_(j+1)$. Later, we will also use i_j^- to represent vertex $(i-1)_(j-1)$. Moreover, as mentioned before, we use a_{ij}, f_{ij} and b_{ij} to represent an element of the input, forward and backward matrices, respectively. Note that each element in N belongs to a single maximum block. Let $\overline{X}(i_j)$ be a function that returns the index of the maximum block to which an element i_j belongs. For instance, in the example presented in Fig. 2a, $\overline{X}(1_1) = 1$ and $\overline{X}(1_7) = 2$.

We can see that N is the set of all elements i_j belonging to a maximum block with at least k elements. A_o is the set of arcs connecting the artificial origin vertex o_o to all elements i_j, such that the element i_j has at least $k-1$ elements below it. Analogously, A_d is the set of all arcs connecting elements of N with at least $k-1$ elements above it to the artificial destination vertex d_d. Finally, A_i is the set of arcs linking

a

$A = [a_{ij}] =$

	$ABCDCDAB$
A	1 0 0 0 0 0 1 0
B	0 1 0 0 0 0 0 1
C	0 0 1 0 1 0 0 0
D	0 0 0 1 0 1 0 0
A	1 0 0 0 0 0 1 0
B	0 1 0 0 0 0 0 1
G	0 0 0 0 0 0 0 0
H	0 0 0 0 0 0 0 0

b

$F = [f_{ij}] =$

	$ABCDCDAB$
A	1 1
B	2 2
C	3 1
D	4 2
A	1 3
B	2 4
G	
H	

c

$B = [b_{ij}] =$

	$ABCDCDAB$
A	4 2
B	3 1
C	2 4
D	1 3
A	2 2
B	1 1
G	
H	

Fig. 2. (a) Input matrix; (b) forward matrix; (c) backward matrix.

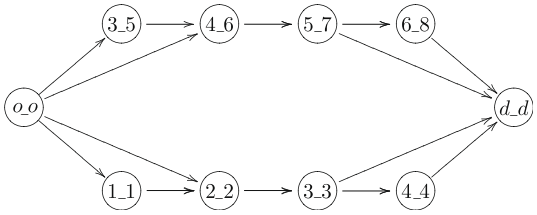


Fig. 3. Graph G corresponding to input matrix in Fig. 2a.

each element $i_j \in N$ to the element that comes immediately after it in the maximum block $\overline{X}(i_j)$, if such an element exists. Graph G for the input matrix in Fig. 2a and $k = 3$ is presented in Fig. 3.

4.2. Formulations

In this subsection, we use the just described representation to model the MCCB problem.

4.2.1. Path formulation

Let $P = (i_1 = o_o, \ldots, i_\ell = d_d)$ denote a path originating at vertex o_o and ending at vertex d_d, containing ℓ vertices. Define $\mathscr{P}_h$ as the set of paths P in G with length $3 \leqslant \ell \leqslant k+1$. Each path $P \in \mathscr{P}_h$ contains at most $k-1$ vertices $\in N$ (the set of vertices that do not include o_o and d_d) and thus is equivalent to an infeasible choice of substring, since it violates the string limit constraint. Note that for $k = 1$, set $\mathscr{P}_h$ is empty.

We now define binary variables y_{i_j} and x_{i_j,g_h}, associated with each vertex $i_j \in N$ and with each arc $(i_j, g_h) \in A_o \cup A_d \cup A_i$, respectively. Variables y_{i_j} are equal to one if, in the solution, there is an arc incident to vertex i_j and equal to zero, otherwise. Variables x_{i_j,g_h} are equal to one if arc (i_j, g_h) is present in the solution and zero, otherwise. With these definitions, we can present a mathematical formulation for the problem:

$$\text{Maximize} \quad \sum_{i_j \in N} y_{i_j} \tag{3}$$

$$\text{subject to} \quad y_{i_j} = \sum_{(g_h, i_j) \in A_o \cup A_i} x_{g_h, i_j}, \quad i_j \in N, \tag{4}$$

$$\sum_{(g_h, i_j) \in A_o \cup A_i} x_{g_h, i_j} = \sum_{(i_j, g_h) \in A_d \cup A_i} x_{i_j, g_h}, \quad i_j \in N, \tag{5}$$

$$\sum_{j | i_j \in N} y_{i_j} \leqslant 1, \quad i = 1, \ldots, I, \tag{6}$$

$$\sum_{i | i_j \in N} y_{i_j} \leqslant 1, \quad j = 1, \ldots, J, \tag{7}$$

$$\sum_{t=2}^{\ell} x_{p_{t-1}, p_t} \leqslant \ell - 2, \quad P = (p_1 = o_o, \ldots, p_\ell = d_d) \in \mathscr{P}_h, \tag{8}$$

$$y_{i_j} \in \{0, 1\}, \quad i_j \in N, \tag{9}$$

$$x_{i_j, g_h} \in \{0, 1\}, \quad (i_j, g_h) \in A_o \cup A_d \cup A_i. \tag{10}$$

The objective function maximizes the number of matches found. Variables y_{i_j} are descriptive and count the number of incoming links of a vertex i_j, as shown in constraints (4). Eq. (5) are flow conservation constraints. Constraints (6) and (7) are responsible for avoiding overlaps: constraints (6) guarantee that a single match is chosen in each row while constraints (7) guarantee that a single match is chosen per column. Finally, constraints (8) assure that no string with less than k characters belonging to N is chosen. These constraints, called hop constraints, are common in the literature dealing with optimizations in graphs (Gouveia, 1999; Costa et al., forthcoming, e.g.), however, to our knowledge, it has only been used to limit the solutions to those containing paths of at most a given number of links. In our case, the hop constraints are used with the inverse objective, i.e, to limit solutions to those containing at least k links. We believe that this new application of the constraints might be useful in other situations. It is also worth noting the unusual formulation of a string comparison problem as a network design model.

4.2.2. Time-dependent variables formulation

A second formulation can be obtained by using the so-called time-dependent variables (Gouveia, 1999; Costa et al., forthcoming). Variables y_{i_j}, $i_j \in N$, $x_{o_o, i_j} \in A_o$ and $x_{i_j, d_d} \in A_d$ are defined as above but arc variables for arcs in A_i are defined as follows: let $x^t_{i_j, i_j^+}$ be a binary variable that is equal to one if arc (i_j, i_j^+) is present in the solution *and* i_j is the t^{th} vertex in the path from the origin vertex o_o, and to zero otherwise. We carefully choose the values of t that can be applied to each arc, in order to use the minimum number of variables. This is done with the help of the forward and backward matrices: for each arc (i_j, i_j^+) we create $x^t_{i_j, i_j^+}$ with $t = t^{i_j}_{\min}, \ldots, t^{i_j}_{\max}$. The limit $t^{i_j}_{\min}$ is given by $\max\{1, k - b_{ij} + 1\}$ and $t^{i_j}_{\max}$ is given by f_{ij}.

The rationale behind the choice of $t_{\min}$ and $t_{\max}$ is simply the fact that the possible set of positions of each vertex in a solution path is restricted by the graph topology. Consider, for instance, the graph presented in Fig. 3, it is clear that node 3_3 can only appear in positions 2 or 3 in a path originating at vertex o_o. Indeed, these are the values that one obtains by computing $t^{3_3}_{\min}$ and $t^{3_3}_{\max}$. Note that once we have variables indicating the position of each arc in a path, we can explicitly set flow conservation constraints that only allow paths with at least

k vertices $\in N$. This is done by relating, in the flow constraints, the value of exit variables x_{i_j,d_d} to the value of variables $x^t_{i_j^-,i_j}$ for $t \geqslant k-1$. Since any variable $x^t_{i_j^-,i_j} = 1$ with $t \geqslant k-1$ can lead to a variable $x_{i_j,d_d} = 1$, we are not obliged to count the position of an arc in the path, once we know that it has surpassed the minimum limit k. We can, therefore, further restrict $t^{i_j}_{\max}$ to $\max\{f_{ij},k\}$.

With the variables described above, one can write the problem as follows:

$$\text{Maximize} \quad \sum_{i_j \in N} y_{i_j} \tag{11}$$

subject to (4), (6), (7), (9) and (10) and

$$x_{i_j,i_j^+} = \sum_{t=t^{i_j}_{\min}}^{t^{i_j}_{\max}} x^t_{i_j,i_j^+}, \quad (i_j, i_j^+) \in A_i, \tag{12}$$

$$x_{o_o,i_j} = x^1_{i_j,i_j^+}, \quad i_j \in N | t^{i_j}_{\min} = 1, \tag{13}$$

$$x^{t-1}_{i_j^-,i_j} = x^t_{i_j,i_j^+}, \quad \max\{2, t^{i_j}_{\min}\} \leqslant t \leqslant \min\{k-1, t^{i_j}_{\max}\}, \quad i_j \in N, \tag{14}$$

$$\sum_{t=k-1}^{t^{i_j^-}_{\max}} x^t_{i_j^-,i_j} = x^k_{i_j,i_j^+} + x_{i_j,d_d}, \quad i_j \in N | t^{i_j}_{\max} = k, \tag{15}$$

$$x^t_{i_j,i_j^+} \in \{0,1\}, \quad (i_j, i_j^+) \in A_i, t = t^{i_j}_{\min}, \ldots, t^{i_j}_{\max}. \tag{16}$$

In this new formulation, we still maximize the number of matches found. Variables $x_{i_j,i_j^+} \in A_i$ are descriptive and their values are determined by constraints (12). Note that these variables could be eliminated from the formulation, but they are kept for clarity.[1] Constraints (13)–(15) are flow conservation constraints. These constraints can be understood as follows: constraints (13) indicate that all paths must start at the root node. Constraints (14) ensure that once a first link is chosen in a path, this path must be taken for at least k steps. Constraints (15) allow the path to be terminated at a destination vertex once at least k vertices have been chosen in a path.

5. Theoretical comparison of formulations

When dealing with mixed integer linear models, the strength of the associated linearly relaxed models is an important, if not essential, characteristic. The main reason for this is the fact that many of the available solution methods, notably the branch-and-cut method, rely on these relaxations and the stronger the relaxations, the more efficiently the methods will perform. In this section, we establish theoretical domination relations between the LP relaxations of the proposed formulations. We call PF_{LP} and TDF_{LP} the linear relaxations of the path and time dependent formulations, respectively.

We show that TDF_{LP} dominates PF_{LP}. To the extent of our knowledge, this result is new in the literature. Indeed, no results comparing these two families of formulations could be obtained in (Costa et al., forthcoming), where other families of hop constrained formulations have been compared.

The proof will be done by showing that all constraints of PF_{LP} are already implied by TDF_{LP} but the contrary is not true. First, let us prove the two following propositions:

Proposition 1. *Constraints (5) are redundant for the TD formulation.*

Proof. For each $i_j \in N$, using the descriptive variables defined at constraints (12), constraints (5) follow directly by the sum of constraints (13)–(15) for $i_j \in N$, when defined. □

Proposition 2. *Constraints (8) are redundant for the TD formulation.*

Proof. Take any path $P = \{p_1 = o_o, \ldots, p_\ell = d_d\} \in \mathscr{P}_h$. The sum of the values of the flows associated to the arcs in the path is given by

$$L = x_{o_o,p_2} + x_{p_{\ell-1},d_d} + \sum_{t=3}^{t=\ell-1} x_{p_{t-1},p_t}. \tag{17}$$

Note that L is the left-hand side of constraints (8). Note also that vertex $p_{\ell-1}$ is not the last vertex in its maximum block (otherwise, since $\ell \leqslant k+1$, x_{o_o,p_2} would not exist). Therefore, the flow exiting vertex $p_{\ell-1}$, F, is given by the flow destinated to vertex d_d plus the flow destinated to i_j, the next vertex in the maximum block.

$$F = \sum_{p_{\ell-1},g_h \in A_d \cup A_i} x_{p_{\ell-1},g_h} = x_{p_{\ell-1},d_d} + x_{p_{\ell-1},i_j} = x_{p_{\ell-1},d_d} + x^{\ell-2}_{p_{\ell-1},i_j} + \sum_{t=t^{p_{\ell-1}}_{\min}}^{\ell-3} x^t_{p_{\ell-1},i_j} + \sum_{t=\ell-1}^{t^{p_{\ell-1}}_{\max}} x^t_{p_{\ell-1},i_j}.$$

Using constraints (13) and (14), and, again, the fact that $\ell \leqslant k+1$, we can say that $x^{\ell-2}_{p_{\ell-1},i_j} = x_{o_o,p_2}$, and, therefore

$$F = x_{p_{\ell-1},d_d} + x_{o_o,p_2} + \sum_{t=t^{p_{\ell-1}}_{\min}}^{\ell-3} x^t_{p_{\ell-1},i_j} + \sum_{t=\ell-1}^{t^{p_{\ell-1}}_{\max}} x^t_{p_{\ell-1},i_j} \leqslant 1.$$

[1] In order to eliminate constraints (12), one must introduce constraints $\sum_{t=t^{i_j}_{\min}}^{t^{i_j}_{\max}} x^t_{i_j,i_j^+} \leqslant 1, (i_j, i_j^+) \in A_i$.

Table 3
Results of ILP on random instances in the path and time-dependent formulation.

String size	Block edits	$\|\Sigma\|$	Path formulation				Time dependent formulation			
			Time	Optimality gap [%]			Time	Optimality gap [%]		
			[s]	Minimum	Average	Maximum	[s]	Minimum	Average	Maximum
20	10	2	0.225	0.00	**0.00**	0.00	**0.138**	0.00	**0.00**	0.00
20	10	3	**0.071**	0.00	1.22	5.56	0.121	0.00	**0.67**	3.33
20	10	5	0.065	0.00	**0.00**	0.00	**0.062**	0.00	**0.00**	0.00
20	10	10	**0.052**	0.00	**0.00**	0.00	0.086	0.00	**0.00**	0.00
20	10	26	0.077	0.00	**0.00**	0.00	**0.052**	0.00	**0.00**	0.00
20	20	2	0.108	0.00	**0.24**	2.38	**0.089**	0.00	**0.24**	2.38
20	20	3	**0.060**	0.00	**1.00**	10.0	0.123	0.00	**1.00**	10.00
20	20	5	0.081	0.00	0.56	5.56	**0.056**	0.00	**0.00**	0.00
20	20	10	**0.060**	0.00	**0.00**	0.00	0.089	0.00	**0.00**	0.00
20	20	26	**0.063**	0.00	0.67	6.67	0.106	0.00	**0.00**	0.00
20	30	2	0.145	0.00	**0.71**	7.14	**0.122**	0.00	**0.71**	7.14
20	30	3	**0.056**	0.00	**0.00**	0.00	0.086	0.00	**0.00**	0.00
20	30	5	**0.050**	0.00	**0.00**	0.00	0.058	0.00	**0.00**	0.00
20	30	10	**0.057**	0.00	**0.00**	0.00	0.062	0.00	**0.00**	0.00
20	30	26	**0.057**	0.00	**0.00**	0.00	0.062	0.00	**0.00**	0.00
50	10	2	4.570	0.00	**0.00**	0.00	**1.710**	0.00	**0.00**	0.00
50	10	3	0.817	0.00	1.37	3.75	**0.638**	0.00	**1.24**	3.59
50	10	5	**0.119**	0.00	0.26	2.59	0.161	0.00	**0.17**	1.72
50	10	10	**0.097**	0.00	**0.00**	0.00	0.147	0.00	**0.00**	0.00
50	10	26	0.136	0.00	**0.00**	0.00	**0.129**	0.00	**0.00**	0.00
50	20	2	10.070	0.00	**0.00**	0.00	**1.464**	0.00	**0.00**	0.00
50	20	3	2.284	0.00	1.46	3.79	**0.898**	0.00	**1.06**	3.05
50	20	5	0.177	0.00	**0.38**	1.92	**0.174**	0.00	**0.38**	1.92
50	20	10	0.157	0.00	**0.00**	0.00	**0.125**	0.00	**0.00**	0.00
50	20	26	**0.103**	0.00	**0.00**	0.00	0.156	0.00	**0.00**	0.00
50	30	2	2.378	0.00	**0.00**	0.00	**0.705**	0.00	**0.00**	0.00
50	30	3	**0.783**	0.00	0.63	2.33	0.960	0.00	**0.50**	1.55
50	30	5	**0.112**	0.00	0.56	3.37	0.193	0.00	**0.19**	1.92
50	30	10	**0.155**	0.00	0.81	5.13	0.221	0.00	**0.29**	2.94
50	30	26	**0.100**	0.00	0.42	4.17	0.157	0.00	**0.00**	0.00

The inequality comes from the fact that the sum of the outgoing flows of a vertex is bounded by one. Now, since all flow variables are positive, we can write:

$$x_{p_{\ell-1},d_d} + x_{o_o,p_2} \leqslant 1. \tag{18}$$

Finally, using (17), (18) and the fact that each flow variable is bounded by one, we have:

$$L = x_{o_o,p_2} + x_{p_{\ell-1},d_d} + \sum_{t=3}^{t=\ell-1} x_{p_{t-1},p_t}, \tag{19}$$

$$L \leqslant 1 + \sum_{t=3}^{t=\ell-1} x_{p_{t-1},p_t}, \tag{20}$$

$$L \leqslant 1 + \ell - 3 = \ell - 2. \qquad \square \tag{21}$$

We are now able to prove the main result:

Proposition 3. *Formulation TDF_{LP} dominates PF_{LP}.*

Proof. First, we show that TDF_{LP} is not dominated by PF_{LP}. This is done by inspection of Section 6 computational results. We can see that, by the figures presented in Table 3, there are instances for which the gap given by TDF_{LP} (compared to the optimal integral value) is smaller than that given by PF_{LP}, and, therefore PF_{LP} does not imply TDF_{LP}. To show that TDF_{LP} implies PF_{LP}, one needs to show that all constraints of PF_{LP} are implied by TDF_{LP}. Constraints (4), (6), (7), (9) and (10) are explicitly present in TDF_{LP}. It suffices, therefore, to show that constraints (5) and (8) are dominated by the constraints of TDF_{LP}, which has been done by Propositions 1 and 2. $\square$

We will now present a series of computational tests that confirm the advantages of using TDF as well as the efficacy of the integer linear approach to solve the MCCB.

6. Computational results of the ILP approach

We have tested the performance of the ILP approach and the quality of the LP relaxation on randomly generated instances, based on a block edit model. We first create a random string of a given length, and then apply a number of random block edits (deletions, copies and moves), where each block has a random length, up to an upper limit. The result of these operations is the second string used for comparison.

In our experiments, we used initial string lengths 20,50, block edits with blocks of length up to 10 characters, 10,20 and 30 block edits and an alphabet size of 2,3,5,10 and 26. For every combination of these parameters we generated a group of 10 instances.

The experiments have been conducted on an AMD Athlon 64 Dual Core 3800 with 1 GB of RAM using XpressMP v18.00.08. For each of the 24 groups, Table 3 reports the average time for solving the ILP and the optimality gap of the LP relaxation for the path and the time-dependent formulation, using a minimum substring size of $k = 3$.

The optimality gaps show that both formulations yield tight bounds for the ILP. We can also see that the time-dependent formulation consistently yields stronger bounds than the path formulation and therefore can improve the execution time for the difficult instances of alphabet size two by a factor of two to three. Observe also, that the execution time tends to decrease as the alphabet size increases. This is due to the reduced probability of multiple common substrings for an increasing alphabet, and is desirable, since practical instances in string comparison usually use large alphabets. Altogether the solution times show that the ILP approach is able to solve instances of practical interest in reasonable time.

7. Results of approximate string matching

We carried out a series of data retrieval experiments in order to assess the quality of Carla in comparison to other existing similarity functions. The value of the length of the common substrings C has been obtained by the time-dependent formulation, which has proven to be more efficient. The quality of a similarity function for data matching or integration purposes can be measured in terms of its ability in assigning higher scores to strings that represent the same real world object as the string being queried than to strings that do not represent the same object. This ability is usually measured in terms of average precision (AP), as done in (Bilenko et al., 2003). In order to calculate AP, it is necessary to rank a collection of strings according to their similarity score with a string used as query. The entries in this ranking can be classified as relevant or not relevant depending on whether or not they represent the same object as the query. Based on such a ranking, AP is calculated as follows: For each position p in the ranking that corresponds to a relevant item, the precision is the ratio $r_p = n/p$, where n is the number of relevant results retrieved up to line p. After calculating the ratio r for all m relevant ranks, the AP for a query is defined as the average of all m ratios r. The mean average precision (MAP) for a similarity function is the arithmetic mean of the APs for the individual queries. In order to illustrate the calculation of AP, let us consider a query "European journal of operational research" posed over a database that contains eight strings. In such a database, there are $m = 5$ relevant strings for the query. Table 4 shows the ranking of the database items according to Carla. The last column from the table shows r_p for each position p corresponding to a relevant item. If all relevant items were assigned higher scores than all irrelevant items, then $AP = 1$.

The string collections used in the experiments were automatically generated using the data set generator from FEBRL – freely extensible biomedical record linkage (Christen et al., 2004). This tool introduces misspellings (i.e. character insertions, removals, substitutions or inversions) to simulate mistakes that are committed by people when keying in data. Eight string collections were generated. Each collection has one thousand strings that correspond to one hundred distinct person names. The inversion probability of blocks of characters was varied systematically from 0.1 to 0.8 with increments of 0.1. The aim was to assess the quality of the similarity functions as the number of inversions increases.

The tests were done using Carla and six other similarity functions available from SimMetrics (Chapman, 2006), namely: BlockDistance, Cosine, Jaro, Levenshtein, QGramsDistance, Smith Waterman and Soundex. The parameters used for Carla are $\alpha = 0.5$ and $k = 2$, as empirical results have shown that this an adequate configuration for the dataset being analyzed. Recall from Section 1 that $\alpha = 0.5$ can be used to compare two strings and that $k = 2$ means that we are considering common blocks of at least 2 characters.

Table 4
Example of average precision calculation.

String	Score	Relevance	
European journal of operational research	1.0000	Relevant	$r_1 = 1/1 = 1$
Operational research, European journal	0.9230	Relevant	$r_2 = 2/2 = 1$
Europ Jrnl of Operational Research	0.9189	Relevant	$r_3 = 3/3 = 1$
Operations research letters	0.5970	Not relevant	
Computers & operations research	0.5915	Not relevant	
Eur Jrnl Op Res	0.5454	Relevant	$r_6 = 4/6 = 0.66$
Transportation research part A	0.5428	Not relevant	
EJOR	0.0909	Relevant	$r_8 = 5/8 = 0.62$
$AP = (1 + 1 + 1 + 0.66 + 0.62)/5 = 0.85$			

Table 5
Results for the data retrieval experiments. Each cell contains the value for MAP for each similarity function on each string collection.

Similarity function	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	Average
Carla	**0.89**	**0.89**	**0.90**	**0.91**	**0.90**	**0.91**	**0.93**	**0.94**	**0.90**
Q grams distance	**0.89**	0.85	0.82	0.82	0.81	0.82	0.85	0.88	0.85
Smith Waterman	0.88	0.86	0.82	0.81	0.81	0.83	0.85	0.87	0.84
Jaro	0.80	0.72	0.66	0.60	0.58	0.62	0.62	0.68	0.68
Levenshtein	0.79	0.69	0.62	0.55	0.54	0.56	0.59	0.64	0.65
Soundex	0.64	0.57	0.53	0.46	0.45	0.50	0.53	0.54	0.55
Block distance	0.22	0.27	0.29	0.31	0.30	0.37	0.44	0.45	0.32
Cosine similarity	0.22	0.27	0.29	0.31	0.30	0.37	0.44	0.45	0.32

The experiments were done using the SimEval Tool (Heuser et al., 2007). From each data collection, 5 samples of 50 randomly chosen strings were used as queries against the whole collection. The results of MAP for each sample were averaged to produce the final results shown in Table 5.

The results show that Carla obtains the best MAP scores for all collections. Amongst the other functions, Smith-Waterman and QGrams-Distance achieve the best performance. It is also possible to observe that as the number of inversions increases, Carla's advantage in comparison to the other similarity functions becomes more evident.

8. Conclusions

We have introduced a new problem called *maximum common characters in blocks*, which arises in the context of approximate string matching. We have shown that this problem is NP-complete. Moreover, we have proposed two integer linear formulations, which contain a novel application of the so-called hop constraints, either by introducing explicit path constraints or by the use of additional variables. A theoretical comparison shows that the latter dominates the former.

On the practical side, we have shown that these formulations are able to compute the optimal values of our similarity measure Carla for instances of interest. An experimental comparison of Carla with other similarity measures demonstrates that the quality of the results obtained is significantly better for strings with block inversions.

In a future work, we intend to study the applicability of this distance measure in problems of bioinformatics (f. ex. genome rearrangement).

Acknowledgements

We would like to thank the anonymous referees for their helpful suggestions. This work was partially supported by a CAPES-PRODOC grant from the Brazilian Ministry of Education. Sergio Mergen is funded by CAPES.

References

Bilenko, M., Mooney, R., Cohen, W.W., Ravikumar, P., Fienberg, S., 2003. Adaptive name matching in information integration. IEEE Intelligent Systems 18 (5), 16–23.
Buss, S.R., Yianilos, P.N., 1995. A bipartite matching approach to approximate string comparison and search, Technical Report 95–193, NEC Research Institute.
Chapman, S., 2006. Simmetrics. <http://www.dcs.shef.ac.uk/~sam/simmetrics.html>.
Christen, P., Churches, T., Hegland, M., 2004. FEBRL – a parallel open source data linkage system. In: Proceedings of PAKDD (LNAI 3056). Springer, pp. 638–647.
Costa, A.M., Cordeau, J.F., Laporte, G., forthcoming. Models and branch-and-cut algorithms for the Steiner tree problem with revenues, budget and hop constraints. Networks.
Gouveia, L., 1999. Using hop-indexed models for constrained spanning and Steiner tree models. In: Sansò, B., Soriano, P. (Eds.), Telecommunications Network Planning. Kluwer, Boston, pp. 21–32.
Heuser, C., Krieser, F.N.A., Orengo, V.M., 2007. Simeval – a tool for evaluating the quality of similarity functions. In: Grundy, J., Laender, A.H.F., Maciaszek, L., Roddick, J.F. (Eds), Twenty-Sixth International Conference on Conceptual Modeling – ER 2007 – Tutorials, Posters, Panels and Industrial Contributions, Australian Computer Society, Auckland, New Zealand.
Levenshtein, V., 1966. Binary codes capable of correcting deletions, insertions and reversals. Soviet Physics Doklady 10 (8), 707–710.
Lopresti, D., Tomkins, A., 1997. Block edit models for approximate string matching. Theoretical Computer Science 181, 159–179.
Mergen, S., Heuser, C., 2005. Carla: Uma técnica para comparação de cadeias de caractéres. I Escola Regional de Banco de Dados. SBC, Porto Alegre. pp. 55–70.
Navarro, G., 2001. A guided tour to approximate string matching. ACM Computing Surveys 33 (1), 31–88.
Tichy, W.F., 1984. The string-to-string correction problem with block moves. ACM Transactions on Computer Systems 2 (4), 309–321.