Dictionary-Symbolwise Flexible Parsing

Maxime Crochemore[1,4], Laura Giambruno[2], Alessio Langiu[2,4], Filippo Mignosi[3], and Antonio Restivo[2]

[1] Dept. of Computer Science, King's College London, London WC2R 2LS, UK
maxime.crochemore@kcl.ac.uk

[2] Dipartimento di Matematica e Informatica, Università di Palermo, Palermo, Italy
{lgiambr,alangiu,restivo}@math.unipa.it

[3] Dipartimento di Informatica, Università dell'Aquila, L'Aquila, Italy
filippo.mignosi@di.univaq.it

[4] Institut Gaspard-Monge,Université Paris-Est, France

Abstract. Linear time optimal parsing algorithms are very rare in the dictionary based branch of the data compression theory. The most recent is the *Flexible Parsing* algorithm of Mathias and Shainalp that works when the dictionary is prefix closed and the encoding of dictionary pointers has a constant cost. We present the *Dictionary-Symbolwise Flexible Parsing* algorithm that is optimal for prefix-closed dictionaries and any symbolwise compressor under some natural hypothesis. In the case of LZ78-alike algorithms with *variable costs* and any, linear as usual, symbolwise compressor can be implemented in linear time. In the case of LZ77-alike dictionaries and any symbolwise compressor it can be implemented in $O(n \log(n))$ time. We further present some experimental results that show the effectiveness of the dictionary-symbolwise approach.

1 Introduction

In [16] Mathias and Shainalp gave a linear time optimal parsing algorithm in the case of dictionary compression where the dictionary is prefix closed *and* the cost of encoding dictionary pointer is constant. They called their parsing algorithm *Flexible Parsing.* The basic idea of *one-step-lookahead parsing* that is at the base of flexible parsing was firstly used to our best knowledge in [6] in the case of dictionary compression where the dictionary is prefix closed *and* the cost of encoding dictionary pointer is constant *and* the dictionary is static. A first intuition, not fully exploited, that this idea could be successful used in the case of dynamic dictionaries, was given in [7] and also in [11], where it was called MTPL parsing (maximum two-phrase-length parsing).

Optimal parsing algorithms are rare and linear time optimal parsing results are rather rare. We can only also cite the fact that greedy parsing is optimal and linear for LZ77-alike dictionaries and constant cost dictionary pointers (see [19]) and its generalization to suffix closed dictionaries and constant cost dictionary pointers (see [2]) later used also in [11].

In this paper we consider the case of a free mixture of a dictionary compressor and a symbolwise compressor and we extend, in a non obvious way, the result of Mathias and Shainalp. We have indeed an optimal parsing algorithm in the case of dictionary-symbolwise compression where the dictionary is prefix closed and the cost of encoding dictionary pointer is *variable* and the symbolwise is any classical one that works in linear time. Our algorithm works under the assumption that a special graph that will be described in next section is well defined. Even in the case where this condition is not satisfied it is possible to use the same method to obtain almost optimal parses. In particular, when the dictionary is LZ78-alike our algorithm can be implemented in linear time and when the dictionary is LZ77-alike our algorithm can be implemented in time $O(n \log(n))$.

The study of free mixtures of two compressor is quite involved and it represents a new theoretical challenge. Free mixture has been implicitly or explicitly using for a long time in many fast and effective compressors such as gzip (see [5]), PkZip (see [10]) and Rolz algorithms (see [14]). For a quick look to compression performance on texts see Mahony challenge's page (see [13]).

So, why linear time optimal parsing algorithms are rather rare? Classically (see for example [18]), for static dictionaries it is possible to associate to any dictionary algorithm $\mathcal{A}$ and to any text T a weighted graph $G_{\mathcal{A},T}$ such that there is a bijection between optimal parsings and minimal paths in this graph. The extension of this approach to dynamical dictionaries has been firstly studied, to our best knowledge, in [17] and it has also been later used in [4]. More details will be given in next sections.

The graph $G_{\mathcal{A},T}$ is a Directed Acyclic Graph and it is possible to find a minimal path in linear time with respect to the size of it (see [3]). Unfortunately the size of the graph can be quadratic in the size of the text and this approach was not recommended in [18], because it is too time consuming. From a philosophical point of view, the graph $G_{\mathcal{A},T}$ represents a mathematical modelling of the optimal parsing problem. Thus, finding an optimal parsing in linear time corresponds to discovering a strategy for using only a subgraph of linear size. Indeed, in order to get over the quadratic worst case problem, there are many different approaches and many papers deal with optimal parsing in dictionary compressions. For instance the reader can see [1, 2, 4, 6, 8, 9, 11, 12, 15, 16, 19, 20]. Among them, we stress [4] where it is shown that a minimal path can be obtained by using a subgraph of $G_{\mathcal{A},T}$ of size $O(n\ log(n))$, in the LZ77 case under some natural assumptions on the cost function, by exploiting the discreteness of the cost functions.

In this paper we use a similar strategy, i.e. we consider static or dynamical dictionaries, following the approach of [17] and we discover a "small" subgraph of $G_{\mathcal{A},T}$ that is linear in the size of the text for LZ78-alike dictionaries and $O(n \log(n))$ for LZ77-alike dictionaries. This "small" subgraph is such that any minimal path in it is also a minimal path in $G_{\mathcal{A},T}$.

In Sect. 2 we recall some literature notions about dictionary and dictionary-symbolwise compression algorithms and we define the graph $G_{\mathcal{A},T}$. In Sect. 3

we formalize the definition of optimal algorithm and optimal parsing and extend them to the dictionary-symbolwise domain. In Sect. 4 we present the *Dictionary-Symbolwise Flexible Parsing*, a parsing algorithm that extends in some sense the *Flexible Parsing* (see [16]). We prove its optimality by showing that it corresponds to a shortest path in the full graph, and in Sect. 5 we describe some data structures that can be used for our algorithm in the two main cases of LZ78-alike and LZ77-alike dictionaries together the time analysis. Section 6 reports some experiments, open problems and our conclusions.

2 Preliminaries

Analogously to what stated in [17] and extending the approach introduced in [18] to the dynamical dictionary case, we show how it is possible to associate a directed weighted graph $G_{\mathcal{A},T} = (V, E, L)$ to any dictionary compression algorithm $\mathcal{A}$, any text $T = a_1 a_2 a_3 \cdots a_n$ and any cost function $C : E \to \mathbb{R}^+$ in the following way. The set of vertices is $V = \{0, 1, \ldots, n\}$, where vertex i corresponds to a_i, i.e. the i-th character in the text T, for $1 \leq i \leq n$ and vertex 0 correspond to the position at the beginning of the text, before any characters. The empty word ϵ is associated to vertex 0 that is also called the *origin* of the graph. The set of directed edges is $E = \{(p, q) \subset (V \times V) \mid p < q \text{ and } \exists\, w = T[p+1 : q] \in D_p\}$, where $T[p + 1 : q] = a_{p+1} a_{p+2} \cdots a_q$ and D_p is the dictionary relative to the processing step p-th, i.e. the step in which the algorithm either has processed the input text up to character a_p, for $0 < p$, or it has to begin, for $p = 0$. For each edge (p, q) in E, we say that (p, q) is *associated to* the dictionary phrase $w = T[p+1 : q] = a_{p+1} \cdots a_q \in D_p$. In the case of static dictionary D_i is constant along the algorithm steps, i.e. $D_i = D_j, \forall i, j = 0 \cdots n$. L is the set of edge labels $L_{p,q}$ for every edge $(p, q) \in E$, where the label $L_{p,q}$ is defined as the cost of the edge (p, q) when the dictionary D_p is in use, i.e. $L_{p,q} = C((p, q))$. When $L_{p,q}$ is always defined for each edge of the graph we say that $G_{\mathcal{A},T}$ is *well defined*.

A *dictionary-symbolwise* algorithm is a compression algorithm that uses both dictionary and symbolwise compression methods. Such compressors parse the text as a free mixture of dictionary phrases and literal characters, which are substituted by the corresponding pointers or literal codes, respectively. Therefore, the description of a dictionary-symbolwise algorithm should also include the so called *flag information*, that is the technique used to distinguish the actual compression method (dictionary or symbolwise) used for each segment or factor of the parsed text. Often, as in the case of LZSS (see [19]), an extra bit is added either to each pointer or encoded character to distinguish between them. Encoded information flag can require less (or more) space than one bit.

For instance, a dictionary-symbolwise compression algorithm with a fixed dictionary $D = \{ab, cbb, ca, bcb, abc\}$ and the static symbolwise codeword assignment $[a = 1, b = 2, c = 3]$ could compress the text *abccacbbabbcbcbb* as $F_d 1 F_s 3 F_d 3 F_d 2 F_d 1 F_d 4 F_d 2$, where F_d is the information flag for dictionary pointers and F_s is the information flag for the symbolwise code.

More formally, a parsing of a text T in a dictionary-symbolwise algorithm is a pair $(parse, Fl)$ where *parse* is a sequence $(u_1, \cdots, u_s)$ of words such that $T = u_1 \cdots u_s$ and where Fl is a boolean function that, for $i = 1, \ldots, s$ indicates whether the word u_i has to be coded as a dictionary pointer or as a symbol. See Tab. 1 for an example of dictionary-symbolwise compression.

A dictionary-symbolwise compression algorithm, analogously as done in [1] for the pure dictionary case, is specified by:

1. The dictionary description.
2. The encoding of dictionary pointers.
3. The symbolwise encoding method.
4. The encoding of the flag information.
5. The parsing method.

We can naturally extend the definition of the graph associated to an algorithm for the dictionary-symbolwise case. Given a text $T = a_1 \ldots a_n$, we denote by $T[i:j]$ the factor of T equal to $a_i \ldots a_j$. Given a dictionary-symbolwise algorithm $\mathcal{A}$, a text T and a cost function C defined on edges, the graph $G_{\mathcal{A},T} = (V, E, L)$ is defined as follows. The vertexes set is $V = \{0 \cdots n\}$, with $n = |T|$. The set of directed edges $E = E_d \bigcup E_s$, where $E_d = \{(p,q) \subset (V \times V) \mid p < q-1, \text{ and } \exists w = T[p+1:q] \in D_p\}$ is the set of dictionary edges and $E_s = \{(q-1,q) \mid 0 < q \leq n\}$ is the set of symbolwise edges. L is the set of edge labels $L_{p,q}$ for every edge $(p,q) \in E$, where the label $L_{p,q} = C((p,q))$. Let us notice that the cost function C hereby used has to include the cost of the flag information to each edge, i.e. either $C(p,q)$ is equal to $\langle$ the cost of the encoding of $F_d\rangle$ + $\langle$ the cost of the encoded dictionary phrase $w \in D_p$ associated to the edge $(p,q)\rangle$ if $(p,q) \in E_d$ or $C(p,q)$ is equal to $\langle$ the cost of encoded $F_s\rangle$ + $\langle$ the cost of the encoded symbol $a_q\rangle$ if $(p,q) \in E_s$. Moreover, since E_d does not contain edges of length one by definition, $G_{\mathcal{A},T} = (V, E, L)$ is not a multigraph. Since this graph approach can be extended to multigraph, with a overhead of formalism, one can relax the $p < q-1$ constrain in the definition of E_d to $p \leq q-1$. All the results we will state in this paper, naturally extend to the multigraph case.

We call *dictionary-symbolwise scheme* a set of algorithms having in common the same first four specifics (i.e. they differ one each other for just the parsing methods). A scheme does not need to contain *all* algorithms having the same first four specifics. We notice that any of the specifics from 1 to 5 above can depend on all the others, i.e. they can be mutually interdependent. Fixed a *dictionary-symbolwise scheme*, whenever the specifics of the parsing method

Input	ab	c	ca	cbb	ab	bcb	cbb
Output	F_d1	F_s3	F_d3	F_d2	F_d1	F_d4	F_d2

Table 1. Example of compression for the text *abccacbbabbcbcbb* by a simple Dyctionary-Symbolwise algorithm that use $D = \{ab, cbb, ca, bcb, abc\}$ as static dictionary, the identity as dictionary encoding and the mapping $[a = 1, b = 2, c = 3]$ as symbolwise encoding.

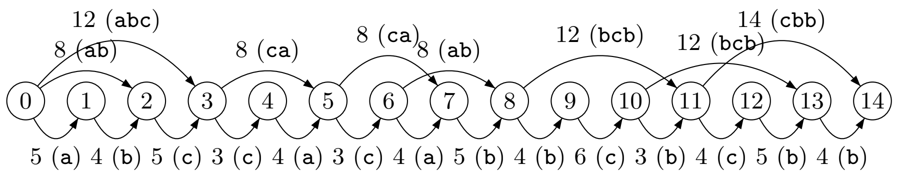
12 (abc)
8 (ab)
8 (ca)
8 (ca)
8 (ab)
12 (bcb)
12 (bcb)
14 (cbb)
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
5 (a) 4 (b) 5 (c) 3 (c) 4 (a) 3 (c) 4 (a) 5 (b) 4 (b) 6 (c) 3 (b) 4 (c) 5 (b) 4 (b)

Fig. 1. Graph $G_{\mathcal{A},T}$ for the text $T =$ `abccacabbcbcbb`, for the dictionary-symbolwise algorithm $\mathcal{A}$ with static dictionary $D = \{$`ab`, `abc`, `bcb`, `ca`, `cbb`$\}$ and cost function C as defined in the graph. The dictionary phrase or the symbol associated to an edge is reported near the edge label within parenthesis.

are given, exactly one algorithm is completely described. Notice that the word *scheme* has been used by other authors with other related meaning. For us the meaning is rigorous.

3 On Optimality

In this section we assume that the reader is familiar with LZ-alike dictionary encoding and with some simple statistical encodings such as the Huffman encoding.

Definition 1. *Fixed a dictionary description, a cost function C and a text T, a* dictionary (dictionary-symbolwise) algorithm *is* optimal *within a set of algorithms if the cost of the encoded text is minimal with respect to all others algorithms in the same set. The parsing of an* optimal *algorithm is called* optimal *within the same set.*

When the length in bit of the encoded dictionary pointers is used as cost function, the previous definition of optimality is equivalent to the classical well known definition of bit-optimality for dictionary algorithm. Notice that the above definition of optimality strictly depends on the text T and on a set of algorithms. A parsing can be optimal for a text and not for another one. Clearly, we are mainly interested on parsings that are optimal either for *all* texts over an alphabet or for classes of texts. Whenever it is not explicitly written, from now on when we talk about optimal parsing we mean optimal parsing for *all* texts. About the set of algorithm it make sense to find sets as large as possible.

Classically, there is a bijective correspondence between parsings and paths in $G_{\mathcal{A},T}$ from vertex 0 to vertex n, where optimal parses correspond to minimal paths and vice-versa. We say that a parse (resp. path) *induces* a path (resp. parse) to denote this correspondence. This correspondence was firstly stated in [18] only in the case of sets of algorithms sharing the same *static* dictionary and where the encoding of pointers has constant cost.

For example the path along vertexes $(0, 3, 4, 5, 6, 8, 11, 12, 13, 14)$ is the shortest path for the graph in Fig. 1. Authors of [17] were the firsts to formally extend the Shortest Path approach to dynamically changing dictionaries and variable costs.

Definition 2. *A scheme $\mathcal{S}$ has the* Schuegraf property *if, for any text T and for any pair of algorithms $\mathcal{A}, \mathcal{A}' \in \mathcal{S}$, the graph $G_{\mathcal{A},T} = G_{\mathcal{A}',T}$ with $G_{\mathcal{A},T}$ well defined.*

This property of schemes is called *property of Schuegraf* in honor to the first of the authors in [18]. In this case we define $G_{\mathcal{S},T} = G_{\mathcal{A},T}$ as the graph of (any algorithm of) the scheme. The proof of the following proposition is straightforward.

Proposition 1. *There is a bijective correspondence between optimal parsings and shortest paths in $G_{\mathcal{S},T}$ from vertex 0 to vertex n.*

Definition 3. *Let us consider an algorithm $\mathcal{A}$ and a text T and suppose that graph $G_{\mathcal{A},T}$ is well defined. We say that $\mathcal{A}$ is* dictionary *optimal (with respect to T) if its parsing induces a shortest path in $G_{\mathcal{A},T}$ from the origin (i.e. vertex 0) to vertex n, with $n = |T|$. In this case we say that its parsing is* dictionary *optimal.*

Let $\mathcal{A}$ be an algorithm such that for any text T the graph $G_{\mathcal{A},T}$ is well defined. We want to associate to it a scheme $\mathcal{SC}_A$ in the following way. Let S be the set of all algorithms $\mathcal{A}$ such that for any text T $G_{\mathcal{A},T}$ exists (i.e. it is well defined). Let $\mathcal{B}$ and $\mathcal{C}$ two algorithms in S. We say that $\mathcal{B}$ and $\mathcal{C}$ are equivalent or $\mathcal{B} \equiv \mathcal{C}$ if, for any text T, $G_{\mathcal{B},T} = G_{\mathcal{C},T}$.

We define the scheme $\mathcal{SC}_A$ to be the equivalence class that has $\mathcal{A}$ as a representative. It is easy to prove that $\mathcal{SC}_A$ has the Schuegraf property.

We can connect the definition of *dictionary* optimal parsing with the previous definition of $\mathcal{SC}_A$ to obtain the next proposition, that says, roughly speaking, that dictionary optimality implies scheme (or global) optimality within the scheme $\mathcal{SC}_A$.

Proposition 2. *Let us consider an algorithm $\mathcal{A}$ such that for any text T the graph $G_{\mathcal{A},T}$ is well defined. Suppose further that for a text T the parsing of $\mathcal{A}$ is dictionary optimal. Then the parsing of $\mathcal{A}$ of the text T is (globally) optimal within the scheme $\mathcal{SC}_A$.*

We have simple examples where a parsing of a text is dictionary optimal and the corresponding algorithm belongs to a scheme that has not the Schuegraf property and it is not (globally) optimal within the same scheme. For pure dictionary scheme having the Schuegraf Property we mean a dictionary-symbolwise scheme having the Schuegraf Property where all algorithms in the scheme are pure dictionary. We have to be a bit careful using this terminology. Indeed, LZ78, LZW, LZ77 and related algorithms often parse the text with a dictionary pointer and then add a symbol, i.e. the parse phrase is composed by a dictionary pointer and a symbol. In these cases all edges of $G_{\mathcal{A},T}$ denote parse phrases coupled to the corresponding symbol. Edges are labeled by the cost of the dictionary pointer plus the cost of the symbol. We consider these cases included in the class of "pure dictionary" algorithms and schemes.

4 Dictionary-Symbolwise Flexible Parsing Algorithm

In this section we extend the notion of flexible parsing to the dictionary-symbolwise case and we prove that it is still optimal within any scheme having the Schuegraf Property. We assume here that the dictionary must be at any moment prefix closed. The algorithm is quite different from the original Flexible Parsing but it has some analogies with it and, in the case of LZ78-alike dictionaries, it makes use of one of the main data structures used for the original flexible parsing in order to be implemented in linear time. Concerning the costs of encoding pointers, we recall that costs can vary but that they assume positive values and that they include the cost of flag information. Concerning the symbolwise compressor, the costs of symbols must be positive, including the flag information cost. They can vary depending on the position of the character in the text and on the symbol itself. We suppose further that a text T of length n is fixed and that we are considering the graph $G_{\mathcal{A},T}$, where $\mathcal{A}$ is a dictionary-symbolwise algorithm, and $G_{\mathcal{A},T}$ is well defined under our assumption. We denote by d the function that represent the distance of the vertexes of $G_{\mathcal{A},T}$ from the origin of the graph. Such a distance $d(i)$ is classically defined as the minimal cost of all possible weighted paths from the origin to the vertex i, where $d(0) = 0$. This distance obviously depends on the cost function. We say that cost function C is *prefix-non-decreasing* at any moment if for any $u, v \in D_p$ phrases associated to edges $(p, i), (p, q)$, with $p < i < q$, that implies that u is prefix of v, one has that $C((p, i)) \leq C((p, q))$.

Lemma 1. *Let $\mathcal{A}$ be a dictionary-symbolwise algorithm such that for any text T the graph $G_{\mathcal{A},T}$ is well defined. If the dictionary is always (at any moment) prefix-closed and if the cost function is always (at any moment) prefix-non-decreasing then the function d is non-decreasing monotone.*

In what follows in this paper we suppose that the graph $G_{\mathcal{A},T}$ is well defined. Let us call vertex j a *predecessor* of vertex $i \iff \exists (j, i) \in E$ such that $d(i) = d(j) + C((j, i))$. Let us define $pre(i)$ to be the smallest of the predecessors of vertex $i, 0 < i \leq n$, that is $pre(i) = \min\{j \mid d(i) = d(j) + C((j, i))\}$. In other words $pre(i)$ is the smallest vertex j that contributes to the definition of $d(i)$. Clearly $pre(i)$ has distance smaller than $d(i)$. Moreover the function pre is not necessarily injective. For instance, a vertex can be a predecessor either "via" a dictionary edge or "via" a symbol edge. It is also possible to extend previous definition to pointers having a cost smaller than or equal to a fixed c.

Definition 4. *For any cost c we define $pre_c(i) = \min\{j \mid d(i) = d(j) + C((j, i))$ and $C((j, i)) \leq c\}$. If none of the predecessor j of i is such that $C((j, i)) \leq c$ then $pre_c(i)$ is undefined.*

If all costs of the pointers are smaller than or equal to c then for any i one has obviously that $pre_c(i)$ is equal to $pre(i)$.

Analogously to the notation of [15], we want to define two boolean operations *Weighted-Extend* and *Weighted-Exist*.

Definition 5. *Given an edge (i, j) in $G_{\mathcal{A},T}$ and its associated phrase w, a cost value c and a character 'a', the operation* Weighted-Extend$((i, j), a, c)$ *finds out whether the word wa is a phrase in D_i having cost smaller than or equal to c.*

More formally, let (i, j) be such that $w = T[i+1 : j] \in D_i$ and, then, (i, j) is in $G_{\mathcal{A},T}$. *Weighted-Extend*$((i, j), a, c) =$ *"yes"* $\iff$ $wa = T[i + 1 : j + 1] \in D_i$ *and* $C((i, j + 1)) \leq c$, where C is the cost function associated to the algorithm $\mathcal{A}$. Otherwise *Weighted-Extend*$((i, j), a, c) =$ *"no"*.

Definition 6. *Given $0 < i, j \leq n$ and a cost value c, the operation* Weighted-Exist(i, j, c) *finds out whether or not the phrase $w = T[i + 1 : j]$ is in D_i and the cost of the corresponding edge (i, j) is smaller than or equal to c.*

Let us notice that doing successfully a *Weighted-Extend* operation on $((i, j), a,$ $c)$ means that $wa \in D_i$ is the weighted extension of w and the encoding of $(i, j+1)$ has cost less or equal to c. Similarly, doing a *Weighted-Exist* operation on (i, j, c) means that an edge (i, j) exist in $G_{\mathcal{A},T}$ having cost less or equal to c.

Let E_c be the subset of all edges of the graph having cost smaller than or equal to c.

Definition 7. *Let us also define, for any cost c the set $M_c \subseteq E_c$ to be the set of* c-supermaximal *edges, where $(i, j) \in M_c \iff (i, j) \in E_c$ and $\forall p, q \in V$, with $p < i$ and $j < q$, the arcs $(p, j), (i, q)$ are not in E_c. For any $(i, j) \in M_c$ let us call i a c-starting point and j a c-ending point.*

Proposition 3. *Suppose that (i, j) and (i', j') are in M_c. One has that $i < i'$ if and only if $j < j'$.*

By previous proposition, if $(i, j) \in M_c$ we can think j as function of i and conversely. Therefore it is possible to represent M_c by using an array $M_c[j]$ such that if (i, j) is in M_c then $M_c[j] = i$ otherwise $M_c[j] = Nil$. Moreover the non-*Nil* values of this array are strictly increasing. The positions j having value different from *Nil* are the ending positions.

We want now to describe a simple algorithm that outputs all *c-supermaximal* edges scanning the text left-to-right. We call it *Find Supermaximal*(c). It uses the operations *Weighted-Extend* and *Weighted-Exist*. The algorithm starts with $i = 0, j = 1$ and $w = a_1$.The word w is indeed implicitely defined by the arc (i, j) and therefore it will not appear explicitely in the algorithm. At each step j is increased by one and w is set to w concatenated to $T[j]$. The algorithm executes a series of *Weighted-Extend* until this operation give a positive answer or the end of the text is reached. After a negative answer of *Weighted-Extend*, the algorithm does a series of *Weighted-Exist* increasing i by one until a positive answer. The algorithm is stated more formally in the following pseudo code.

```
FIND SUPERMAXIMAL ($c$)
01.  $i \leftarrow 0, j \leftarrow 1$
02.  WHILE $j < n$
03.  DO
04.      WHILE $\textit{Weighted-Extend}((i,j), a_{j+1}, c) =$ "yes" AND $j < n$
05.      DO
06.          $j \leftarrow j + 1$
07.      INSERT $(i,j)$ in $M_c$, $j \leftarrow j + 1$
08.      DO
09.          $i \leftarrow i + 1$
10.      WHILE $\textit{Weighted-Exist}(i,j,c) =$ "no" AND $i < j$
```

We notice that when exiting from cycle of lines $4-6$, the cost of the edge (i,j) could still be strictly smaller than c. The function INSERT simply insert the edge (i,j) in the dynamical set M_c. If we represent M_c by an array as described after Prop. 3, function INSERT sets $M_c[j]$ equal to i. Array $M_c[j]$ was initialized by setting all its entries to *Nil*.

Proposition 4. *Above algorithm correctly computes M_c.*

Proposition 5. *For any edge $(i,j) \in E_c$ there exists a* c-supermaximal *edge $(\hat{\imath}, \hat{\jmath})$ containing it, i.e. such that $\hat{\imath} \le i$ and $j \le \hat{\jmath}$.*

By previous proposition for any node $v \in G_{\mathcal{A},T}$ if there exists a node $i < v$ such that $C((i,v)) = c$ and $d(v) = d(i) + c$ then there exists a *c-supermaximal* edge $(\hat{\imath}, \hat{\jmath})$ containing (i,v) and such that $\hat{\jmath}$ is the *closest* arrival point greater that v. Let us call this *c-supermaximal* edge $(\hat{\imath}_v, \hat{\jmath}_v)$. We use $\hat{\imath}_v$ in next proposition.

Proposition 6. *Suppose that $v \in G_{\mathcal{A},T}$ is such that there exists a previous node i such that $C((i,v)) = c$ and $d(v) = d(i) + c$. Then $\hat{\imath}_v$ is a predecessor of v, i.e. $d(v) = d(\hat{\imath}_v) + C((\hat{\imath}_v, v))$ and, moreover, $d(\hat{\imath}_v) = d(i)$ and $C((\hat{\imath}_v, v)) = c$.*

Corollary 1. *For any vertex v, the edge $(\hat{\imath}_v, v)$ is the last edge of a path of minimal cost from the origin to vertex v.*

In what follows we describe a graph $G'_{\mathcal{A},T}$ that is a subgraph of $G_{\mathcal{A},T}$ and that is such that for any node $v \in G_{\mathcal{A},T}$ there exists a minimal path from the origin to v in $G'_{\mathcal{A},T}$ that is also a minimal path from the origin to v in $G_{\mathcal{A},T}$. The proof of this property, that will be stated in the subsequent proposition, is a consequence of Proposition 6 and Corollary 1.

We describe the building of $G'_{\mathcal{A},T}$ in an algorithmic way. Even if we do not give the pseudocode, algorithm BUILD $G'_{\mathcal{A},T}$ is described in a rigorous way and it makes use, as a part of it, of algorithm FIND SUPERMAXIMAL.

The set of nodes of $G'_{\mathcal{A},T}$ is the same of $G_{\mathcal{A},T}$. First of all we insert all symbolwise edges of $G_{\mathcal{A},T}$ in $G'_{\mathcal{A},T}$. Let now $\mathcal{C}$ be the set of all possible costs that any dictionary edge has. This set can be build starting from $G_{\mathcal{A},T}$ but in all known meaningful situations the set $\mathcal{C}$ is usually well known and can be ordered and stored in an array in a time that is linear in the size of the text.

For any $c \in \mathcal{C}$ we use algorithm FIND SUPERMAXIMAL to obtain the array $M_c[j]$. For any *c-supermaximal* edge (i, j), we add in $G'_{\mathcal{A},T}$ all edges of the form (i, x) where x varies from j down to (and not including) the previous arrival position j' if this position is greater than $i + 1$ otherwise down to $i + 2$. More formally, for any j such that $M_c[j] \neq Nil$ let j' be the greatest number smaller than j such that $M_c[j] \neq Nil$. For any x, such that $\max(j', i + 2) \leq x \leq j$, add (i, x) to $G'_{\mathcal{A},T}$ together with its label. This concludes the construction of $G'_{\mathcal{A},T}$.

Since (i, j) and the dictionary D_i is prefix closed then all previous arcs of the form (i, x) are also arcs of $G_{\mathcal{A},T}$ and, therefore, $G'_{\mathcal{A},T}$ is a subgraph of $G_{\mathcal{A},T}$.

Proposition 7. *For any node $v \in G_{\mathcal{A},T}$ there exists a minimal path from the origin to v in $G'_{\mathcal{A},T}$ that is also a minimal path from the origin to v in $G_{\mathcal{A},T}$.*

We can now finally describe the *Dictionary-symbolwise flexible parsing.*

The *Dictionary-symbolwise flexible parsing* firstly uses algorithm BUILD $G'_{\mathcal{A},T}$ and then uses the classical SINGLE SOURCE SHORTEST PATH algorithm to recover a minimal path from the origin to the end of graph $G_{\mathcal{A},T}$. The correctness of above algorithm is stated in the following theorem and it follows from above description and from Prop. 7.

Theorem 1. Dictionary-symbolwise flexible parsing *is dictionary optimal.*

With respect to the original Flexible Parsing algorithm we gain the fact that it can works with variable costs of pointers and that it is extended to the dictionary-symbolwise case. But we loose the fact that the original one was "on-line". A minimal path has to be recovered, starting from the end of the graph backward. But this is an intrinsic problem that cannot be eliminated. Even if the dictionary edges have just one possible cost, in the dictionary-symbolwise case it is possible that any minimal path for a text T is totally different from any minimal path for the text Ta, that is the previous text T concatenated to the symbol 'a'. Even if the cost of pointers is constant. The same can happen when we have a "pure dictionary" case with variable costs of dictionary pointers. In both cases for this reason, there cannot exists "on-line" optimal parsing algorithms, and, indeed, flexible parsing fails being optimal in the pure dictionary case when costs are variable.

On the other hand our algorithm is suitable when the text is cut in several blocks and, therefore, in practice there is not the need to process the whole text but it suffices to end the current block in order to have the optimal parsing (relative to that block). As another alternative, it is possible to keep track of just one minimal path all along the text and to use some standard tricks to arrange it if it does not reach the text end, i.e. the wished target node. In the last cases one get a suboptimal solution that is a path with a cost extremely close to the minimal path.

5 Data Structures and Time Analysis

In this subsection we analyze *Dictionary-symbolwise flexible parsing* in both LZ78 and LZ77-alike algorithms.

Concerning LZ78-alike algorithms, the dictionary is prefix closed and it is usually implemented by using a technique that is usually referred as LZW implementation. We do not enter in details of this technique. We just recall that the cost of pointers increases by one unit whenever the dictionary size is "close" to a power of 2. The moment when the cost of pointers increases is clear to both encoder and decoder. In our dictionary-symbolwise setting, we suppose that the flag information for dictionary edges is constant. We assume therefore that it takes $O(1)$ time to determine the cost of all dictionary edges outgoing node i.

The maximal cost that a pointer can assume is smaller than $\log_2(n)$ where n is the text-size. Therefore the set $\mathcal{C}$ of all possible costs of dictionary edges has logarithmic size and it is cheap to calculate.

In [15] it is used a data structure, called trie-reverse pair, that is able to perform the operation of *Extend* and *Contract* in $O(1)$ time.

Since at any position we can calculate in $O(1)$ time the cost of outgoing edges, we can use the same data structure to perform our operations of *Weighted-Extend* and of *Weighted-Exist* in constant time. In order to perform a *Weighted-Extend* we simply use the *Extend* on the same non-weight parameters and, if the answer is "yes" we perform a further check in $O(1)$ time on the cost. In order to perform a *Weighted-Exist* we simply use the *contract* on the same non-weight parameters and, if the answer is "yes" we perform a further check in $O(1)$ time on the cost.

For any cost c finding M_c and the corresponding arcs in order to build $G'_{\mathcal{A},T}$ takes then linear time. Therefore, at a first look, performing the algorithm Build $G'_{\mathcal{A},T}$ would take $O(n\ log(n))$. But, since there is only one cost active at any position then if $c < c'$ then $M_c \subseteq M_{c'}$ as stated in the following proposition.

Definition 8. *We say that a cost function C is* LZW-alike *if for any i the cost of all dictionary pointers in D_i is a constant c_i and that for any i, $0 \leq i < n$ one has that $c_i \leq c_{i+1}$.*

Proposition 8. *If the cost funtcion C is* LZW-alike*, one has that if $c < c'$ then $M_c \subseteq M_{c'}$.*

At this point, in order to build $G'_{\mathcal{A},T}$ it suffices to build M_c where c is the greatest possible cost. Indeed it is useless checking for the cost and one can just use the standard operation *Extend* and *Contract.* Those operation can be implemented in $O(1)$ time using the trie reverse trie data structure for LZ78 standard dictionary or for the LZW dictionary or for the FPA dictionary (see [15]). Indeed we call a dictionary *LZ78-alike* if the operations *Extend* and *Contract* can be implemented in $O(1)$ time using the trie reverse trie data structure.

We notice that previous definition of LZ78-alikeness can be relaxed by asking that the operations *Extend* and *Contract* can be implemented in $O(1)$ amortized time using any data structure, including obviously the time used for building such data structure.

The overall time for building $G'_{\mathcal{A},T}$ is therefore linear, as well as its size. The Single Source Shortest Path over $G'_{\mathcal{A},T}$, that is a DAG topologically ordered, takes linear time.

In conclusion we state the following theorem.

Theorem 2. *Suppose that we have a dictionary-symbolwise scheme, where the dictionary is LZ78-alike and the cost function is LZW-alike. The symbolwise compressor is supposed to be, as usual, linear time. Using the trie-reverse pair data structure,* Dictionary-Symbolwise flexible parsing *is linear.*

Concerning LZ77, in [4] it has been given, with a similar shortest path approach, an optimal parsing algorithm under some assumptions on the cost function. Our prefix-non-decreasing assumption is weaker than their assumptions in the sense that it is a consequence of their assumptions (see [4, Fact 4]). The maximal cost that a pointer can have under their assumption is still $O(\log(n))$ where n is the size of the text. It seems that it is possible to use the data structure used in [4] to perform, for any cost, *Weighted-Extend* and *Weighted-Exist* in amortized $O(1)$ time. Then the overall time for the *Dictionary-symbolwise flexible parsing* when the dictionary is LZ77-alike would be $O(n\log(n))$, extending their result to the dictionary-symbolwise case. The subgraph $G'_{\mathcal{A},T}$ of $G_{\mathcal{A},T}$ is totally different from the one used in [4]. Indeed, quite recently, we discovered a simpler data structure that allows us to perform, for any cost, *Weighted-Extend* and *Weighted-Exist* in amortized $O(1)$ time. This data structure is built by using in a clever way $O(\log(n))$ suffix trees and it will be described in the journal version of this paper.

6 Conclusions

In this paper we present some advancement on dictionary-symbolwise theory. We describe the *Dictionary-Symbolwise Flexible Parsing*, a parsing algorithm that extends in non obvious way the *Flexible Parsing* (see [16]) to *variable* and *unbounded* costs and to the dictionary-symbolwise algorithm domain. We prove its optimality for prefix-closed dynamic dictionaries under some reasonable assumption. *Dictionary-Symbolwise Flexible Parsing* is *linear* for LZ78-alike dictionaries and even if it is not able to run online it allow to easily make a block programming implementation and a near to optimal online implementation, too. In the case of LZ77-alike dictionary, we have reobtained the $O(n\log(n))$ complexity as authors of [4] recently did and we use a completely different and simpler subgraph and a simpler data structure.

Our algorithm has therefore two advantages with respect to the classical Flexible Parsing. First, it can handle *variable cost* of dictionary pointers. This fact allows to extend the range of application of Flexible Parsing to almost all LZ78-alike known algorithms of our extension. Secondly, our Dictionary-Symbolwise Flexible Parsing implemented in the case of LZ77 dictionary gives as particular case when the symbolwise is not in use, a result that is similar to the one presented in [4] that has $O(n\ log(n))$ complexity, using a completely different and simpler subgraph and a simpler data structure. Last but not least our algorithm allows to couple classical LZ-alike algorithms with several symbolwise algorithms to obtain dictionary-symbolwise algorithms that achieve better compression with prove of optimality.

It is possible to prove, and we did not do it due to space limitation, that dictionary-symbolwise compressors can be asymptotically better than optimal pure dictionary compression algorithms in compression ratio terms, with LZ78 based dictionary and the same can be proved for LZ77 based dictionary. In our case, using a simple static Huffman coding as symbolwise compressor we improved the compression ratio of the *Flexible Parsing* of $7\% - 4\%$ on texts such as prefixes of English Wikipedia data base with a negligible slow down in compressing and decompressing time. The slow down comes from the fact that we have to add to the dictionary compression and decompression time the Huffman coding and decoding time. The same experimental result holds in general when the dictionary is LZ78-alike. Indeed a dictionary-symbolwise compressor when the dictionary is LZ78-alike and the symbolwise is a simple Huffman coding with optimal parsing has a compression ratio that is is more or less 5% better than the compression ratio of a pure LZ78-alike dictionary compressor that uses an optimal parsing. In general smaller is the file greater is the gain. The 5% refers to text sizes of around 20 megabytes. Moreover, preliminary results show that using more powerful but still fast symbolwise compressor, such as an arithmetic encoder of order 1, there is a further 10% gain in compression ratio. When the dictionary is, instead, LZ77-alike the gain in compression when we use a dictionary-symbolwise compressor with optimal parsing and Huffman coding with respect to a pure dictionary compressor with optimal parsing reduces down to more or less 3% over texts of 20 megabytes, smaller the file greater the gain. The compression ratio seems to be sensibly better than in the case of LZ78-alike dictionaries when we use, in both cases, unbounded dictionaries. The distance, however, between the compression ratio of dictionary-symbolwise compressors that use LZ78-alike dictionaries and the ones that use LZ77-alike dictionaries is smaller, following our preliminary results, when we use an arithmetic encoder of order 1 instead than an Huffman encoding. We have experimental evidence that many of the most relevant commercial compressors use, following our definition, optimal parsing in the dictionary-symbolwise case where the dictionary is LZ77-alike. The method described in this paper therefore has as a consequence the possibility of optimizing the trade-off between some of the main parameters used for evaluating commercial compressors, such as compression ratio, decompression time, compression time and so on. We plan to extend our experimentation on LZ-alike dictionary algorithms and many other symbolwise algorithms, since this direction seems to be very promising.

We conclude this paper with two open problems.

1. Theoretically, LZ78 is better on memoryless sources than LZ77. Experimental results say that when optimal parsing is in use it happens the opposite. Prove this fact both in pure dictionary case and in dictionary-symbolwise case.

2. Common symbolwise compressors are based on the arithmetic coding approach. When these compressors are used, the costs in the graph are almost surely non integer and, moreover, the graph is usually not well defined. The standard workaround is to use an approximation strategy. A big goal should be to find an optimal solution for these important cases.

References

1. Timothy C. Bell and Ian H. Witten. The relationship between greedy parsing and symbolwise text compression. *J. ACM*, 41(4):708–724, 1994.
2. Martin Cohn and Roger Khazan. Parsing with prefix and suffix dictionaries. In *Data Compression Conference*, pages 180–189, 1996.
3. Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, second edition, 2001.
4. Paolo Ferragina, Igor Nitto, and Rossano Venturini. On the bit-complexity of lempel-ziv compression. In *SODA '09: Proceedings of the Nineteenth Annual ACM -SIAM Symposium on Discrete Algorithms*, pages 768–777, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.
5. Gzip's Home Page: `http://www.gzip.org`.
6. Alan Hartman and Michael Rodeh. *Optimal parsing of strings*, pages 155–167. Springer - Verlag, 1985.
7. R. N. Horspool. The effect of non-greedy parsing in ziv-lempel compression methods. In *Data Compression Conference*, 1995.
8. Jyrki Katajainen and Timo Raita. An approximation algorithm for space-optimal encoding of a text. *Comput. J.*, 32(3):228–237, 1989.
9. Jyrki Katajainen and Timo Raita. An analysis of the longest match and the greedy heuristics in text encoding. *J. ACM*, 39(2):281–294, 1992.
10. Phil Katz. Pkzip archiving tool, `http://en.wikipedia.org/wiki/pkzip`, 1989.
11. Tae Young Kim and Taejeong Kim. On-line optimal parsing in dictionary-based coding adaptive. *Electronic Letters*, 34(11):1071–1072, 1998.
12. Shmuel T. Klein. Efficient optimal recompression. *Comput. J.*, 40(2/3):117–126, 1997.
13. Matt Mahoney. Large text compression benchmark: `http://mattmahoney.net/text/text.html`.
14. Christian Martelock. Rzm order-1 rolz compressor, `http://encode.ru/forums/index.php?action=vthread&forum=1&topic=647`, 4 2008.
15. Yossi Matias, Nasir Rajpoot, and S"uleyman Cenk Shainalp. The effect of flexible parsing for dynamic dictionary-based data compression. *ACM Journal of Experimental Algorithms*, 6:10, 2001.
16. Yossi Matias and S"uleyman Cenk shainalp. On the optimality of parsing in dynamic dictionary based data compression. In *SODA*, pages 943–944, 1999.
17. G. Della Penna, A. Langiu, F. Mignosi, and A. Ulisse. Optimal parsing in dictionary-symbolwise data compression schemes. http://www.di.univaq.it/ mignosi/ulicompressor.php, 2006.
18. Ernst J. Schuegraf and H. S. Heaps. A comparison of algorithms for data base compression by use of fragments as language elements. *Information Storage and Retrieval*, 10(9-10):309–319, 1974.
19. James A. Storer and Thomas G. Szymanski. Data compression via textural substitution. *J. ACM*, 29(4):928–951, 1982.
20. Robert A. Wagner. Common phrases and minimum-space text storage. *Commun. ACM*, 16(3):148–152, 1973.